EROS: Un servicio para los robots de descubrimiento de recursos en la World Wide Web

Tesis de Licenciatura

Director

Ing. Claudio Righetti

Tesistas

Mariela Sassi LU 803/87
Ariel Fabian LU 685/87

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Argentina

1998

TEMARIO

Resumen 5

Abstract 5

1. INTRODUCCION 6

1.1. Organización 8

2. EL PROBLEMA DE LA BUSQUEDA EN LA WEB 10

2.1. Indices 11

2.2. Servidores de catálogos 14

2.3. Servidores de índices 15

2.4. Comparación entre servidores de catálogos e índices 16

2.5. Problemas de escalabilidad 16

2.5.1. Tiempo de obtención de recursos 17

2.5.2. Generación de índices 17

2.5.3. Cuellos de botella 18

3. ROBOTS DE DESCUBRIMIENTO DE RECURSOS (ARAÑAS) 19

3.1. Robots 19

3.1.1. Definición 19

3.1.2. Recorrido de la Web 19

3.1.3. Método de obtención de recursos 22

3.2. Objetivos de los robots 22

3.3. Problemas causados por las arañas 24

3.4. Estándar de exclusión de robots 27

3.5. Alternativas al uso de arañas 30

3.5.1. Aliweb 30

3.5.2. Harvest 33
3.5.3. Robots cooperativos 37

3.6. Conclusiones 39

4. EROS: SERVICIO EXCLUSIVO PARA ROBOTS 41

4.1. Descripción del servicio 45

4.2. Protocolo 45

4.3. Mensaje de solicitud 46

4.4. Mensaje de respuesta 50

4.5. Implementación del servicio 51

4.5.1. El servidor 51
4.5.2. El sumarizador 52
4.5.3. Diseño del servidor 53

4.6. Comparaciones 55

4.6.1. Obtención de recursos 55
4.6.2. Generación de las sumarizaciones 58
4.6.3. Diferencias 60

5. UN ROBOT QUE UTILIZA EL SERVICIO EROS 63

5.1. El proceso Cliente 66

5.2. El motor de búsqueda 69

5.3. El proceso destino 71

5.4. Conclusiones 73

6. CONCLUSIONES Y TRABAJOS FUTUROS 74

6.1. Pedidos por servidor 74

6.2. Robots cooperativos 76

6.3. Servidor de búsqueda 76

6.4. Mejoras al servidor 77

Apéndice A: Internet 78

Apéndice B: La World Wide Web 91

Apéndice C: Lista de robots 111

Apéndice D: Manual del Usuario 118

BIBLIOGRAFIA 121

Resumen

Los robots de descubrimiento de recursos o *arañas* brindan un servicio valioso a los usuarios de la World Wide Web, ya que se encargan de recolectar los recursos de ésta con el objetivo de crear índices. Sin embargo, ocasionan serios problemas; incrementan el tráfico y sobrecargan a los servidores. La principal causa es que usan el protocolo HTTP para acceder a dichos recursos. Nuestra solución consiste en la realización de un servicio que atiende exclusivamente a las arañas y resuelve los problemas mencionados.

Abstract

Resource discovery robots or *spiders* offer a valuable service to the users of the World Wide Web, since they take charge of gathering it´s resources with the objective of creating indexes. However, they cause serious problems; they increase the traffic and they overload the servers. The main cause is that they use the HTTP protocol to access to this resources. Our solution consists on the realization of a service that assists the spiders exclusively and solves the mentioned problems.

1. INTRODUCCION

La *World Wide Web*, que suele ser abreviada como WWW o Web, es el servicio más usado dentro de Internet. Esto se debe principalmente a la posibilidad de publicar información de manera sencilla y a la variedad de recursos que se pueden acceder a través de ella.

En los comienzos de la Web, los usuarios encontraban la información deseada utilizando la técnica de *navegación*. Esto se hizo cada vez más difícil debido a su enorme crecimiento. Por ello, surgieron herramientas con el fin de guiar al usuario hacia la información solicitada. Estas se dividen principalmente en dos categorías: los *servidores de catálogos* y los *servidores de índices*.

Los servidores de catálogos proveen al usuario de un catálogo de recursos de la Web, organizado en una jerarquía de títulos, donde cada uno abarca un tema. Estos catálogos generalmente cubren una pequeña fracción de la Web.

En cambio, los servidores de índices, ofrecen una inmensa base de datos centralizada de recursos de la Web que puede ser consultada con el objetivo de encontrar documentos que contengan cierta información. La mayor cobertura por parte de los servidores de índices, es lo que hace que hoy en día sea la herramienta más usada para la búsqueda de información dentro de la Web.

Para poder generar sus índices, los servidores de índices utilizan programas que recorren la Web automáticamente y obtienen los recursos que van descubriendo. Estos programas reciben el nombre de *robots de descubrimiento de recursos o arañas*.

Las arañas proveen un servicio valioso. Son ellas las que se encargan de recolectar los recursos de la Web para luego crear un índice de ésta. Pero, por otro lado, ocasionan serios problemas: incrementan el tráfico en la red y en muchos casos sobrecargan a los servidores al accederlos reiteradamente en un corto intervalo de tiempo.

Desarrollar una araña no es una tarea muy complicada. No se necesita ser un experto para ello. Es así, que hoy en día, conviven arañas 'bien educadas' que se cuidan de no causar grandes problemas con otras que no lo son. El hecho de que existan arañas que causan problemas se basa principalmente en el desconocimiento del daño que se puede provocar o debido a que es mucho más sencillo realizar una araña que no haga ningún tipo de control que una que sí lo haga.

Para recuperar los recursos de la Web las arañas utilizan el protocolo *HTTP*, es decir, obtienen los recursos de la misma manera que lo hace un usuario a través de un *navegador*. Sin embargo el uso de HTTP por parte de las arañas es el principal causante de los problemas ocasionados por ellas. Por ejemplo, mediante HTTP las arañas no tienen forma de solicitar a los servidores los títulos de un documento; deben recuperarlo en su totalidad para luego quedarse con los títulos, desperdiciando así, más recursos de los que se deberían.

Los servidores de índices ofrecen por una parte un servicio gratuito a los usuarios y por otra espacios de publicidad a los anunciantes, con los que obtienen ganancias millonarias. Mientras sigan existiendo intereses económicos, se hace difícil que los servidores de índices dejen de lado al protocolo HTTP a través de las arañas, ya que esta es la única manera de recuperar los recursos de la Web sin tener que cambiar la infraestructura existente.

Existen alternativas que utilizan otros modelos para la realización de un índice de la Web. Estas se basan generalmente en una cooperación por parte de los servidores. Por diferentes motivos, ninguna ha sido aceptada por una parte significativa de estos.

Nuestra solución consiste en la realización de un servicio adicional para los hosts que sean servidores Web, que atiende exclusivamente a las arañas (y en general a cualquier tipo de robot) y básicamente resuelve los dos problemas fundamentales ocasionados por éstas: la sobrecarga de los servidores y de los enlaces. Al nuevo servicio lo llamamos *EROS* (Exclusive Robots Service)

EROS tiene como principales características:

- **Comunicación asincrónica entre robots y servidores**. Esto se logra al permitir que los robots actúen como clientes, en el momento que solicitan la información y como servidores con el objetivo de recibir las respuestas generadas por los servidores EROS.

- **Respuesta en diferido.** Los servidores EROS responden a los pedidos enviados por los robots en forma batch.

- **Sumarización de la información en los servidores**. La extracción de las partes relevantes de los documentos se realiza en los servidores. Esto permite un alto grado de paralelismo en dicha tarea y la posibilidad de transferir las sumarizaciones en lugar de los documentos completos, utilizando de esta manera menos ancho de banda.

- **Sumarización a pedido**. Son los robots los que deciden qué se debe incluir en la sumarización de acuerdo a sus necesidades.

1.1. Organización

Este trabajo consiste de 6 capítulos y 4 apéndices, los cuales se resumen a continuación:

En el capítulo 2, "*El problema de la búsqueda en la Web*", se introduce el tema de los índices y se describen las herramientas de búsqueda más utilizadas: los servidores de catálogos y los servidores de índices.

En el capítulo 3, "*Robots de descubrimiento de recursos (arañas)*", se introducen los robots y en particular las arañas. Se analizan los problemas ocasionados por los robots y se presentan las dos alternativas más importantes al uso de robots: Aliweb y Harvest.

En el capítulo 4, "*EROS: servicio exclusivo para robots*", se presenta el servicio EROS. Se describe su funcionamiento, tanto el protocolo como el formato de los mensajes. Se lo compara con HTTP y con las alternativas mencionadas.

En el capítulo 5, "*Un robot que utiliza el servicio EROS*", se describe un robot sencillo que utiliza el servicio EROS. Se muestra la interacción entre éste y el servicio.

En el capítulo 6 se presentan las conclusiones y posibles extensiones al trabajo realizado.

Por último se encuentran los apéndices:

En el apéndice A, "*Internet*", se realiza una breve reseña histórica del surgimiento de Internet, se ejemplifica su crecimiento con cuadros comparativos, se brinda información sobre TCP/IP y el modelo cliente/servidor.

El apéndice B, "*La World Wide Web*" detalla su crecimiento y describe los elementos de ésta (URL, protocolo HTTP, lenguaje HTML).

El apéndice C consta de una lista de robots y de una breve descripción de ellos.

En el apéndice D "*Manual del Usuario*", se presenta un pequeño manual con los pasos y requerimientos necesarios para utilizar el servicio y el robot.

2. EL PROBLEMA DE LA BUSQUEDA EN LA WEB

En los comienzos de la Web[1] [LEE/94a] el método para ubicar información relevante era la *navegación*. Se denomina *navegación* a la actividad guiada por el usuario de explorar los contenidos de la Web. Está relacionada con la organización, ya que, cuanto más organizada está la información, más fácil es navegar. Para navegar se cuenta con programas llamados *navegadores* (que también llamaremos browsers), que toman como entrada la dirección de un recurso de la Web (URL[2]), lo obtienen mediante el protocolo HTTP[3] [LEE/94b] y se lo muestran al usuario. A partir de ese documento obtenido, el usuario 'navega' seleccionando referencias a otros recursos (denominadas links) que le interesan, que son, también, recuperados por los *browsers*.

Por lo tanto, la técnica de navegación es útil cuando uno conoce el URL que contiene la información que le interesa, o al menos conoce un buen punto de partida desde el cual se pueda 'navegar' hasta encontrar la información deseada.

Para formalizar lo escrito en los párrafos anteriores notemos que la Web puede ser representada como un grafo orientado que abarca el conjunto de documentos HTML[4] ofrecidos por los servidores de ésta.

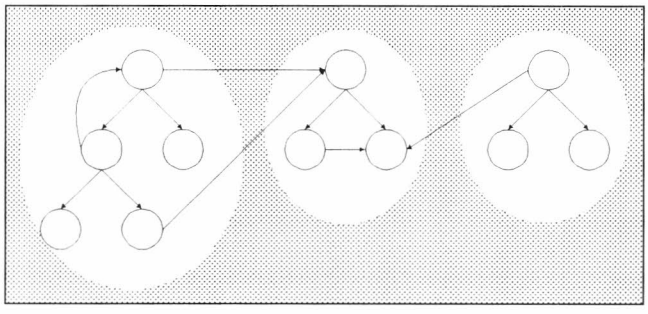


Figura 1: La Web vista como grafo

[1] Cuando mencionamos a 'la Web' nos estamos refiriendo a la estructura formada por los servidores que ofrecen el servicio HTTP. Y 'recursos de la Web' se refiere a los documentos contenidos en dichos servidores.

[2] Uniform Resource Locator, ver apéndice B.

[3] HyperText Transfer Protocol, ver apéndice B.

[4] HyperText Markup Language, ver apéndice B.

Cada nodo representa a un documento y se rotula con su URL. Cada eje corresponde a una referencia entre un documento origen y un documento destino. El grafo total que contiene a la Web puede ser dividido en subgrafos disjuntos que conforman grupos de documentos, cada uno correspondiente a un servidor Web diferente.

La Web es un sistema distribuido de información ya que sus recursos se encuentran en distintos servidores. Esto se puede representar en el grafo diferenciando los ejes que van de un documento a otro dentro del mismo servidor de los que unen documentos de distintos servidores.

Entonces, viendo a la Web como a un grafo, podemos acceder a un nodo de ésta siempre y cuando conozcamos su rótulo o al menos el de un nodo cercano desde el cual se pueda llegar en pocos saltos.

El problema es que, en general, es difícil conocer un buen punto de partida al querer buscar información dentro de la Web. Esto se debe a la gran cantidad de documentos que contiene y a la casi total libertad que existe en cuanto a la publicación de estos. En estos casos se hace imposible la localización de un recurso mediante la técnica de navegación.

Surge entonces la necesidad de realizar búsquedas dentro de la Web. La manera más natural de permitirlas es mediante la creación de índices que hagan referencias a los recursos y puedan ser consultados con el objetivo de encontrar la información deseada. La entrada correspondiente a un recurso dentro de un índice contiene información acerca de dicho recurso (por ejemplo, título, autor, etc).

2.1. Indices

Existen dos enfoques diferentes en la realización de índices de los recursos de la Web: *índices centralizados* e *índices distribuidos*.

En un índice centralizado, todo se encuentra en un único lugar, lo que hace que la búsqueda sea más fácil. Hoy en día, utilizar este tipo de índices es la única alternativa

para encontrar información en la Web. Es más, estos índices son ampliamente usados y responden a los requerimientos diarios de millones de usuarios. El principal inconveniente que presentan es que no ofrecen una solución escalable ante el crecimiento exponencial de la Web [LAL/96], [KOS/5]. Otro problema es el método que utilizan para recolectar la información a indexar. Obtienen, mediante programas, los documentos de la Web y luego extraen las 'partes relevantes' de estos con el propósito de generar el índice. De esta manera se hace un uso ineficiente del ancho de banda así como también se sobrecarga a los servidores.

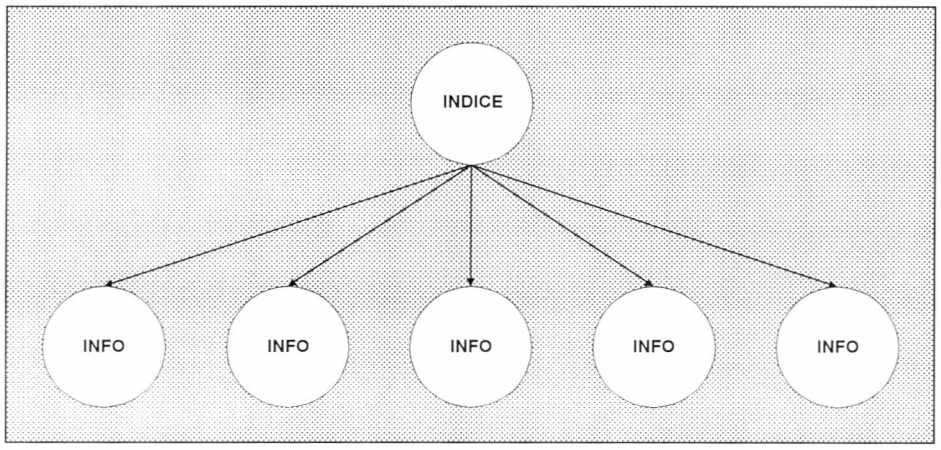


Figura 2: Indices centralizados

La otra alternativa es la generación de índices distribuidos. Aquí, los índices se distribuyen en diferentes servidores. Generalmente la relación entre los servidores es jerárquica. Estas estructuras tienen problemas cerca de la raíz, ya que deben tener toda la información contenida en los índices de nivel inferior, o en caso contrario, no poseen toda la información para guiar las consultas a servidores de nivel inferior [WEL/96].

Algunas ventajas de los índices distribuidos son:

- Reducen el tiempo de procesamiento durante las consultas, debido a que los servidores mantienen índices pequeños.

- Los índices se realizan en forma paralela.

- Se transmiten solamente sumarizaciones, y no documentos completos, con lo que se ahorra el ancho de banda.

Un ejemplo de índices índices distribuidos es Harvest [BOW/94] que será visto en el siguiente capítulo.

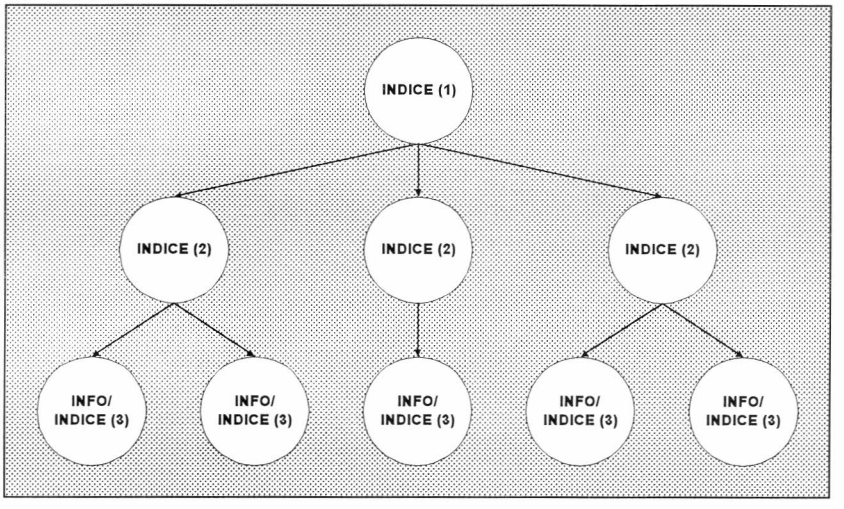


Figura 3: Indices distribuidos

A pesar de todas sus ventajas, los índices distribuidos no son usados en gran escala dentro de la Web. Una de las razones es que, en general, se requiere colaboración por parte de los servidores, es decir, los servidores deben tener uno o más procesos que indexen la información y permitan acceder a ella.

Un problema aun más grave es el hecho de que los índices centralizados son parte de un negocio. Los proveedores de estos índices permiten a los usuarios acceder a estos sin cargo alguno, pero obtienen ganancias a través de avisos publicitarios que son mostrados en sus documentos. Existe una gran cantidad de índices centralizados y cada uno de ellos posee un algoritmo propio para la indexación de los documentos. Ante esto, no se vislumbra, al menos en un futuro cercano, un modelo distribuido que permita, a los distintos proveedores de índices dejar de utilizar su esquema centralizado.

Un aspecto fundamental, tanto en los índices centralizados como en los distribuidos, es el contenido del índice. Qué es relevante y qué no depende del objetivo de

cada servidor de índices, que tiene que ver, entre otros factores, con lo que se quiera ofrecer a los usuarios.

Algunos de los datos que se suelen mantener en los índices son:

- Qué palabras se guardan de cada documento? : Todas, las más frecuentes, etc.
- Qué información se brinda como resumen? : Título, Primeras líneas, URL, etc.
- Cuántos documentos se indexaron?

2.2. Servidores de catálogos

Los servidores de catálogos ofrecen a los usuarios el acceso al contenido de la Web a través de un conjunto de categorías y subcategorías temáticas que pueden ser 'navegadas' por estos. Al llegar a cierto nivel de detalle, empiezan a aparecer en los documentos los links finales, es decir, aquellos que conducen directamente a las páginas candidatas a contener la información buscada. Tanto el trabajo de ordenamiento y jerarquización como el de la selección de los documentos a incluir en el catálogo es realizado por un equipo de expertos que evalúa las páginas disponibles. La inmensa tarea manual realizada hace que los resultados de una búsqueda dentro del catálogo sean de una calidad muy alta, pero por otra parte, se deja mucha información sin catalogar.

Volviendo a ver a la Web como un grafo, cada servidor de catálogos forma un subgrafo de ésta y puede ser visto de la siguiente forma.

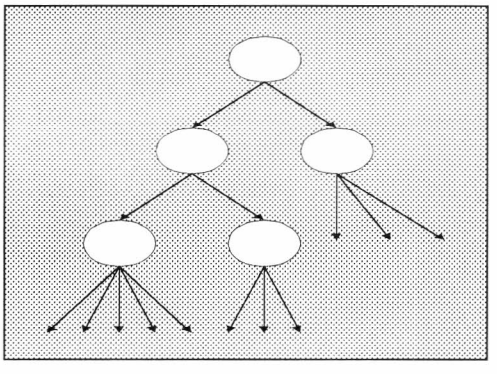


Figura 4: Servidor de catálogos

A partir de un nodo inicial (página de inicio del servidor de catálogos) hay referencias a nodos que representan una jerarquía de categorías. Un nodo que se encuentra en el último nivel de una jerarquía generalmente contiene links a la información real.

El usuario debe ubicar a la información que desea encontrar en una de las categorías provistas por el servidor de catálogos.

2.3. Servidores de índices

Los servidores de índices buscan palabras ingresadas por los usuarios en bases de datos y envían como respuesta un conjunto de URL's de documentos que contienen las palabras buscadas.

Cada servidor consiste básicamente de tres componentes. Un *robot* o *araña*, que es un programa que recolecta automáticamente información de millones de sitios de la Web, una *base de datos* donde la información es guardada y un *motor de búsqueda* que extrae de la base de datos la información requerida por los usuarios. Algunos servidores, como complemento, también ofrecen un pequeño catálogo, pero el principal objetivo es proveer una enorme base de datos que pueda ser consultada por los usuarios.

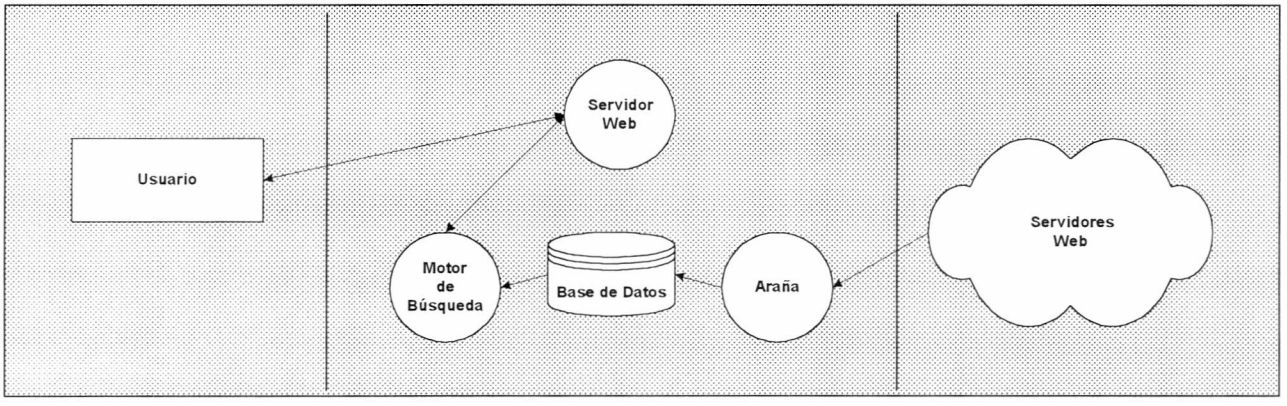


Figura 5: Componentes de un servidor de índices

Existe una gran cantidad de servidores de índices que difieren principalmente en el tamaño y contenido de la base de datos, siendo este último el que determina que tipo de búsqueda puede realizar el usuario.

2.4. Comparación entre servidores de catálogos e índices

La diferencia principal entre los servidores de catálogos y los servidores de índices está dada en que los primeros se basan en una base de datos generada manualmente mientras que los últimos utilizan programas que recorren automáticamente la Web para indexar la información contenida en ésta.

Como consecuencia de esto, las bases de datos utilizadas por los servidores de catálogos son mucho más pequeñas que las de los servidores de índices, ya que es imposible generar manualmente un índice de toda la Web.

Como contrapartida, cuando uno busca información dentro de un servidor de catálogos, éste nos brindará una respuesta mucho más precisa (aunque con menos alternativas), justamente por el hecho de que la meta-información fue obtenida por expertos y no por programas.

En general se utilizan los servidores de catálogos cuando no se tiene una consulta concreta. En cambio se usan los servidores de índices para hacer consultas más específicas.

2.5. Problemas de escalabilidad

Debido al crecimiento exponencial de la Web, tanto en cantidad de servidores como en cantidad de usuarios, se deben ofrecer soluciones que sean escalables, es decir, que se adecuen sin problemas a dicho crecimiento.

Dentro del problema de la búsqueda en la Web, se deben encontrar soluciones escalables para los siguientes temas:

2.5.1. Tiempo de obtención de recursos

Los robots recorren gran parte de los servidores de la Web obteniendo recursos de estos. Esta tarea puede llevar mucho tiempo, sobre todo si los robots hacen uso del protocolo HTTP para obtener los recursos, ya que al trabajar sobre TCP [1] [POS/81] obliga a establecer una conexión por cada uno de los recursos solicitados.

2.5.2. Generación de índices

En la mayoría de los sistemas actuales, la tarea de sumarizar los documentos es realizada luego de que estos son obtenidos mediante los robots.

Si consideramos la inmensa cantidad de documentos disponibles en la Web, se nota claramente que se tardará mucho tiempo en indexar el universo completo de documentos. Una de las consecuencias de este grave problema es que los índices pueden permanecer desactualizados por un gran período de tiempo.

Si las sumarizaciones de los documentos se hicieran en los servidores se obtendría un gran paralelismo en esta costosa tarea logrando una mayor rapidez en la generación y actualización de los índices.

[1] Transmission Control Protocol , Ver Apéndice A

2.5.3. Cuellos de botella

Si el robot corre en una única máquina, éste recibirá todos los documentos que pidió, provocando un cuello de botella en las cercanías de dicha máquina.

Otro cuello de botella se produce en la/s máquina/s que contiene/n los índices, ya que éstas pueden ser consultadas por muchos usuarios a la vez y deben poder realizar una consulta al índice en poco tiempo.

3. ROBOTS DE DESCUBRIMIENTO DE RECURSOS (ARAÑAS)

Como explicamos en el capítulo anterior, existen servidores que poseen índices de la Web, que pueden ser consultados por los usuarios con el fin de encontrar rápidamente la información deseada. Para generar dichos índices se utilizan programas que recorren la estructura hipertextual de la Web en forma automática, recuperando los documentos a través de los links encontrados, mediante el protocolo HTTP. Estos programas reciben el nombre de robots de descubrimiento de recursos o arañas[1] [KOS/1].

3.1. Robots

3.1.1. Definición

Dentro de la Web, se llaman robots a programas que la recorren en forma automática accediendo a sus recursos. Automática, se refiere a que un robot no necesita intervención humana a excepción de cierto conocimiento inicial. A medida que recorren la Web los robots van descubriendo nuevos recursos, de ahí el nombre de robots de descubrimiento de recursos.

3.1.2. Recorrido de la Web

Para entender cómo hacen los robots para recorrer la Web, representemos nuevamente a ésta como un grafo orientado en donde los nodos son los recursos y un eje

[1] Si bien se denomina arañas a un tipo especial de robots (los que se usan para el descubrimiento de recursos), utilizaremos indistintamente a partir de ahora los términos 'araña' y 'robot'.

representa un link de un recurso a otro. En las siguientes figuras se muestra parte de la Web y cómo se la puede representar como grafo.

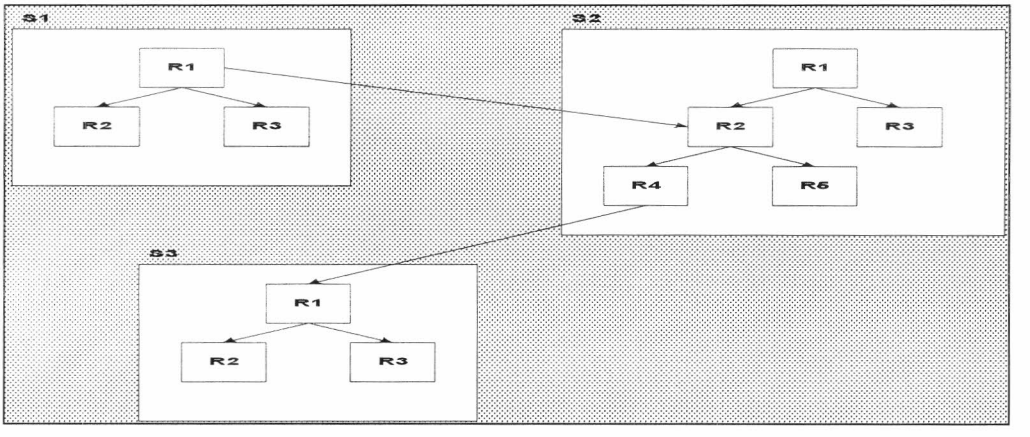


Figura 6: Web representada como grafo (1/3)

En el gráfico se visualizan 3 servidores; S1, S2 y S3. El servidor S1 cuenta con los recursos (documentos) R1, R2 y R3; el servidor S2 con los recursos R1, R2, R3 , R4 y R5 y el servidor S3 con R1, R2 y R3.

Representada como grafo, esa parte de la Web se ve de la siguiente manera:

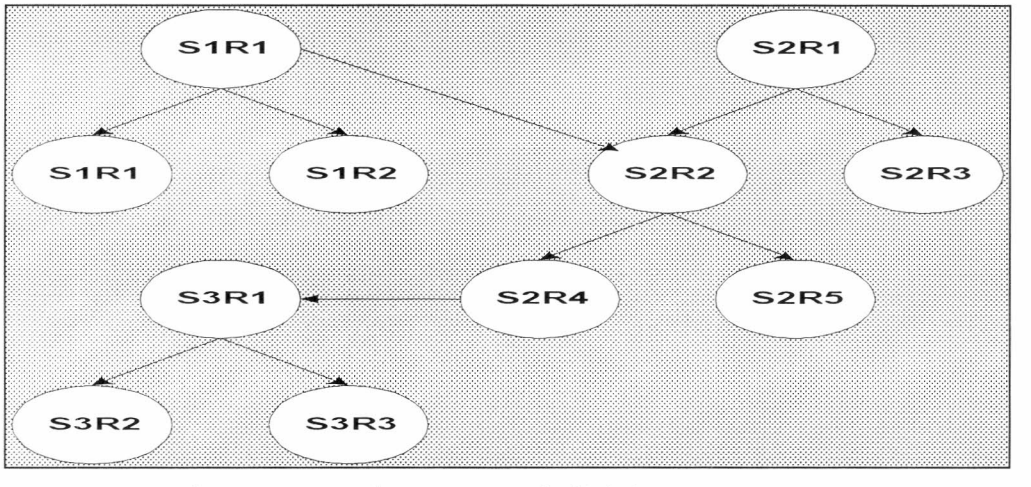


Figura 7: Web representada como grafo (2/3)

Cada nodo se rotula con su URL, que se corresponde a la concatenación del nombre del servidor con el nombre del recurso dentro de ese servidor.

Una representación más completa resulta de rotular a los ejes con una "I" (interno) si el nodo destino es un recurso dentro del mismo servidor o una "E" (externo) si se encuentra en otro servidor.

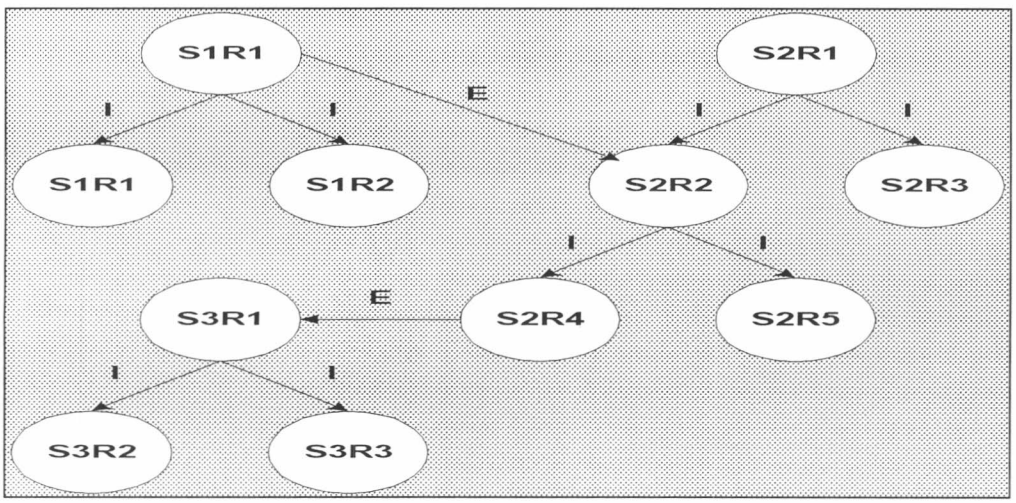


Figura 8: Web representada como grafo (3/3)

Con la noción de la Web representada como grafo podemos pensar el recorrido de la Web como un algoritmo de recorrido de grafos. Por lo tanto, un robot puede utilizar los algoritmos tradicionales como *profundidad primero* (*depth first*) o *ancho primero* (*breadth first*) para recorrer la Web [RUS/95]. Pero en general se usan algoritmos más sofisticados en la elección del próximo nodo a visitar. Por ejemplo, muchos robots utilizan el algoritmo ancho primero junto con la heurística de no visitar al mismo servidor dentro de un intervalo de tiempo.

En pocas palabras, los robots empiezan a recorrer la Web contando inicialmente con un conjunto de URL´s conocidos (nodos iniciales) y a partir de estos recuperan recursivamente los documentos a que hacen referencia los links (ejes) embebidos en estos.

Un tema importante es que los robots deben recorrer la Web una vez tras otra, ya que luego de recuperar un documento no tienen manera de saber si éste ha sido modificado, cambiado de lugar o eliminado. Por esto es muy difícil mantener un índice actualizado de la Web, ya que los robots pierden su tiempo recorriéndola toda y no solamente la parte actualizada de ésta. Lo que se suele hacer es llevar una estadística de cada URL en cuanto a la frecuencia de modificación y así establecer un mecanismo de prioridades en base a dicha frecuencia (se visitan primero los URL´s que cambian más frecuentemente).

3.1.3. Método de obtención de recursos

Luego de entender el mecanismo que utilizan los robots para recorrer la Web nos queda por ver cómo hacen los robots para obtener cada uno de los recursos (nodos). Aquí es donde aparece un punto clave para entender el comportamiento de los robots. No hay diferencia en como hace un robot para recuperar un documento y como lo hace un usuario a través de un browser. Los documentos son obtenidos utilizando la primitiva GET[1] del protocolo HTTP. Esto tiene como ventaja que no hay que agregar nada en los servidores para que un robot pueda funcionar, pero tiene el problema de usar un protocolo que no fue pensado para el tipo de tarea que hacen los robots. Veremos los inconvenientes más adelante.

3.2. Objetivos de los robots

Los principales objetivos de los robots son:

Análisis estadístico: Uno de los usos posibles de un robot es contar la cantidad de servidores de la Web [KOS/1]. También hay robots (por ejemplo RBSE [EIC/94]), que obtienen estadísticas del contenido de las páginas de la Web.

[1] Ver Apéndice B, "La World Wide Web" dentro de la sección "Protocolo HTTP"

Mantenimiento: Una de las principales dificultades en mantener una estructura hipertextual es que las referencias a otras páginas pueden pasar a ser 'links muertos', cuando la página referenciada es movida o eliminada [KOS/1]. No hay un mecanismo práctico en la Web de notificar a los que mantienen las páginas acerca de estos cambios. Los robots pueden ser usados para ayudar en el mantenimiento de la estructura hipertextual de la Web (por ejemplo MOMspider [FIE/94]).

Espejado: Esta es una técnica popular en el mantenimiento de sitios FTP[1][POS/85]. Un mirror (espejo) es un programa que copia una jerarquía de directorios recursivamente mediante el protocolo FTP, y luego recupera regularmente todos los archivos que cambiaron. En la Web, la tarea de espejado puede ser realizada por un robot.

Descubrimiento de recursos: Este es el uso principal de los robots: la obtención de los recursos de la Web con el objetivo de crear un índice. Estos robots reciben el nombre de arañas. Los servidores de índices (Alta Vista, Lycos, etc) los utilizan para la creación de sus índices. Las arañas permiten obtener en pocos minutos lo que a un ser humano le lleva varias horas.

Ahora bien, ¿qué hace la araña con los documentos obtenidos? Siendo el objetivo la indexación de la Web, las arañas sumarizan automáticamente los documentos, con mayor o menor detalle, con lo que generan una base de datos del contenido de la porción de la Web recorrida.

La primera araña utilizada, llamada World Wide Web Worm o WWWW [MCB/94], creaba un índice de títulos y URL´s, es decir, no indexaba el contenido de los documentos, solamente tomaba de estos los títulos y algunas cabeceras, así como también los links embebidos. Posteriormente surgen arañas, como Repository Based Software Engineering (RBSE) [EIC/94] o WebCrawler [PIN/94], que indexan todo el contenido de los documentos.

[1] File Transfer Protocol

3.3. Problemas causados por las arañas

Las arañas brindan un servicio valioso a los usuarios de la Web. Sin ellas sería imposible la creación de índices de la Web y la actualización permanente de estos. Pero también causan grandes problemas. Incrementan enormemente el tráfico en la red y en el caso de estar mal programadas pueden saturar a un servidor al intentar accederlo reiteradamente, empeorando la calidad del servicio que ofrece dicho servidor a sus usuarios.

Veamos más detalladamente los problemas ocasionados por las arañas:

Alto uso del ancho de banda: Las arañas requieren un ancho de banda considerable. Básicamente hay dos razones que contribuyen a este hecho. La primera es que las arañas operan continuamente en largos períodos de tiempo y muchas veces, con el objetivo de acelerar las operaciones, realizan recuperaciones paralelas, resultando en un mayor uso del ancho de banda. La segunda se refiere a que el mecanismo utilizado para la realización de un índice es ineficiente, ya que las arañas utilizan la primitiva GET del protocolo HTTP para obtener documentos enteros, quedándose luego de analizarlos con una parte muy pequeña de ellos. Por ejemplo, supongamos la existencia de una araña que extrae las 100 palabras más frecuentes de los documentos que obtiene. Tomando 15 como promedio de longitud de palabras (que es un valor muy alto), de un documento de 200000 bytes la araña solo utilizará 1500 bytes (100 x 15), transmitiendo 198500 bytes innecesariamente.

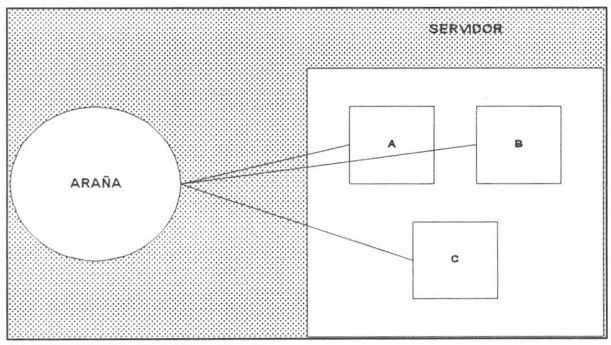


Figura 9: Conexiones paralelas

Sobrecarga de los servidores: Además de imponer altas demandas a la red, una araña también provoca grandes problemas en los servidores. Dependiendo de la frecuencia en que obtiene los documentos de un servidor, la sobrecarga puede ser considerable, lo que resulta en una peor calidad de servicio para los usuarios que acceden al servidor. El problema de que un robot acceda muchas veces a un mismo servidor en un corto intervalo de tiempo recibe el nombre de *rapid fire*.

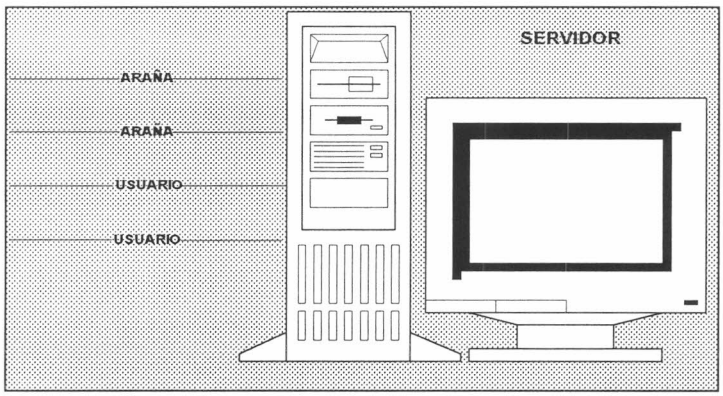


Figura 10: Arañas y usuarios compartiendo el servidor

La idea no es que los servidores dejen de atender a las arañas (como es muy probable que pase), sino que lo hagan mediante algún mecanismo de prioridades. El problema actual es que si el servidor decidiera darle menos prioridad a las arañas estas quedarían conectadas mucho tiempo con el servidor por las características sincrónicas del protocolo HTTP (es decir, el cliente no se desconecta hasta que el servidor no lo haga; sería mucho mejor que el cliente se pudiera desconectar). Más grave aún es el hecho de que una o varias arañas accedan simultáneamente al mismo servidor, multiplicándose los efectos mencionados.

Otro problema que afecta tanto a las arañas como a los servidores es que luego de que una araña obtiene un documento mediante la primitiva GET, el servidor se desconecta. Es decir, la araña debe abrir una nueva conexión TCP cada vez que se comunica con un servidor, independientemente de que se haya conectado en un tiempo

muy cercano para obtener otro documento. Es por esta razón que se dice que HTTP es un protocolo '*sin estado*'[1], ya que no recuerda lo pasado en conexiones anteriores.

Malas implementaciones: Los problemas mencionados se multiplican con arañas mal implementadas. Un error muy común es que la araña obtenga el mismo recurso más de una vez, por no mantener en una base de datos la historia de los URL´s ya visitados. Otro error es recuperar tipos de documentos que luego no son manejados, como por ejemplo, archivos que contienen gráficos.

Actualización de recursos: Hay un aspecto fundamental que trae como consecuencia una mala utilización de los recursos. Si bien como indicamos anteriormente, las arañas causan grandes problemas al intentar recuperar gran parte de la información de la Web, esto no sería tan importante si lo hicieran una única vez. Pero las arañas reinciden periódicamente ya que la información de cualquier documento pudo haber cambiado. Lamentablemente el único método que tienen las arañas para no recuperar todo un documento y darse cuenta luego de que éste no había cambiado, es utilizar el campo IF-MODIFIED-SINCE[2] para indicarle al servidor que no retorne el documento en caso de que no se haya modificado desde la fecha indicada, lo que implica realizar una conexión TCP con el servidor, que en muchos casos es más costoso que la transferencia misma. El problema es que para que el uso de IF-MODIFIED-SINCE tenga sentido, los documentos recuperados deben tener el header LAST MODIFIED [3] indicando fecha y hora de la última modificación. La mayoría de los documentos no lo contienen.

[1] Para más detalle acerca de los problemas de HTTP se pueden ver: [MOG/1], [MOG/2], [SPE/1]

[2] Ver Apéndice B, "La World Wide Web" dentro de la sección "Protocolo HTTP", subsección "El requerimiento"

[3] Ver Apéndice B, "La World Wide Web" dentro de la sección "Protocolo HTTP", subsección "La respuesta"

3.4. Estándar de exclusión de robots

Por los problemas mencionados en la sección anterior en cuanto a la carga de los servidores, algunas arañas empezaron a no ser bienvenidas por estos. La principal razón es que cuando una araña visita a un servidor puede pasar que pida una gran cantidad de documentos en un período muy corto de tiempo o quiera recuperar el mismo documento (o secuencia de documentos) reiteradamente.

Estos incidentes mostraron la necesidad de un mecanismo operacional para los servidores de la Web; así surgió el llamado "*Estándar para la exclusión de robots*" (EER), que es un conjunto de reglas informales que permiten a los servidores indicar qué áreas no deben ser exploradas por los robots [KOS/2].

El método usado para excluir robots es sencillo, se crea un archivo en el que el servidor especifica una política de acceso para robots. Este archivo, llamado archivo de exclusión de robots, es accesible mediante HTTP y se encuentra dentro del servidor, en /robots.txt.

Corresponde al robot chequear por la existencia del archivo en primer lugar, recuperarlo y adherirse a sus restricciones.

Uno de los usos comunes del EER es prevenir a los robots de la indexación de documentos temporales. Por ejemplo, muchos servidores generan documentos automáticamente para responder requerimientos de los usuarios y luego los borran.

Formato del archivo de exclusión de robots

El archivo consiste de uno o más registros con la forma campo:valor, siendo los principales campos:

User Agent	Identifica al robot en cuestión
Disallow	Especifica directorios que no pueden ser visitados

Ejemplos:

```
User Agent: *
Disallow: /temp/
Disallow: /pub/temp/
```

Indica que ningún robot debería visitar los documentos que se encuentran en los directorios /temp y /pub/temp/.

```
User Agent: *
Disallow: /temp/
User Agent: scooter
Disallow:
```

Indica que ningún robot excepto 'scooter' debería visitar el directorio /temp.

Robots éticos

Hay una serie de recomendaciones para el buen comportamiento de los robots [KOS/3], que se resumen en:

- Reconsiderar: ¿Se necesita un nuevo robot?
- Asegurarse que el robot pueda ser identificado y que su autor pueda ser fácilmente contactado.
- Testear el robot con datos locales antes de largarlo a la Web.
- Prevenir el rapid fire (por ejemplo utilizando breadth first).
- Eliminar la redundancia en la recuperación de los documentos.
- Seguir el estándar para la exclusión de robots.
- Recuperar sólo tipos de documentos que le sean útiles

Conclusiones

El EER es una solución parcial, que sólo intenta resolver el problema de la carga en los servidores restringiendo la cantidad de documentos que el robot puede acceder. El problema es que un robot no está obligado a respetar al EER, y aunque todos lo hicieran, no se resuelven los problemas anteriormente mencionados en cuanto al exceso de conexiones y el gran ancho de banda consumido.

3.5. Alternativas al uso de arañas

Ante los problemas que describimos en la sección anterior surgieron alternativas que se basan generalmente en la sumarización de los documentos dentro de los servidores.

3.5.1. Aliweb

Aliweb es un entorno para la recolección automática y procesamiento de índices dentro de la Web [KOS/94]. Su tarea es obtener archivos de índices de varios servidores de la Web y combinarlos en una única base de datos para facilitar la búsqueda.

La arquitectura de Aliweb se basa en la creación de índices distribuidos. Los generadores de la información tienen el control sobre la indexación de su propia información únicamente y existe un método para la combinación de estos índices.

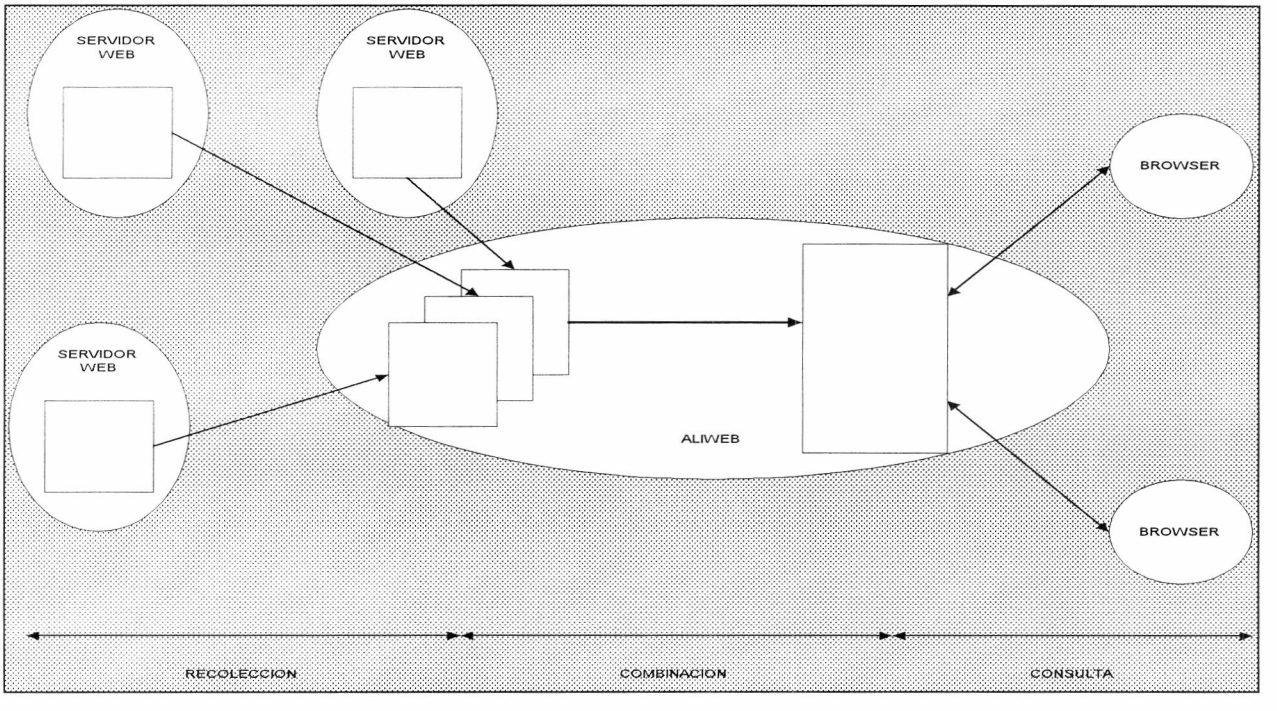


Figura 11: Arquitectura de Aliweb

Los administradores de los diferentes servidores crean los índices de los documentos de su servidor, en un formato común. Este formato se basa en un esquema del tipo atributo/valor, donde los atributos describen propiedades de un objeto.

Por ejemplo:

Tipo: Documento
Título: Robots en la Web
URL: /robots/robots.html
Descripción: Introducción a los robots dentro de la Web.
Palabras reservadas: robot, descubrimiento de recursos, índices.

Luego de que el administrador del servidor haya creado su archivo de índices, se le debe avisar a Aliweb para que lo tome en cuenta.

Para esta tarea Aliweb usa un form que contiene la siguiente información:

Nombre del servidor
Número de Port
Nombre de la persona responsable del servidor
Dirección electrónica de esa persona
Nombre del archivo índice, relativo a la raíz del servidor
Frecuencia con la cual actualizar
Protocolo

Con los datos del nombre del servidor junto con el número de port y el nombre del archivo índice, Aliweb posee la suficiente información para obtener el archivo índice.

Aliweb obtiene los archivos índices (a través de los protocolos de la Web) registrados en forma regular, recorre la lista de los usuarios y compara las fechas para

ver si necesitan ser actualizados y luego combina los índices correctos con los que forma una base de datos global.

Hay algunas críticas que se le hacen a esta alternativa:

La primera es el hecho de la indexación manual por parte de los proveedores y la idea de que estos no sean capaces de mantener sus propios archivos índices.

La segunda tiene que ver con la granularidad de los datos que son indexados. Se puede buscar lo que fue provisto por los proveedores de la información; si proveyeron sólo información general, no va a poder ser encontrada aunque exista en el servidor.

Para la búsqueda de información por un usuario una desventaja de usar Aliweb es que sólo los sitios registrados son buscados (no en toda la Web), pero a su vez la ventaja es que estos sitios que están registrados tienen una chance más alta de tener información (ya que en teoría ellos cambian sus índices cuando cambia algún dato o link).

El último problema es el cuello de botella de un solo sitio y el tamaño de una sola base de datos para toda la Web.

3.5.2. Harvest

Harvest [BOW/94] es un conjunto de herramientas, y, al igual que Aliweb, trata de resolver los problemas causados por los robots que se basan en HTTP para obtener la información a indexar. Los elementos fundamentales de Harvest son los *gatherers* y los *brokers*.

Los gatherers

El objetivo de los gatherers es recolectar información mediante métodos estándar como HTTP, FTP o archivos locales, para luego sumarizar dichos recursos de diferentes formas generando información estructurada para un índice.

Por ejemplo, un gatherer puede recuperar un archivo mediante FTP y extraer de él, el autor y el título; en otro caso, el gatherer puede obtener un documento de hipertexto (formato HTML) y extraer los links embebidos en éste.

A continuación se ven las dos formas de utilizar un gatherer. En la primera un gatherer es configurado para acceder en forma remota a varios hosts, proveedores de información, usando los protocolos FTP, Gopher o HTTP.

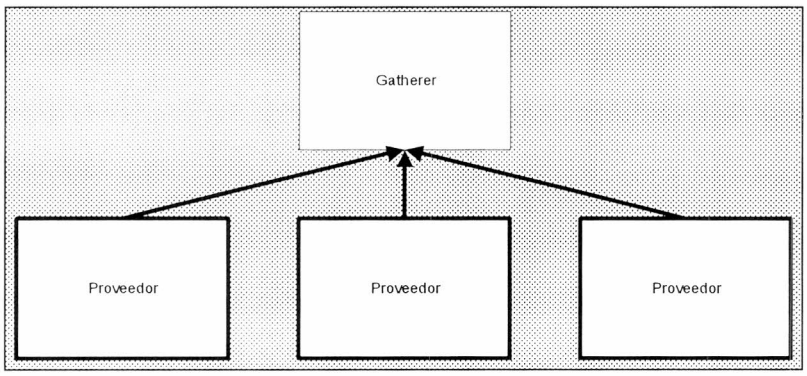


Figura 12: Un gatherer para varios proveedores

Esta configuración es útil para operar con sistemas que no corren el software de Harvest pero provoca una sobrecarga, tanto en los enlaces como en los servidores (por eso las líneas gruesas).

El otro caso consiste en que cada proveedor posea un gatherer local.

Figura 13: Gatherers locales

Correr un gatherer localmente es mucho más eficiente, ya que no se debe utilizar la red para obtener los recursos. Sin embargo los servidores tienen la sobrecarga provocada por el mismo gatherer.

El gatherer se basa en un archivo de configuración para saber qué datos recuperar. Veamos un ejemplo de un archivo de configuración:

```
Gatherer Name: Gather1
Top Directory: /usr/local/harvest/gatherers/sample

#Url's a recuperar
<ROOTNODES>
http://harvest.cs.colorado.edu/
</ROOTNODES>

<LEAFNODES>
http://www.cs.colorado.edu/home.html
http://www.cs.colorado.edu/~hardy/home.html
</LEAFNODES>
```

Como se ve en el ejemplo, un URL puede ser especificado como *leafnode* (nodo hoja) o como *rootnode* (nodo raíz). El gatherer sólo recupera y procesa los URL's que son leafnodes. En cambio, para los rootnodes, el gatherer expande al URL en cero o más leafnodes. Por ejemplo, en el caso de HTTP, un URL rootnode es expandido siguiendo los links embebidos en el documento HTML.

Harvest no está pensado para operar como una araña, ya que no recupera URL's que no estén especificados en rootnodes. Sin embargo, si se especifican muchos rootnodes, Harvest estaría actuando como una araña. Esto sería un despropósito ya que se cargaría excesivamente a los servidores y aumentaría notablemente el ancho de banda utilizado, problemas que Harvest quiere resolver.

Una vez que el gatherer recupera un documento, lo envía a un subsistema llamado *Essence*, que se encarga de sumarizar el documento. La información a sumarizar es totalmente configurable, es decir, cada servidor decide qué información extraer de sus documentos.

Los brokers

El objetivo principal de los brokers es recolectar la información de uno o más gatherers o de otros brokers, actualizar sus índices incrementalmente y ofrecer una interface de búsqueda a los usuarios.

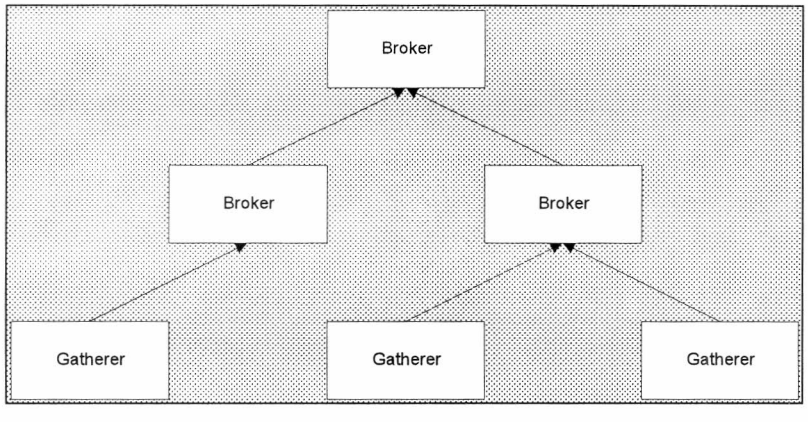


Figura 14: Brokers

Los brokers utilizan un archivo de configuración que les permite administrar su conectividad con los gatherers u otros brokers. El archivo contiene registros con el nombre del gatherer o del broker, el port en el que está escuchando, el tipo de colección y el filtro.

Veamos un ejemplo, el cuál obtiene información de dos gatherers (uno incremental y comprimido y el otro completo y no comprimido) y de dos brokers (uno incremental y el otro usando un filtro).

```
gatherer-host1.foo.com  8500  3  --
gatherer-host2.foo.com  8500  0  --
broker-host1.foo.com    8501  5  --
broker-host3.foo.com    8501  7  --Query Harvest
```

La descripción de los campos del ejemplo es la siguiente: el primero es el host del gatherer o broker, el segundo el número de port de ese host, el tercero el tipo de colección y el cuarto el filtro de la consulta.

El significado del código de colección se detalla a continuación:

Tipo	Proceso Remoto	Descripción	Comprimido
0	Gatherer	Colección completa	No
1	Gatherer	Colección incremental	No
2	Gatherer	Colección completa	Si
3	Gatherer	Colección incremental	Si
4	Broker	Colección completa	No
5	Broker	Colección incremental	No
6	Broker	Colección completa basada en un filtro	No
7	Broker	Colección incremental basada en un filtro	No

3.5.3. Robots cooperativos

Ninguna de las arañas más usadas comparten sus datos. El resultado de esto es que muchas de ellas pueden indexar los mismos recursos y documentos. Esto resulta en una gran carga innecesaria tanto para la red como para el servidor.

Surge así la idea de los *robots cooperativos (WebAnts)* [CHE/96], que consiste en crear *exploradores cooperativos* (llamados *hormigas*) que compartan el trabajo de encontrar recursos en la Web sin duplicar los esfuerzos. Cada hormiga indexa una parte de la Web y se comunican entre ellas mediante un protocolo interno.

El desarrollo de robots cooperativos fue motivado por las siguienters consideraciones:

- El descubrimiento de recursos se vuelve una tarea demasiado grande para un solo explorador.

- El hecho de tener un solo sitio para dicho servicio crea un cuello de botella.

- No es deseable que muchos exploradores examinen los mismos sitios. Si explorar la Web ya es un problema, tener un gran número de robots no cooperativos, y por ende, redundantes, es todavía peor.

Las hormigas, en contraposición con los robots, están diseñadas para compartir resultados con otras hormigas sin duplicar esfuerzos. Este modelo de robots cooperativos puede ser usado tanto para la búsqueda como la indexación de la Web.

Para el propósito de búsqueda, diferentes hormigas pueden basarse en resultados de otras. Cuando una encuentra un documento que satisface el criterio de búsqueda, puede compartir las referencias de ese documento con otras hormigas, de esta manera, estas hormigas sabrían que no tienen que examinar ese mismo documento, permitiendo así que la información sea obtenida más eficientemente.

En cuanto a la indexación, la cooperación entre hormigas permite que cada indexador conserve recursos distribuyendo la carga de la indexación entre diferentes exploradores. Cada índice proveería toda la información obtenida por una de las hormigas durante la exploración. Cuando se hacen consultas, un usuario podría restringir la consulta a una hormiga local o permitir que se propague a otras. Esto reduce el efecto de cuello de botella.

Para que esta alternativa tenga sentido, las arañas en existencia deberían acoplarse a esta solución, es decir, todas deberían utilizar la misma base de datos de información. Esto se ve muy lejano a la realidad, ya que cada servidor de índices trata justamente de diferenciarse del resto ofreciendo una mejor base de datos.

A nuestro juicio los robots cooperativos tienen sentido, no como solución de los problemas de sobrecarga de los servidores y ancho de banda, sino como método para ofrecer índices distribuidos en lugar de centralizados.

3.6. Conclusiones

Se ve claramente que hay dos alternativas actualmente en uso para la generación de un índice de la Web.

La primera se basa en utilizar programas llamados arañas, que se encargan de recuperar la información contenida en la Web para luego, localmente, generar un índice. Esta alternativa es la más usada ya que tiene como gran ventaja que puede llegar a la información de la Web con la estructura de información existente. Los problemas son que ocasionan una gran carga tanto en los enlaces como en los servidores.

La otra alternativa utilizada es generar los índices en los servidores, ya sea manualmente, como en Aliweb, o automáticamente, como se da en Harvest a través de los gatherers. Aquí pasa todo lo contrario; se solucionan los problemas ocasionados por las arañas a costa de la necesidad de una colaboración por parte de los servidores. En el caso de Aliweb la colaboración parte de la generación manual de los índices, mientras que en Harvest hay que instalar en cada servidor un gatherer, que es un programa que consume mucha memoria y tiempo de procesador.

Tanto Aliweb como Harvest son soluciones que no provocan los problemas causados por las arañas (alto uso del ancho de banda y sobrecarga de los servidores). Sin embargo, no ofrecen nada para poder solucionarlos, ya que están pensados para funcionar paralelamente a las arañas en existencia, en lugar de ofrecerles a éstas un entorno para que puedan funcionar sin causar tantos daños.

El principal inconveniente de Harvest es que no es universal, es decir no brinda a cualquier robot la posibilidad de obtener de los servidores lo que desea. Los Gatherers actúan independientemente de los pedidos de los clientes. Cada servidor indexa lo que le parece conveniente. Por lo tanto, si el objetivo de un robot fuera obtener de cada documento solamente las 10 palabras más frecuentes, sólo lo lograría de los servidores que configuraron al gatherer para que lo haga. Algunos servidores pueden haber elegido indexar las 100 palabras más frecuentes y otros pueden haber tomado la decisión de no indexar las palabras más frecuentes. En pocas palabras, en Harvest, no son los clientes

los que deciden que información indexar, sino los servidores, lo que hace impracticable utilizarlo como un servicio para los robots. En Aliweb pasa exactamente lo mismo.

4. EROS: SERVICIO EXCLUSIVO PARA ROBOTS

Luego de un análisis de los problemas que causan los robots obtuvimos las siguientes conclusiones:

- Se pueden realizar robots que actúen de tal manera que no consuman excesivamente los recursos críticos. Sin embargo esto no ayuda a la solución de los problemas, simplemente no causa más daño al ya existente.

- Se debe dejar de lado el protocolo HTTP como método de acceso a los recursos de la Web por parte de los robots. El protocolo HTTP, a pesar de sus deficiencias es útil para el objetivo de los browsers que es recuperar un documento y mostrárselo al usuario. Pero es muy costoso como método de acceso por parte de los robots. Al dejarlo de lado surgen alternativas claramente favorables como veremos más adelante.

- Los servidores deben diferenciar el acceso de un robot del de un browser para poder elegir distintos tratamientos ante cada uno de ellos.

- Los servidores de la Web deben atender a los robots en momentos en que se encuentren poco cargados para no perjudicar a sus usuarios.

Surge de estos ítems el objetivo de nuestra tesis; **crear un servicio que atienda exclusivamente a los robots y que solucione los problemas causados por estos. El nombre que le pusimos al nuevo servicio es EROS (Exclusive RObots Service).**

Decidimos que el servicio EROS debe cumplir las siguientes condiciones, igualmente importantes:

- Debe ser universal

Esto quiere decir que el servicio debe ser útil para todos los robots, tanto para robots nuevos, diseñados para utilizar el servicio, como para los robots en uso actualmente (por ejemplo, los utilizados por Altavista o Lycos), que deberán cambiar parte de su código para poder aprovecharlo. Si el servicio sólo fuera usado por unos pocos robots no se solucionarían los problemas, ya que el resto de los robots los seguirían ocasionando.

El servicio EROS está pensado para lograr este cometido, ya que le permite a los robots obtener la información que estos desean. Es decir, el servicio es flexible, permitiendo que el robot le indique al servidor lo que tiene que retornar. Por ejemplo, un robot que sirva para el mantenimiento de links, le puede solicitar a los servidores que retornen solamente los links embebidos en los documentos. Un robot como Lycos, que registra solamente las palabras claves de los documentos, le puede pedir a los servidores que retornen las N palabras más frecuentes, sin la necesidad de obligar a transmitir todo el documento. En el peor de los casos, para los robots que necesitan obtener los documentos completos ya que, por ejemplo, indexan todas las palabras, se le puede solicitar a los servidores que transmitan los documentos completos, lo que emula a la primitiva GET de HTTP, aunque internamente actúa en forma diferente, como veremos en los puntos posteriores.

Es decir, el servicio es útil para cualquier robot, independientemente del objetivo de estos.

- Debe producir mejoras en los servidores de la Web

Para lograr que el servicio sea expandido en toda la Web (o aunque sea, en gran parte de ella), debe provocar alguna mejora en los servidores.

Las mejoras producidas por el nuevo servicio son básicamente las siguientes:

- El mero hecho de ser un nuevo servicio, que corre en forma paralela a HTTP, le permite a los servidores tratar en forma diferente a los pedidos de un robot que a los de un browser.

- Actúa en forma batch, lo que les permite a los servidores responder a los pedidos de los robots en los momentos en que se encuentren desocupados o dándole menos prioridad a los pedidos de un robot que a los pedidos de un browser.

- Los pedidos de los robots son atendidos de a uno por vez. Debido a esto no existe el problema ocasionado por uno o más robots al pedir numerosos recursos de un servidor en un pequeño período de tiempo.

- Por último, es importante notar que los procesos que conforman el nuevo servicio no consumen muchos recursos de los servidores. Este punto es crucial, ya que de lo contrario los servidores se encontrarían sobrecargados todo el tiempo y no solamente ante el acceso de los robots.

De esta manera se logra la solución al problema de la sobrecarga de los servidores ocasionada por el acceso desmedido de los robots.

- Debe producir mejoras en los robots

Que el servicio permita mejorar el rendimiento de los servidores hace que sea factible disponer de éste en la mayoría de los servidores de la Web. Sin embargo, con esto no basta para que los robots decidan usarlo en lugar de usar HTTP para obtener los recursos.

Hay dos puntos cruciales en los que los robots se verían beneficiados por el servicio:

El primero es que el servicio es asincrónico, el robot no se tiene que quedar esperando por la respuesta del servidor. Para lograr el asincronismo, el robot actúa no sólo como cliente, enviando requerimientos, sino también como servidor esperando las respuestas de los servidores que ofrecen el servicio EROS.

El segundo es que las sumarizaciones se realizan en los servidores en lugar de tenerlas que hacer el robot luego de recuperar la información.

Esto les permite a los robots realizar el recorrido de la Web en forma mucho más rápida, ya que, aunque los servidores vistos uno por uno, pueden tardar más en retornar la información requerida, tendremos a una gran cantidad de servidores trabajando en forma paralela. El tema del tiempo que tarda un robot en recorrer la Web es fundamental, ya que si demora mucho, la información obtenida quedará rápidamente desactualizada debido al gran dinamismo de la Web (cambios, movimientos, eliminación y creación de documentos).

En resumen, estas son las características principales del servicio EROS.

- Le retorna al robot sólo lo que éste le pide.

- Transfiere sumarizaciones, no documentos completos (reduciendo así el ancho de banda).

- La comunicación entre los robots y el servicio es asincrónica.
- Trabaja en forma batch.
- Consume pocos recursos de los servidores.

4.1. Descripción del servicio

Cada servidor de la Web que desee un acceso diferenciado para los robots de descubrimiento de recursos debe ofrecer el servicio EROS en un port bien conocido. El servicio debe estar asociado a UDP[1] [POS/80]. La razón de usar UDP en lugar de TCP la explicaremos más adelante.

Por otra parte, todos los robots que hagan uso del servicio deben tener un proceso 'escuchando' en un port local bien conocido, que será donde el servidor envíe la respuesta. Este proceso también debe estar asociado a UDP.

La interacción entre el robot y el servidor EROS se basa en el siguiente protocolo.

4.2. Protocolo

- Los servidores EROS escuchan en un port bien conocido, asociado a UDP.
- Cuando comienza a funcionar un robot, se inicia un proceso que escucha en un port bien conocido, lugar en donde llegarán las respuestas del servidor. Este port está asociado con UDP.

[1] User Datagram Protocol

- El robot envía a un servidor EROS un mensaje con el formato de mensaje de solicitud.
- El servidor simplemente guarda el mensaje en una cola mediante el proceso servidor.
- Para cada uno de los mensajes encolados, el servidor realiza las siguientes tareas en un momento de poca actividad.
  - Extrae del documento indicado en el mensaje la información solicitada mediante un proceso que llamaremos *sumarizador* o SUM.
  - Envía a la dirección indicada en el mensaje, dicha información.

4.3. Mensaje de solicitud

A continuación se muestra el formato y descripción del mensaje de solicitud, que es el mensaje que los robots le envían a los servidores EROS.

El formato es muy simple y consta de una serie de campos separados por blancos:

Dirección_Retorno Nombre_Recurso Tipo Palabras_Clave Palabras_iniciales Headers Fecha

La descripción de los campos es la siguiente:

Dirección de retorno	Es el nombre o dirección IP de la máquina en donde el servidor debe retornar los resultados. En general es la misma máquina que inició al robot. El servidor enviará los resultados a un port bien conocido dentro de esta dirección. **Ejemplo: robot.com**

Nombre del recurso	Esta es la ubicación dentro del servidor del documento que se quiere sumarizar. **Ejemplo: /textos/resumen.html.**
Tipo de respuesta	Este campo le indica al servidor qué información debe incluir en la sumarización (**L**: Solamente links, **K**: Solamente palabras claves, **A**: Ambos (tanto links como palabras claves), **T**: Texto lineal): Este campo es muy importante en términos de eficiencia. Por ejemplo, hay muchos robots a los que solamente les interesa obtener los links de los documentos, con el objetivo de chequear la validez de estos. Estos robots enviarán un mensaje con Tipo de Respuesta 'L', ahorrando de esta manera un gran ancho de banda. El último valor ('T') sirve en el caso de que un robot necesite todo el documento y no solamente las palabras claves. En lo posible conviene que los robots solamente utilicen esta opción cuando no les quede otro remedio ya que consume más ancho de banda. De todas formas, si bien el resultado obtenido es muy parecido a usar el método Get del protocolo HTTP, recordemos la gran diferencia: el servidor no transmite el documento en el momento en que es solicitado, sino que lo hace en un momento de poca actividad y además lo hace de a un pedido por vez. **Ejemplo: "A"**
Palabras claves	Indica la cantidad de palabras claves (palabras más frecuentes del documento) que el servidor debe incluir en la sumarización **Ejemplo: 30**
Palabras iniciales	Este campo indica la cantidad de palabras a partir del comienzo del documento que se deben incluir en la sumarización. Estas palabras pueden ser utilizadas luego por una interface de búsqueda a modo de comentario sobre la página. **Ejemplo: 50**
Headers	Indica los códigos de los headers que se deben incluir en la

	sumarización. Este campo sirve para acotar la información enviada por el servidor. Hay muchos robots que solamente extraen algunos headers de los documentos. **Ejemplo: 001011 (códigos 00, 10 y 11)**
Fecha del documento	Este campo es enviado en caso de que ya se tenga localmente el documento y se lo quiera obtener nuevamente sólo si ha cambiado. El servidor comparará la fecha del documento que envía el robot con la propia y solamente retornará los resultados en el caso en que difieran. Este campo le permite tanto a los robots como a los servidores ahorrarse tiempo y recursos. También se utiliza menos ancho de banda ya que sólo se transmiten los documentos que cambiaron.

Como se ve, el servicio es muy general y puede ser utilizado por cualquier robot, ya que, como peor caso puede solicitar 'T' como tipo de respuesta para obtener el documento, que es muy similar, en cuanto a contenido, a lo obtenido por el método GET del protocolo HTTP.

Ejemplos

A continuación se muestran dos ejemplos de mensajes de solicitud:

Mensaje: **Robot.com index.html A 20 30 001011**
Destino: Robot.com
Recurso: index.html
Tipo: Ambos (texto + links)
Cantidad de palabras clave: 20
Cantidad de palabras iniciales: 30
Headers incluidos: 00 (<TITLE>), 10 (<META>), 11 (Descripción de los links)

El mensaje le indica a un servidor que retorne tanto los links como la sumarización del documento index.html. La sumarización pedida consta de 20 palabras clave, las 30 palabras iniciales y el contenido de los headers indicados.

Mensaje: **Maq1.gov /datos/provincias.html T**
Destino: Maq1.gov
Recurso: /datos/provincias.html
Tipo: T (Documento completo)

En este caso se le indica al servidor que retorne todo el documento /datos/provincias a la máquina Maq1.gov.

4.4. Mensaje de respuesta

El siguiente es el formato de un mensaje de respuesta:

```
URL
TIPO
FECHA

Contenido dependiente del tipo de respuesta
```

A continuación se describen los campos:

URL	Es el URL del recurso cuya sumarización envía el servidor. Se encuentra en la primera línea del mensaje de respuesta **Ejemplo: http://www.disney.com/mickey.html**
TIPO	Indica el tipo de respuesta: **TEXTO** si es una sumarización, **LINKS** si contiene los links embebidos en el documento, **COPIA** si es el documento íntegro, **INALC** si es un pedido erróneo.
Fecha del documento	Indica la fecha del recurso.

Ejemplos

```
Serv1.com/index.html
LINKS
www.serv2.com/index.html
www.serv2.com/databases.html

Es un mensaje de respuesta que contiene los links
embebidos en el documento /index.html del servidor
Serv1.com
```

Y aquí se muestra un ejemplo de respuesta de tipo TEXTO:

www.dc.uba.ar/index.html
TEXTO
Palabras clave
Alumnos Computación Docentes Facultad Materias Universidad
Palabras Iniciales
Esta es la Home Page de la Facultad de ...
Título
Facultad de Ciencias Exactas de la Universidad de Buenos Aires.

4.5. Implementación del servicio

El servicio consta de dos procesos que corren en forma paralela y se comunican entre sí a través de una cola de mensajes y de señales. El primero se llama *servidor* y el segundo, *sumarizador* (SUM). Ambos están pensados para consumir muy pocos recursos.

4.5.1. El servidor

Como se indicó anteriormente, el proceso servidor se encarga de escuchar en un port bien conocido que es exclusivo del servicio para robots. Cuando llega un mensaje a ese port, simplemente lo ubica en una cola. Esta tarea es muy rápida, razón por la cual el servidor es iterativo [COM/93], es decir, atiende de a un mensaje a la vez. Si llegan dos o más pedidos simultáneamente y el servidor no puede atenderlos (muy poco probable, justamente, por la escasa tarea que debe realizar por cada pedido), no hay problema ya que quedan guardados en una cola de entrada del servidor.

Básicamente, el servidor hace lo siguiente:

```
Escuchar en port bien conocido esperando mensajes de los robots
Iniciar el proceso sumarizador con la mínima prioridad
Repetir
   Suspender proceso hasta la llegada de un nuevo mensaje
   Grabar nuevo mensaje en cola MENSAJES # Ver formato en sección anterior
   Avisar al proceso sumarizador la existencia de mensajes
Por siempre
```

4.5.2. El sumarizador

Este proceso se encuentra muy cercano al proceso servidor. Su tarea es ir tomando los mensajes de la cola de salida del proceso servidor y extraer la información útil de los documentos solamente en el caso en que la máquina servidora se encuentre poco ocupada.

El mensaje indica dónde se encuentra el documento dentro del servidor, cuál es la dirección a donde se deben enviar los resultados, y el contenido deseado del mensaje de respuesta.

El sumarizador crea un mensaje que contiene la información solicitada por el robot. Luego lo envía a la dirección indicada.

A continuación se muestra el pseudocódigo del proceso sumarizador.

```
Quedar suspendido hasta recibir señal del proceso servidor

Repetir

   Mientras existan mensajes en cola MENSAJES

           Obtener nuevo mensaje

           Extraer del mensaje: destino, recurso, pedidos_robot (tipo,
           palabras_clave, palabras_iniciales, headers)

           Abrir recurso

           Generar archivo RESPUESTA tomando de recurso lo solicitado en
           pedidos_robot

           Enviar a destino el archivo RESPUESTA # Ver formato en sección
           anterior

   Fin Mientras

   Quedar suspendido hasta recibir señal del proceso servidor

Por Siempre
```

4.5.3. Diseño del servidor

La comunicación entre el robot y el servidor EROS primero y entre el servidor EROS y el robot después (en la respuesta), se realiza mediante el protocolo UDP, sin establecer conexión alguna. Esta decisión se basa en:

- La longitud de los mensajes, tanto en el requerimiento como en la respuesta, es pequeña, permitiendo el envío de un solo mensaje UDP, evitando la posibilidad de que lleguen mensajes en forma desordenada, es decir, el mensaje llega o no llega, pero si llega lo hace en forma ordenada (pues es un único mensaje!).

- Es sencillo detectar los errores en el nivel de aplicación, por lo que no es necesario utilizar TCP. Si el mensaje fue enviado y no se recibió una respuesta en un tiempo respetable, se deduce que el mensaje nunca llegó al servidor o que llegó pero se perdió la respuesta del servidor al robot. En ambos casos el robot debe efectuar nuevamente el requerimiento.

- Tanto el robot como el servidor se sobrecargan mucho menos al usar UDP que TCP. No necesitan establecer una conexión y luego de enviar un mensaje siguen con su tarea.

4.6. Comparaciones

En esta sección veremos como se resuelven diferentes problemas en tres modelos distintos: robots que usan HTTP para la obtención de recursos, Harvest y robots que usan el servicio EROS.

4.6.1. Obtención de recursos

Robots que usan HTTP

Para obtener un recurso, los robots establecen una conexión con el servidor que lo posee y se lo solicitan. El recurso es transferido en su totalidad. La sumarización de la información se realiza luego de la recuperación. Los pasos que se siguen para recuperar un recurso son los siguientes:

1) El robot establece una conexión TCP con el servidor.
2) Le envía al servidor un mensaje con el método GET aplicado al documento indicado en el URL.
3) Una vez obtenido el mensaje el servidor transfiere el documento completo al robot.
4) El servidor se desconecta.
5) El robot también se desconecta.

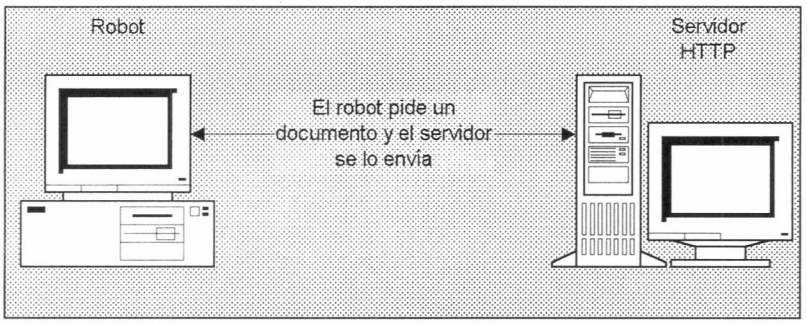


Figura 15: Obtención de recursos por robots que usan HTTP

Harvest

Aquí, el cliente (broker) se conecta con otro broker o con un gatherer y transfiere la información deseada. Al contrario que con HTTP, no se transfieren los recursos completos, sino sumarizaciones de estos. Además, se transfieren todas las sumarizaciones en una sola conexión. Entonces, los pasos son los siguientes:

1) Un broker se conecta con otro broker o con un gatherer, indicándole mediante un filtro qué sumarizaciones desea recuperar.

2) El servidor le envía las sumarizaciones que ya han sido generadas con anterioridad.

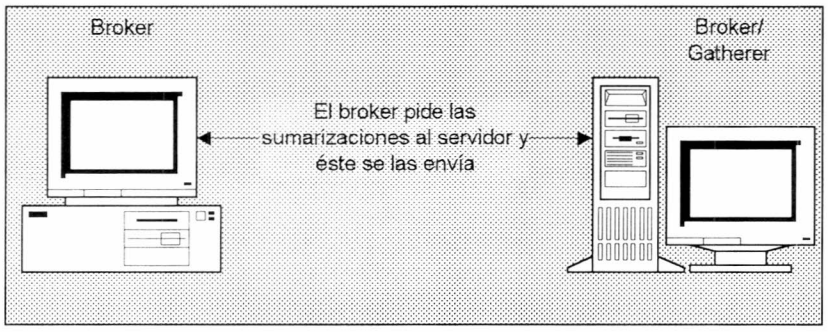


Figura 16: Obtención de recursos en Harvest

Robots que usan EROS

El robot le envía al servidor un mensaje indicándole el recurso a sumarizar, qué debe incluir en la sumarización de éste y dónde debe retornar el resultado. El servidor, en algún momento futuro, sumariza el documento y lo envía a la dirección indicada en el mensaje.

1) El robot envía un mensaje al servidor mediante UDP, indicándole lo que desea obtener y la dirección en la que debe enviar la respuesta. Siempre se encuentra escuchando para recibir las sumarizaciones.

2) El servidor envía un mensaje a la dirección indicada por el robot respondiendo al pedido solicitado.

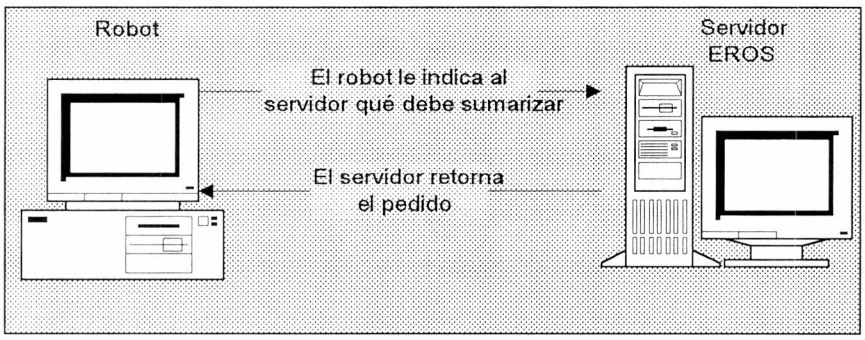


Figura 17: Obtención de recursos en EROS

4.6.2. Generación de las sumarizaciones

Robots que usan HTTP

Los robots que obtienen los documentos mediante HTTP, reciben los documentos completos de los servidores y recién en ese momento los sumarizan.

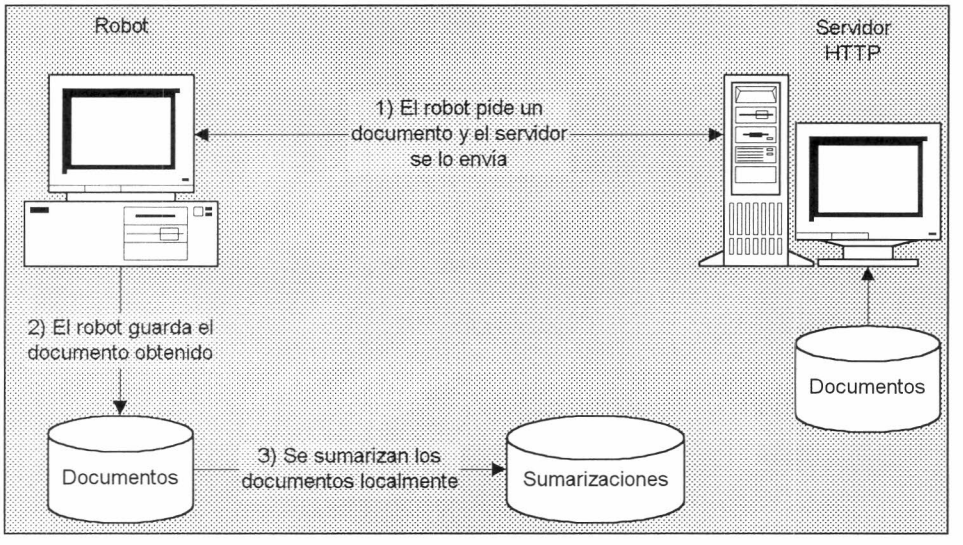


Figura 18: Sumarizaciones en robots que usan HTTP

Harvest

Los documentos son sumarizados por gatherers directamente en el servidor. Esto se hace autónomamente del cliente.

Un broker le pide las sumarizaciones, previamente generadas, al servidor.

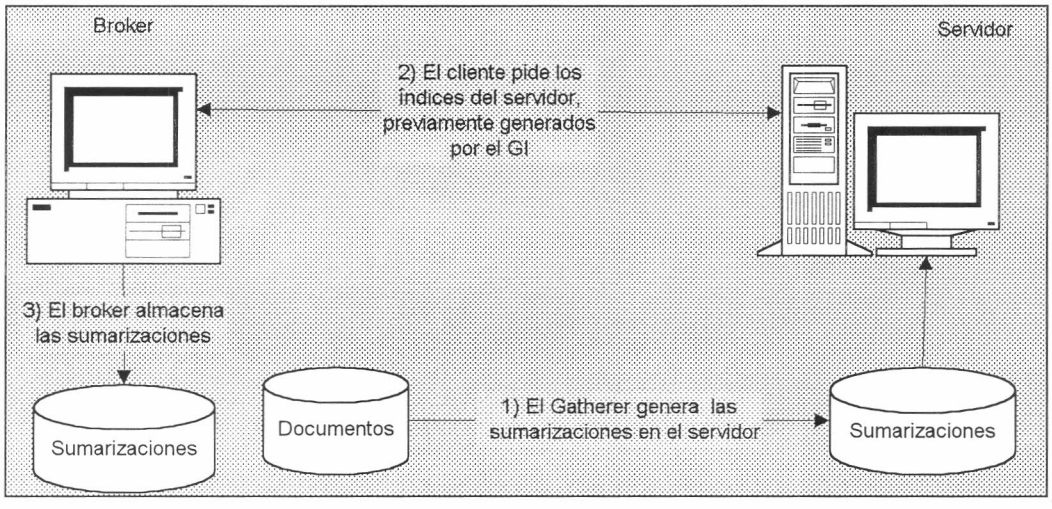


Figura 19: Sumarizaciones en Harvest

Robots que usan EROS

Los documentos son sumarizados en el servidor, luego del pedido de un robot.

El servidor envía los resultados al robot.

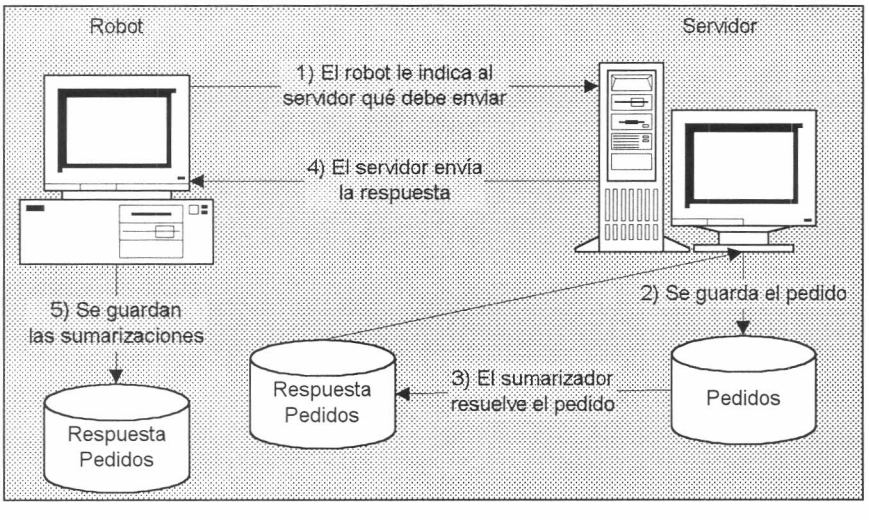


Figura 20: Sumarizaciones en EROS

4.6.3. Diferencias

La siguiente tabla muestra las diferencias entre los robots que usan HTTP y el servicio EROS.

Descripción	***HTTP***	***Nuevo servicio***
1) Clientes	Browsers	Robots
2) Acceso	Sincrónico	Asincrónico
3) Modelo	Interactivo	Batch
4) Transporte	TCP	UDP
5) Respuesta	El servidor debe responder inmediatamente	El servidor decide cuando responder
6) Sumarizaciones	Se obtiene la información. Luego se sumariza.	Se sumariza en los servidores.
7) Transferencia	Transfiere documentos completos	Transfiere la sumarización de los documentos

1) HTTP está pensado para interactuar con usuarios a través de los clientes browsers, aunque también lo utilicen los robots para obtener recursos de los servidores. En cambio, el servicio EROS está pensado solamente para clientes robots.

2) Como ya mencionamos, HTTP es sincrónico. El cliente robot queda conectado con el servidor. Esto implica que los servidores deben responder inmediatamente a los requerimientos de los clientes. Por el contrario, el servicio EROS es asincrónico, es decir , los robots no quedan conectados con los servidores, lo que le permite a estos responder cuando estén poco cargados.

3) HTTP fue pensado para modelos interactivos. En general se lo utiliza mediante un browser, que luego muestra la información solicitada en la pantalla, por lo que es necesaria la transferencia inmediata de los recursos. El servicio EROS está pensado

para un modelo batch, ya que ante el objetivo de los robots de descubrimiento de recursos (generar un índice), no se requiere la transferencia de los recursos en forma inmediata.

4) HTTP usa TCP, pues este es un protocolo confiable, necesario para que al usuario final le lleguen los datos en forma correcta. El servicio EROS utiliza UDP, ya que no es necesaria la confiabilidad a nivel de transporte (los clientes robots se encargan de los errores) y permite una mayor eficiencia tanto en los robots como en los servidores.

5) Al utilizar HTTP, el servidor debe responder en forma inmediata (independientemente de la carga del servidor en ese momento). Por el contrario, en el servicio EROS, el servidor simplemente recibe un mensaje, al que responderá cuando se encuentre poco ocupado.

6) En los robots que utilizan HTTP, las sumarizaciones se realizan luego de obtener la información. En cambio, en el servicio EROS, las sumarizaciones se realizan en cada servidor, lo que permite paralelismo en la sumarización de los documentos.

7) El protocolo HTTP obliga a recuperar los documentos completos a diferencia del servicio EROS que permite al servidor decidir qué parte del documento retornar.

Ahora, veamos las diferencias entre el servicio EROS y Harvest:

Descripción	***Harvest***	***Nuevo servicio***
1) Información a indexar	Lo decide el servidor (gatherer), la misma ante cualquier pedido.	Lo decide el cliente (robot). Dependiente del pedido.
2) Recursos en el servidor	Procesos que consumen muchos recursos.	Poca carga en el servidor
3) Transporte	TCP	UDP
4) Respuesta	El servidor debe responder inmediatamente	El servidor decide cuando responder

1) En Harvest, la sumarización de los documentos dentro de un determinado servidor es única mientras que en el servicio EROS pueden existir distintas sumarizaciones para un mismo recurso ya que es el robot el que indica qué sumarizar.

2) Un servidor Harvest consta de muchos procesos que consumen gran cantidad de memoria así como también utilizan gran parte del tiempo del procesador. Para ser más precisos, el manual de Harvest [HAR/96] especifica que para correrlo normalmente hacen falta por lo menos 64 Mb de ram y 1 Gb de espacio en disco. El hecho de que se consuma mucho tiempo de procesador tiene que ver principalmente con el gatherer, que debe estar constantemente monitoreando si hubo cambios en los documentos del servidor para actualizar los índices de modo que estos estén disponibles. En el nuevo servicio casi no se consumen recursos.

Las diferencias 3) y 4) tienen la misma explicación que en la comparación entre el servicio EROS y los robots que utilizan HTTP.

5. UN ROBOT QUE UTILIZA EL SERVICIO EROS

En este capítulo se describe un robot que utiliza al servicio EROS para obtener los recursos de la Web. El robot es muy sencillo ya que nuestro objetivo al desarrollarlo fue ilustrar como se utiliza el servicio.

El robot toma como entrada un conjunto de URL´s conocidos. A partir de estos irá descubriendo nuevos recursos (otros URL´s) dentro del espacio de servidores EROS que pretende ser el mismo que el de servidores Web. Además, el robot se inicia con una serie de parámetros que le indican qué información debe solicitar a los servidores que visita[1]. Podemos decir entonces que este robot es genérico ya que puede ser utilizado para solicitar diferentes tipos de sumarizaciones.

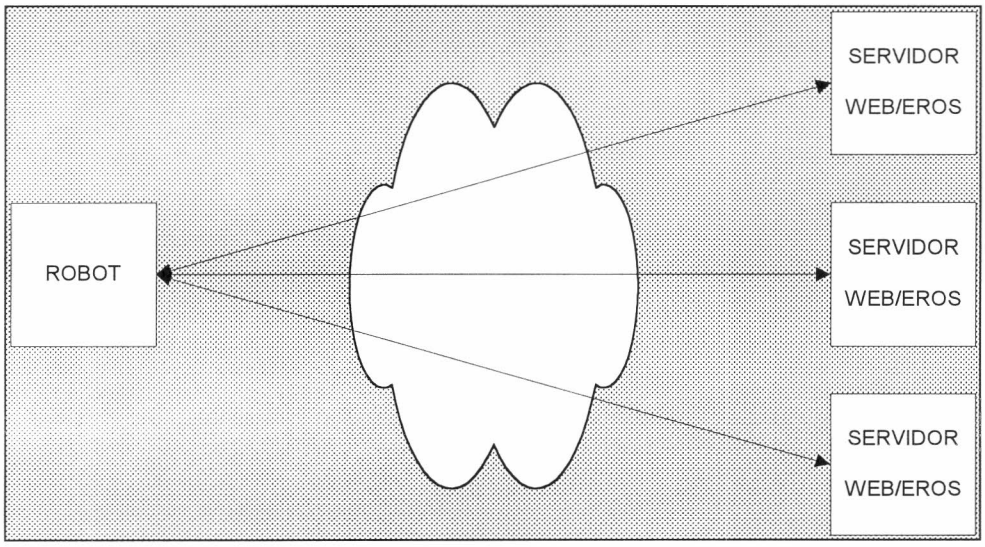


Figura 21: Robot que utiliza el servicio EROS

[1] Ver Apéndice D, "Manual del Usuario"

El robot mantiene la historia de los URL´s, que en un determinado momento se encuentran en un único estado. Los posibles estados son:

- **Pendiente:** El URL es conocido por el robot pero todavía no ha sido seleccionado para una futura visita. Los URL´s iniciales se encuentran en este estado cuando comienza a funcionar el robot.

- **A solicitar:** El URL fue seleccionado para ser visitado pero aun no lo ha sido.

- **Solicitado:** El robot solicitó la información del URL.

- **Recibido:** El robot recibió la información del URL.

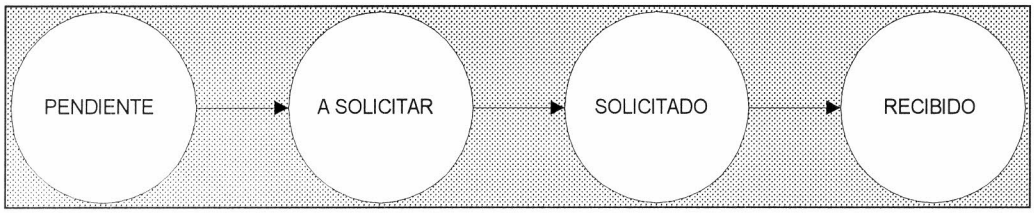


Figura 22: Estados de los URL´s

Veamos un diagrama que muestra el funcionamiento del robot. Se indican los procesos y almacenamientos, tanto del robot como de un servidor EROS y la interacción entre los procesos.

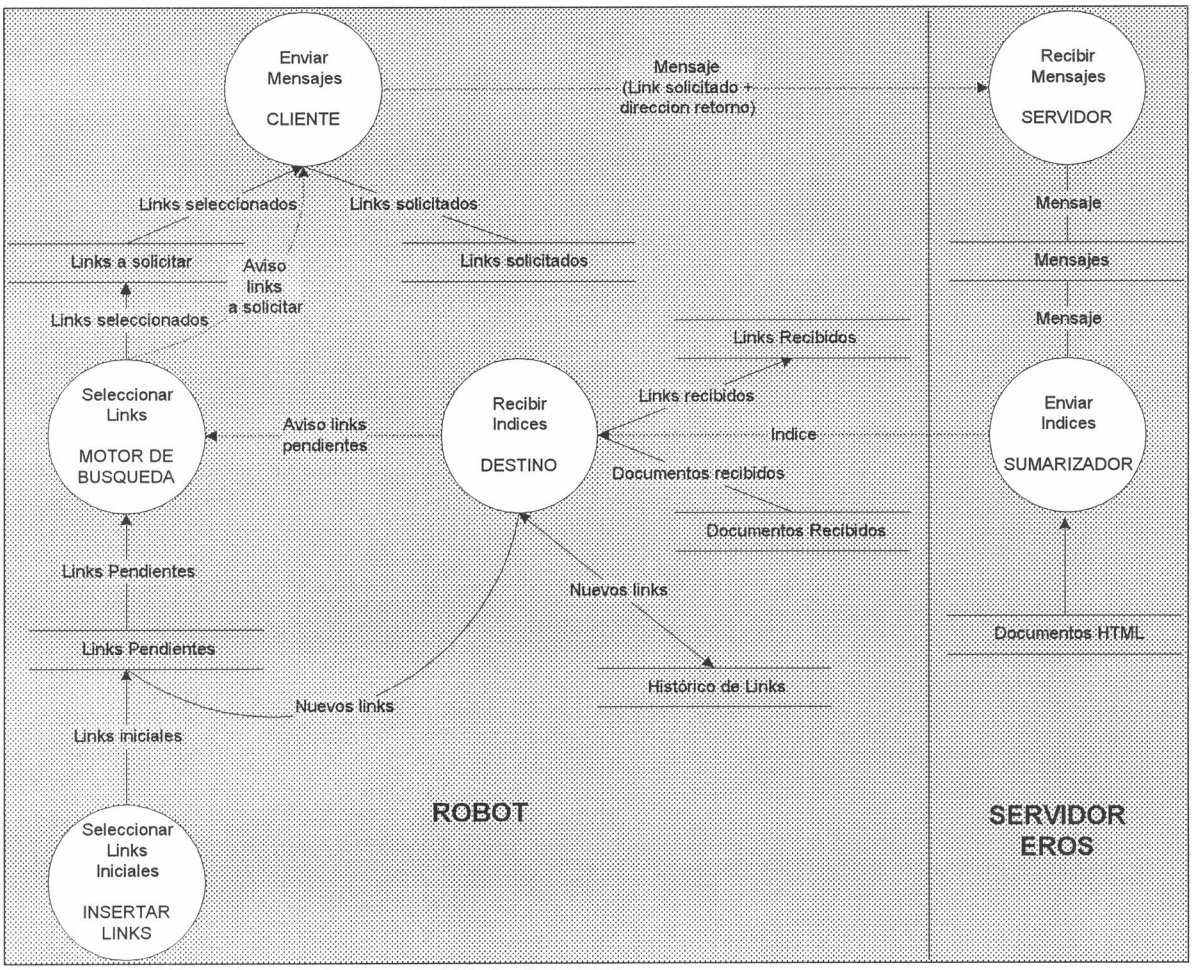


Figura 23: Funcionamiento del Robot (DFD)

Como vemos en el diagrama, el robot consta de cuatro procesos, tres de los cuales (cliente, motor de búsqueda y destino) corren permanentemente y en forma paralela.

El proceso *insertar_links* simplemente permite agregar URL´s dentro del archivo *links_pendientes*. Es mediante este proceso que se insertan los URL´s conocidos para que pueda comenzar a funcionar el robot.

El proceso *motor de búsqueda* tiene como objetivo elegir cual va a ser el próximo link a visitar, decidiendo entre los que se encuentran en *links_pendientes*. En nuestra implementación, por cuestiones de simplicidad, el motor de búsqueda utiliza el archivo *links_pendientes* en forma de cola, eligiendo siempre el primero. Sin embargo, es en este

proceso donde uno establece la política de visita a los distintos servidores EROS. Una buena alternativa es mantener la fecha y la hora de las visitas a los servidores y en base a éstas determinar el siguiente URL a visitar, tratando de distribuir los pedidos en forma equitativa entre los servidores. Los links seleccionados por el motor de búsqueda son almacenados en el archivo *links_a_solicitar*.

El proceso *cliente* es el que, finalmente, solicita a un servidor EROS la información de uno de sus recursos indicado en un URL, que es tomado del archivo *links_a_solicitar*. Esto lo hace enviando un mensaje de requerimiento a un servidor, que incluye los parámetros con que se inició el robot.

En algún momento el servidor sumariza el recurso solicitado en la forma en que lo indica el mensaje y le envía la sumarización al robot, más específicamente al proceso *destino*. Este proceso se encarga de almacenar la información recibida y de actualizar el archivo *links_recibidos*.

El robot obtiene la información enviada por los servidores. No es su objetivo hacer un índice de la Web, sino obtener la información para poder hacerlo. Esta puede ser utilizada por otro proceso que se encargue de generar el índice a partir de las sumarizaciones.

A continuación se describen más detalladamente cada uno de los procesos del robot.

5.1. El proceso Cliente

Los pasos a seguir por el proceso cliente son:

- Tomar el próximo link de la cola links_a_solicitar.
- Obtener el nombre del servidor y de la página que se quiere sumarizar.
- Armar el mensaje, que contiene el nombre de página obtenido, la dirección de retorno y los requerimientos del robot.
- Enviar el mensaje al servidor.

- Eliminar el link de la cola links_a_solicitar.
- Agregar el link en el archivo links_solicitados.

Veamos un esquema del cliente y de los componentes con los que interactúa:

Figura 24: Proceso cliente - Arquitectura

Ejemplo:

1) El cliente toma el primer link de la cola 'links a solicitar', arma el mensaje y lo envía. En este caso el robot está configurado para obtener las 100 palabras claves, los links embebidos en el documento y los títulos.

Figura 25: Cliente - Transmisión del mensaje

2) Una vez enviado el mensaje, el cliente elimina el link de la cola links_a_solicitar y lo inserta en el archivo links_solicitados. Si el mensaje llega correctamente al servidor, éste lo inserta en su cola de mensajes.

Figura 26: Cliente - Luego de la transmisión

Por último se muestra el pseudocódigo.

```
Repetir
        Si no existen links a solicitar entonces
                Suspender proceso
        Finsi
        Obtener próximo link a solicitar
        Armar mensaje
        Enviar mensaje
        Grabar link en links_solicitados
        Eliminar link de links_a_solicitar
Por siempre
```

5.2. El motor de búsqueda

La tarea del proceso *motor de búsqueda* es elegir links del conjunto de links que va obteniendo el robot e insertarlos en la cola del cliente.

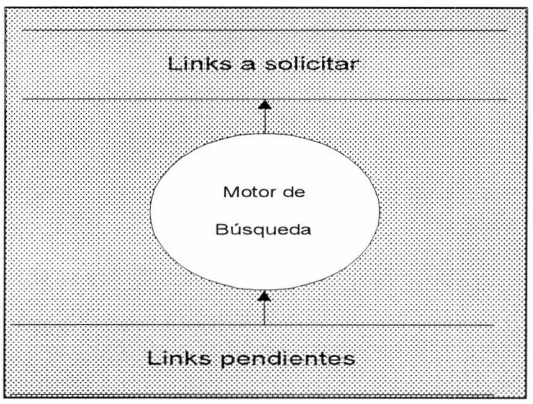


Figura 27: Motor de Búsqueda - Arquitectura

El motor de búsqueda elige de acuerdo a algún criterio cuál será el próximo link a visitar. En el siguiente ejemplo, el motor de búsqueda elige el link http://www.ser2.com/index.html como próximo link a visistar, ubicándolo en la cola del cliente ('URL's a visitar') y eliminándolo de la cola de links pendientes ('URL's pendientes').

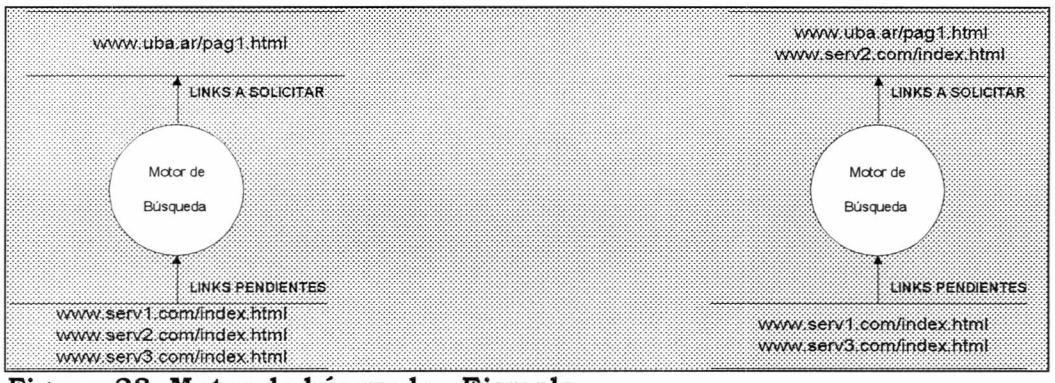


Figura 28: Motor de búsqueda - Ejemplo

A continuación se muestra el pseudocódigo del motor de búsqueda.

```
Repetir
        Si no existen links pendientes entonces
              Suspender proceso
        Finsi
        Obtener Link de links_pendientes
        Grabar link en links_a_solicitar
        Eliminar link de links_pendientes
        Avisar al proceso cliente la existencia de nuevos links
Por siempre
```

5.3. El proceso destino

Este proceso espera las respuestas de los servidores. Cuando llega un mensaje, actualiza el archivo links_recibidos. También actualiza el archivo links_pendientes si llega un mensaje con los links embebidos de algún documento (retroalimentación).

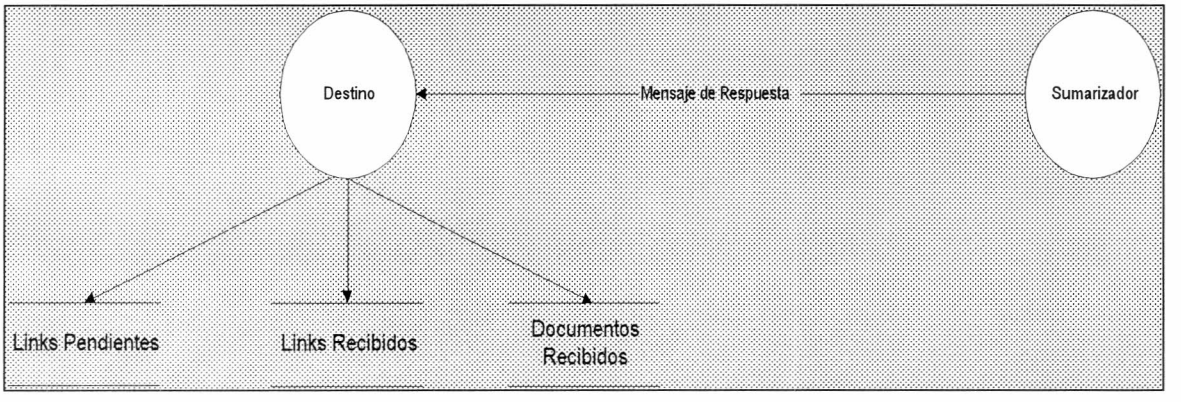


Figura 29: Destino - Arquitectura

Si el mensaje de respuesta contiene la sumarización de un documento, el proceso destino graba su contenido en un archivo local (documentos recibidos). Además inserta el URL que llegó en links_recibidos.

Si el mensaje de respuesta contiene los links embebidos en un documento, el proceso destino graba esos links en el archivo links_pendientes. También inserta el URL que llegó en links_recibidos.

El pseudocódigo del proceso destino es el siguiente.

```
Repetir
        Esperar llegada de nuevo mensaje
        Grabar URL del nuevo mensaje en links_recibidos
        Si el mensaje contiene LINKS
                Grabar nuevos links en links_ pendientes
        Sino (contiene texto)
                Grabar texto en documentos recibidos
        FinSi
Por Siempre
```

5.4. Conclusiones

Realizamos un robot que utiliza el servicio EROS para obtener los recursos de la Web. El robot es muy sencillo y sólo pretende ilustrar como se utiliza el servicio.

La principal desventaja de este robot es que todas las respuestas de los servidores EROS son enviadas a un único proceso (destino) en una única máquina, la misma que inició al robot. Esto puede provocar un cuello de botella en las cercanías de dicha máquina.

La alternativa que soluciona este problema es realizar un robot que corra en varias máquinas. Más específicamente, se podría desarrollar un robot en donde el proceso *destino* se encuentre en diferentes máquinas mientras que los procesos *cliente* y *motor de búsqueda* se encuentren en una sola. El proceso *motor de búsqueda* administra mediante alguna política a qué proceso destino debe responderle cada servidor EROS ante el pedido de un recurso. Al quedar la información en las distintas máquinas en donde se encuentra el proceso destino se necesita un nuevo proceso que recupere dicha información y le permita al robot descubrir nuevos recursos.

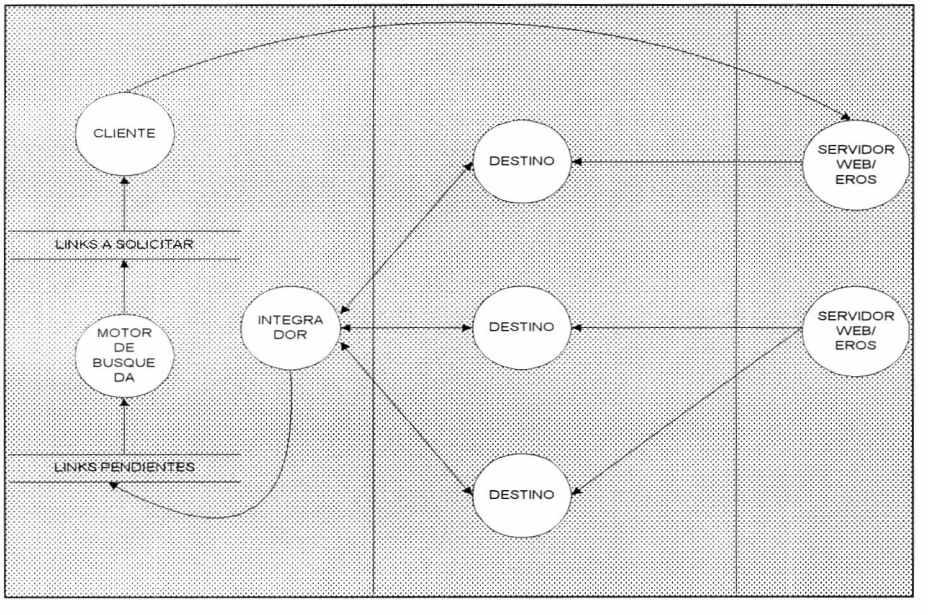


Figura 30: Robots con múltiples destinos

6. CONCLUSIONES Y TRABAJOS FUTUROS

El servicio EROS ofrece a los robots la posibilidad de recuperar la parte que necesitan de los recursos de la Web de un modo eficiente y sin perjudicar a los usuarios de los servidores Web ni utilizar ancho de banda innecesario.

El servicio es muy sencillo. No consume muchos recursos de los servidores Web. Esto es muy importante ya que permite que en el futuro el servicio sea utilizado por gran parte de los servidores Web y así lograr que los robots lo utilicen en lugar de seguir accediendo a los servidores mediante el protocolo HTTP.

Los servidores Web se ven beneficiados al poder responder a los robots en momentos de poca actividad, lo que les permite brindar un mejor servicio a sus usuarios.

Los robots que utilizan el servicio son un poco más complejos de desarrollar ya que necesitan actuar como clientes y servidores al mismo tiempo. Los robots se ven beneficiados al poder reccorrer la Web mucho más rápidamente que si lo hacen recuperando recursos mediante HTTP.

Si bien pueden surgir alternativas al uso de robots para la generación de índices de la Web, no nos parece que estos dejen de existir en un futuro cercano. Tampoco es factible que los diferentes robots se unan para compartir tareas con el objetivo de utilizar menos recursos de la Web debido a que cada uno posee su propio algoritmo de indexación y sus intereses económicos.

Hay varias extensiones y mejoras que se le pueden hacer al servicio EROS.

6.1. Pedidos por servidor

Una de las posibles extensiones es que el servidor retorne una sumarización de todos sus documentos a partir de un pedido de los robots. En otras palabras, permitir a

los robots hacer pedidos por servidor, en lugar de hacerlo por documento. De esta manera se ahorra una gran cantidad de mensajes enviados al servidor. Por ejemplo, si un servidor dispone de 1000 documentos, los robots que hacen pedidos por documentos enviarían aproximadamente 1000 mensajes al servidor, mientras que los robots que hacen pedidos por servidor sólo enviarían 1.

La cantidad de mensajes que se envían al servidor al solicitar documentos no es tan grave como parece ya que:

- Los mensajes enviados son muy pequeños.
- Se trabaja con el protocolo UDP, por lo que no hay conexiones.
- El tiempo que consume el proceso servidor, que es el que recibe los mensajes, es casi despreciable.

Por otro lado si se hacen pedidos por servidor, éste enviaría la sumarización de todos sus documentos, mientras que si se hacen pedidos por página sólo se enviarán las sumarizaciones de los documentos alcanzables a partir de un hipervínculo conocido por el robot.

Se abren dos alternativas en cuanto a cómo el servidor debe retornar las sumarizaciones de los documentos: cada vez que sumarice un documento (de la misma manera en que se hace actualmente) o que espere a sumarizar todos los documentos y los envíe de una sola vez. Esta última alternativa trae la ventaja de enviar un solo mensaje en lugar de uno por documento. Por otro lado, tiene un grave problema; en general, el robot va a tener que esperar mucho tiempo para obtener una respuesta del servidor.

Una desventaja de realizar pedidos por servidor en lugar de por documento es que es una solución menos genérica. Un determinado robot puede querer un tipo de respuesta distinto para diferentes documentos, o simplemente no desear sumarizar todos los documentos de un servidor.

El problema más grave es la sobrecarga que se produciría en el servidor, debido al seguimiento que debe hacer para cada robot de qué documento es el próximo a sumarizar, y en cuanto a la política del orden en que se deben atender los mensajes.

Lo más lógico sería pensar en que el servidor permita tanto pedidos por documentos como por servidor y que sean los robots los que decidan.

En conclusión, la extensión al servidor para que soporte pedidos por servidor parece posible, siempre y cuando se logre una implementación que no consuma muchos recursos de los servidores, ni perjudique a los robots (por ejemplo, en el tiempo de respuesta), ya que esto impediría la universalidad de la solución.

6.2. Robots cooperativos

Uno de los mayores problemas que se presentan es el del crecimiento exponencial de la Web y como consecuencia, el tiempo que les lleva a los robots recorrerla. Otro gran problema es el cuello de botella que se provoca en las cercanías de las máquinas en que corren los robots por toda la información transmitida hacia ellas.

La solución más lógica es realizar robots cooperativos. Es decir distribuir los robots en diferentes máquinas tocándole a cada uno de ellos indexar un subespacio de la Web. De alguna manera los robots se deberían comunicar para obtener los resultados de los otros, sobre todo en cuanto a la obtención de nuevos links a investigar.

El servicio realizado se brinda totalmente a una solución como esta ya que los robots pueden indicarle a los servidores donde enviar las respuestas, sin que necesariamente sea a la máquina en donde corre el robot.

6.3. Servidor de búsqueda

Otro trabajo importante es realizar un servidor de búsqueda. Es decir, un robot que utilice el nuevo servicio para obtener la información, un proceso que genere una estructura eficiente para la posterior búsqueda y una interface para la Web que permita buscar en esta estructura.

6.4. Mejoras al servidor

Hay muchas mejoras y extensiones que se le pueden hacer al servidor. Una extensión posible es que el servidor pueda indexar diferentes tipos de objetos y no solamente documentos html. Una tarea a realizar es mejorar el parser de html.

Apéndice A: Internet

Internet puede definirse como una red de redes de computadoras, de alcance mundial. Las redes que se encuentran interconectadas abarcan desde grandes instalaciones gubernamentales, hasta redes universitarias, de organizaciones sin fines de lucro, comerciales, etc. [COM/95].

Haciendo una breve reseña histórica, en 1969 nace lo que luego se llamaría Internet, como un proyecto de la Agencia de Proyectos de Investigación de Defensa Avanzados de E.E.U.U. (DARPA). Este proyecto surgió ante la necesidad de intercambiar información entre investigadores y científicos militares ubicados en sitios distantes.

Inicialmente la red contaba con cuatro computadoras interconectadas. En 1972 ya se habían conectado 37 y la red pasó a llamarse ARPANET.

En Enero de 1983 TCP/IP se convierte en el único protocolo oficial. A partir de ese momento crece rápidamente el número de redes, máquinas y usuarios conectados a ARPANET. Cuando ARPANET se interconecta con NSFNET[1], el crecimiento se hace exponencial. El nombre de Internet surge a mediados de los '80 haciendo referencia a la red de redes que se había conformado.

La mayor parte del crecimiento de Internet se debe a la conexión a ésta de redes existentes (BITNET[2], EARN[3], entre otras).

---

[1] National Science Foundation (Fundación Nacional para la Ciencia). Es una agencia federal de EEUU, responsable de auspiciar la investigación y la educación en ciencia e ingeniería.

[2] Because It´s Time Network. Fue creado para conectar mainframes IBM en la Universidad de New York.

[3] European Academic and Research Network (Red Académica y de Investigación Europea)

En la siguiente tabla se muestra el crecimiento de Internet en cuanto a la cantidad de redes [CHE/96].

Fecha	Total de Redes
Julio 88	217
Enero 89	384
Julio 89	650
Enero 90	1233
Julio 90	1727
Enero 91	2338
Julio 91	3086
Enero 92	4526
Julio 92	6031
Enero 93	9117
Julio 93	14121
Enero 94	23494
Julio 94	36153
Enero 95	46318

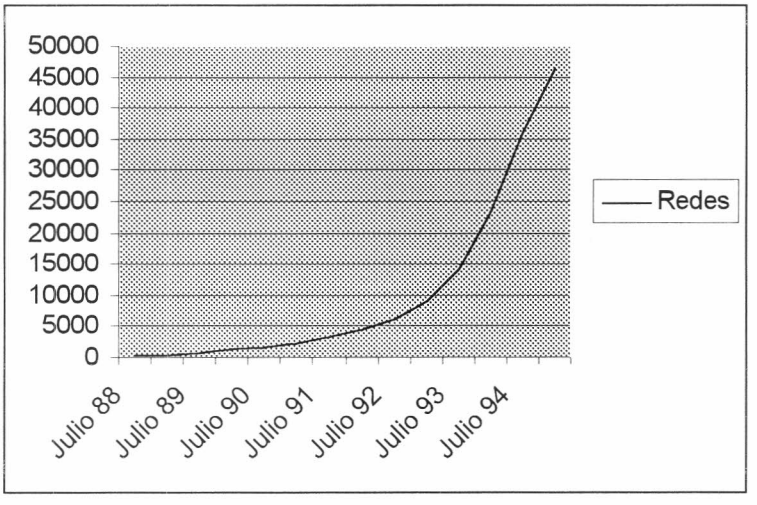


Figura 31: Crecimiento de redes en Internet

En cuanto al crecimiento de Hosts en Internet se obtuvieron los siguientes datos [CHE/96]:

Fecha	Host
Agosto 81	213
Agosto 83	562
Octubre 85	1961
Diciembre 87	28174
Octubre 89	159000
Octubre 90	313000
Octubre 91	617000
Octubre 92	1136000
Octubre 93	2056000
Octubre 94	3864000
Julio 95	6642000
Julio 96	12881000
Julio 97	19540000

El crecimiento es exponencial, como se ve más claramente en el siguiente gráfico.

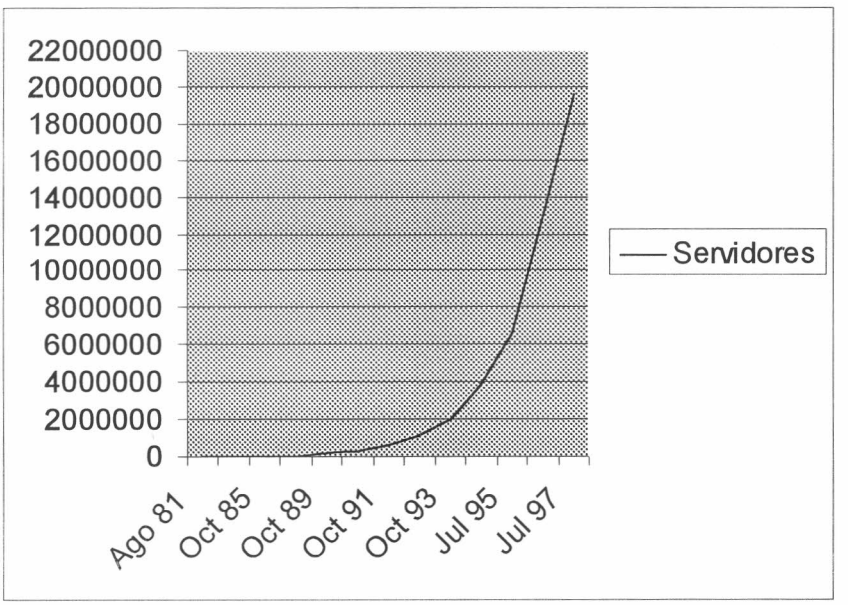


Figura 32: Crecimiento de Hosts en Internet

Lo que mantiene unida a Internet es el conjunto de protocolos TCP/IP. Una máquina pertenece a Internet si utiliza TCP/IP, tiene una dirección IP y tiene la posibilidad de enviar paquetes IP a todas las otras máquinas conectadas a Internet.

Las aplicaciones más corrientes de Internet eran el correo electrónico, el servicio de noticias, el login remoto y la transferencia de archivos.

Esto era así hasta la aparición de la World Wide Web. Esta aplicación se basa en la idea de los hipertextos y su facilidad de uso por parte de cualquier usuario ha hecho crecer a Internet en una forma mucho más rápida.

Después de sus comienzos, el tráfico en la Web ha crecido dos veces el promedio de crecimiento de Internet. En 1994 el tráfico de WWW sobre la red creció un 1500% [CHE/96].

TCP/IP

El conjunto de protocolos TCP/IP permite a las computadoras comunicarse entre sí. No interesa si se encuentran dentro de la misma red física o no. Tampoco interesa si una computadora es una Cray y otra una Macintosh [COM/91a].

Es decir, TCP/IP es un estándar que hace de puente entre computadoras que pueden no poseer el mismo hardware o el mismo sistema operativo. Es el protocolo que maneja a la Internet.

Internet es *una red virtual* porque da la ilusión de ser una sola y gran red. Aunque Internet es una red de redes, el software IP se encarga de los detalles y permite que los usuarios piensen en "Internet" como si fuera una sola red. No se dan cuenta de los *ruteadores* y redes que la forman [COM/95].

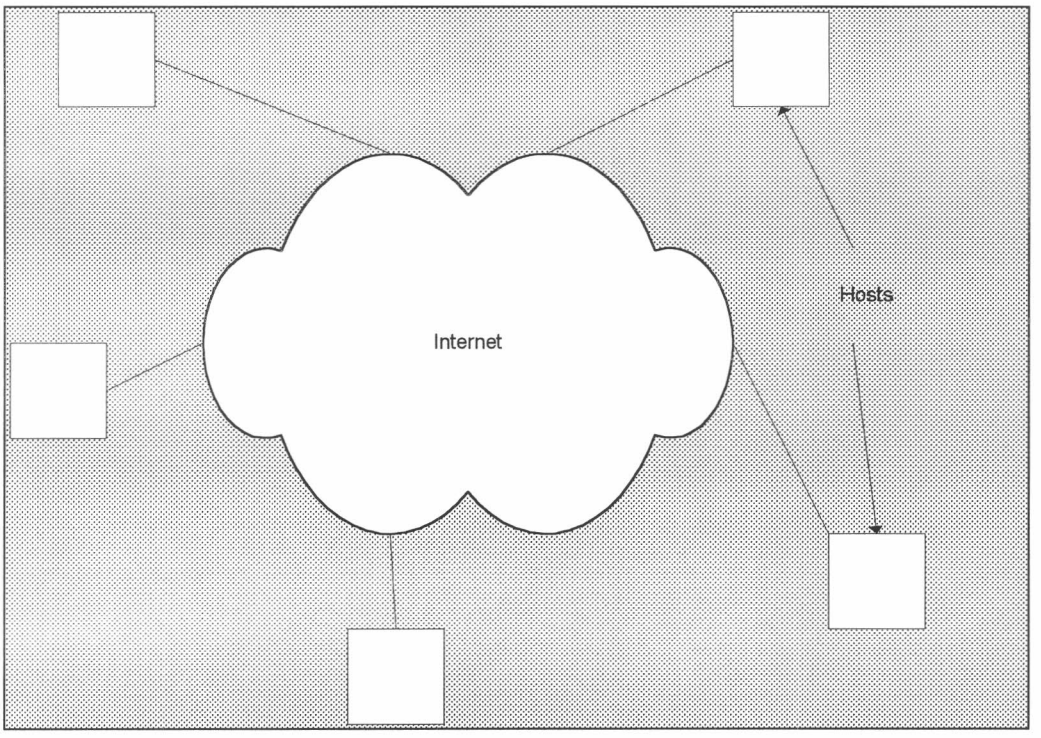


Figura 33: Internet vista por el usuario

La razón de la importancia de TCP/IP es que permite a redes independientes interconectarse con la Internet. Las redes están conectadas por ruteadores (gateways), que son computadoras que transfieren paquetes de datos desde una red a otra. Mediante TCP/IP, la información viaja en unidades discretas denominadas *paquetes* o *datagramas* IP.

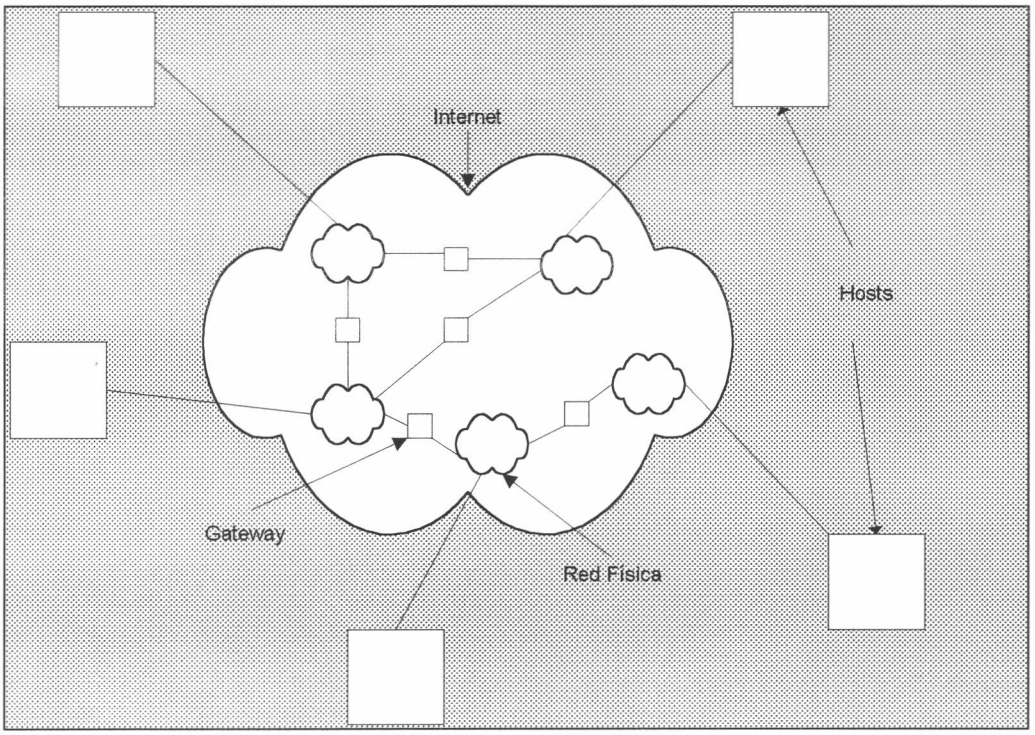


Figura 34: Internet internamente

Cuando un datagrama viaja a través de Internet de una computadora a otra, tiene que seguir una vía física. En cada paso, a lo largo de la vía, el datagrama viaja, ya sea a través de una red física o de un ruteador, hacia otra red. En cierto momento el datagrama alcanza su destino final.

El software TCP/IP oculta a estos ruteadores y a la arquitectura de red subyacente y permite ver a cualquier máquina como si fuera parte de la misma red física.

El protocolo más importante es IP, que se encarga de transmitir datagramas por la Internet, realizando la fundamental tarea del ruteo, es decir, eligiendo el camino que deben seguir los datagramas de un extremo al otro y usando a los ruteadores para llegar a las redes.

TCP es un protocolo de alto nivel que permite el intercambio de datos entre aplicaciones que corren en diferentes hosts. Divide a los datos en segmentos y los

transmite usando IP. En la mayoría de los casos un segmento TCP es enviado en un datagrama IP. Si es necesario, TCP divide los segmentos en múltiples datagramas. Ya que IP no garantiza que los datagramas sean recibidos en el mismo orden en el que se enviaron, TCP reacomoda los segmentos en el receptor formando así un tubo de datos.

Otro miembro del conjunto de protocolos TCP/IP es *UDP*. Se dice que TCP es un protocolo confiable, ya que realiza chequeo de errores y handshaking para verificar que los datos lleguen en forma correcta. En cambio, UDP no lo es; no garantiza que se mantenga el orden en el que se enviaron los datagramas, es más, ni siquiera garantiza que estos lleguen. Si uno necesita confiabilidad utilizando UDP es necesario ubicarla dentro de las aplicaciones. De todas formas, UDP es usado por un gran número de aplicaciones.

Otros protocolos importantes dentro de TCP/IP son *ARP* (Address Resolution Protocol) que traduce direcciones IP en direcciones físicas de red, tal como direcciones Ethernet; *RARP*, que hace lo contrario, es decir, convierte direcciones físicas en direcciones IP e *ICMP* (Internet Message Protocol) que utiliza a IP para comunicaciones de mensajes de control y errores. Si un ruteador no puede enviar un datagrama IP, usa ICMP para informar al emisor la existencia de un problema.

A continuación vemos la arquitectura de TCP/IP: [STE/94].

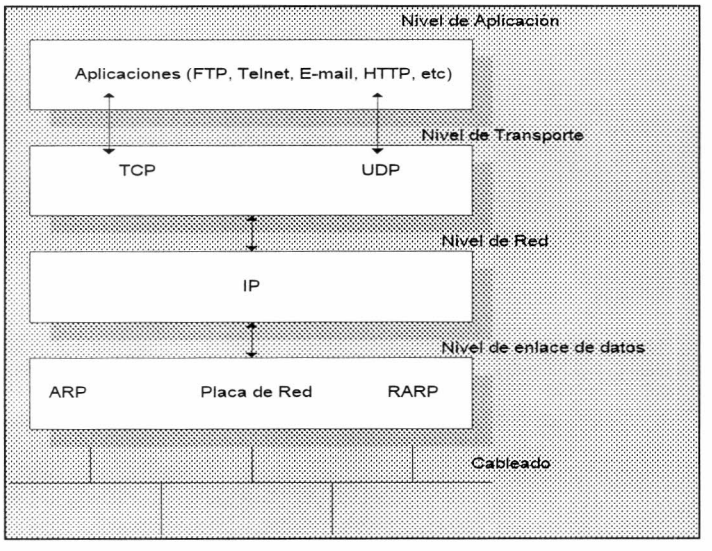


Figura 35: Arquitectura TCP/IP

Modelo Cliente/Servidor

La principal forma de interacción entre aplicaciones es a través del paradigma conocido con el nombre de *cliente/servidor*. Este modelo forma la base de la mayor parte de las comunicaciones de redes [COM/93].

El término servidor se refiere a cualquier programa que ofrece un servicio que pueda ser alcanzado a través de la red.

Los servidores aceptan pedidos que llegan tomando como medio a la red, procesan el pedido y devuelven un resultado al generador del pedido.

El cliente es un programa que envía una solicitud a un servidor y se queda esperando por una respuesta. El cliente y el servidor se comunican usando un lenguaje común llamado *protocolo*.

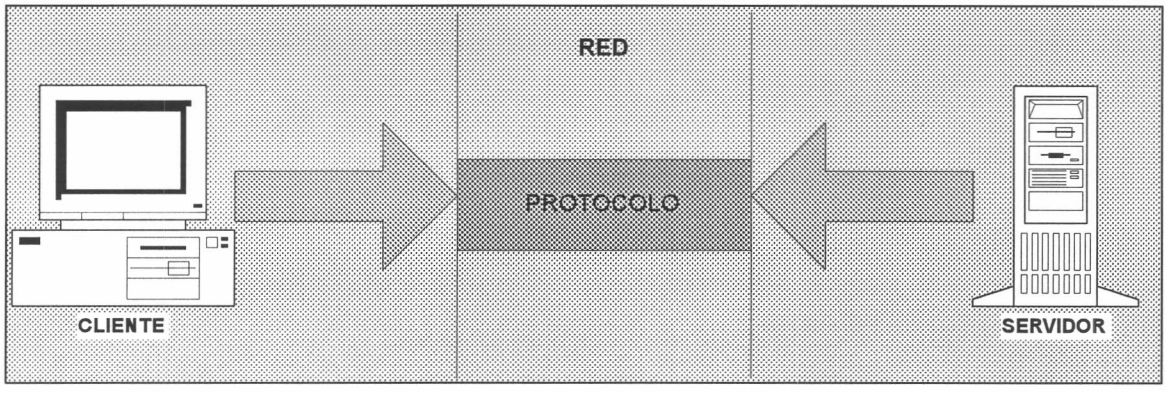


Figura 36 : Relación entre cliente y servidor

Los servidores dentro de Internet ofrecen una gran gama de servicios que van desde los muy simples como brindar la hora del día hasta los muy complejos como resulta ser un servidor de archivos.

Es común implementar a los servidores como programas del nivel de aplicación. La ventaja de esto es que pueden funcionar en cualquier computadora que soporte los

protocolos TCP/IP (tanto en un sistema de multiprocesamiento como en una PC con un sistema operativo monoprocesamiento).

Servidores iterativos vs concurrentes

Los *servidores iterativos* son los que procesan un pedido a la vez mientras que los *servidores concurrentes* pueden manejar múltiples pedidos al mismo tiempo (utilizando varias copias del programa para permitir que muchos usuarios accedan al servicio al mismo tiempo).

Los servidores iterativos son más fáciles de programar pero tienen una peor performance ya que hacen esperar a los clientes que piden el servicio si en ese momento se encuentran ocupados. Por el contrario, los servidores concurrentes son más complejos, pero permiten responder a varios clientes simultáneamente.

Servidores orientados a conexión vs sin conexión

La alternativa entre una implementación orientada a conexión y una sin conexión depende del protocolo de nivel de transporte utilizado.

Los servidores que usan TCP como protocolo de transporte son orientados a conexión, mientras que los que usan UDP ofrecen un servicio sin conexión.

Los servidores orientados a conexión son más fáciles de programar ya que TCP se encarga automáticamente de los problemas de paquetes que llegan fuera de orden, paquetes duplicados, etc. Estos tipos de servidores usan el concepto de conexión. Aceptan conexiones entrantes de los clientes y luego se comunican con estos a través de la conexión.

La desventaja más evidente de los servidores orientados a conexión es justamente el uso de recursos para el manejo de la conexión. En aplicaciones pequeñas, TCP produce

una gran sobrecarga al utilizar el método de 3-way handshake tanto para establecer una conexión como para terminarla.

Otro problema importante es cuando se rompe la conexión. Supongamos que un cliente establece una conexión con un servidor, intercambia ciertos mensajes y luego se cae. El cliente no manda más pedidos al servidor y el servidor tampoco envía más información al cliente ya que ha respondido a todos los pedidos. El problema es que el servidor sigue manteniendo toda la información de estado de la conexión. Este inconveniente se hace evidente si uno o más clientes se caen a menudo con lo que el servidor puede quedar rápidamente sin recursos.

Los servidores sin conexión no pueden depender del servicio de transporte para la confiabilidad de la transmisión. Generalmente los clientes toman la responsabilidad de retransmitir los pedidos si no llega una respuesta, lo que es bastante difícil de programar.

Servidores como clientes

Los programas no siempre caen exactamente en la definición de cliente o servidor. Un programa servidor puede necesitar acceder a servicios de red que requieran que actúe como cliente.

Sockets

Los *sockets* son un mecanismo de intercomunicación entre procesos que permite la comunicación entre estos aunque se encuentren en diferentes máquinas [GRA/97].

La comunicación entre procesos mediante sockets se basa en el modelo cliente/servidor. Un proceso, denominado el proceso servidor, crea un socket, cuyo nombre es conocido por los procesos clientes. Los procesos clientes pueden conversar con el proceso servidor estableciendo una conexión al socket de éste. Para lograrlo, el proceso cliente crea un socket sin nombre y luego realiza el requerimiento de conexión con el

socket con nombre que tiene el servidor. Una conexión exitosa retorna un descriptor al cliente y uno al servidor, los cuales pueden ser utilizados para escribir y leer.

Una vez que un socket ha sido creado, puede ser usado para esperar por conexiones entrantes o iniciar una conexión. Un socket usado por un servidor para esperar por conexiones entrantes es llamado *socket pasivo*, mientras que un socket usado por un cliente para iniciar una conexión es llamado *socket activo*.

Hay dos tipos de sockets: los *stream sockets* y los *datagram sockets*, que difieren principalmente en el protocolo de transporte que utilizan, TCP y UDP respectivamente.

Stream sockets: Estos tipos de sockets son confiables. Cuando se usan, los datos son entregados en orden, en la misma secuencia en la que fueron enviados. No hay duplicación de datos y se hace control de errores y de flujo. Permiten una comunicación bidireccional (full-duplex) y son orientados a conexión. Es decir, una conexión lógica es creada por los dos procesos que usan el socket. La información concerniente a la conexión se establece antes de la transmisión de los datos y es mantenida por cada extremo de la conexión durante la comunicación. Los datos son transmitidos como un tubo de bytes.

Datagram Sockets: Estos tipos de sockets no son confiables. Es decir, los datos que se reciben pueden llegar fuera de orden. Soportan comunicación bidireccional, pero son considerados sin conexión. No hay una conexión lógica entre los procesos emisor y receptor. Cada datagrama es enviado y procesado independientemente. Los diferentes datagramas pueden llegar al mismo destino por diferentes rutas. No hay control de flujo y el control de errores es mínimo. El tamaño de los datagramas es pequeño.

En la siguiente figura podemos ver una secuencia de llamados al sistema para un cliente y un servidor usando TCP. El servidor está en un loop infinito esperando por nuevas conexiones en un port conocido. Acepta cada nueva conexión, procesa el requerimiento del cliente, y luego cierra la conexión.

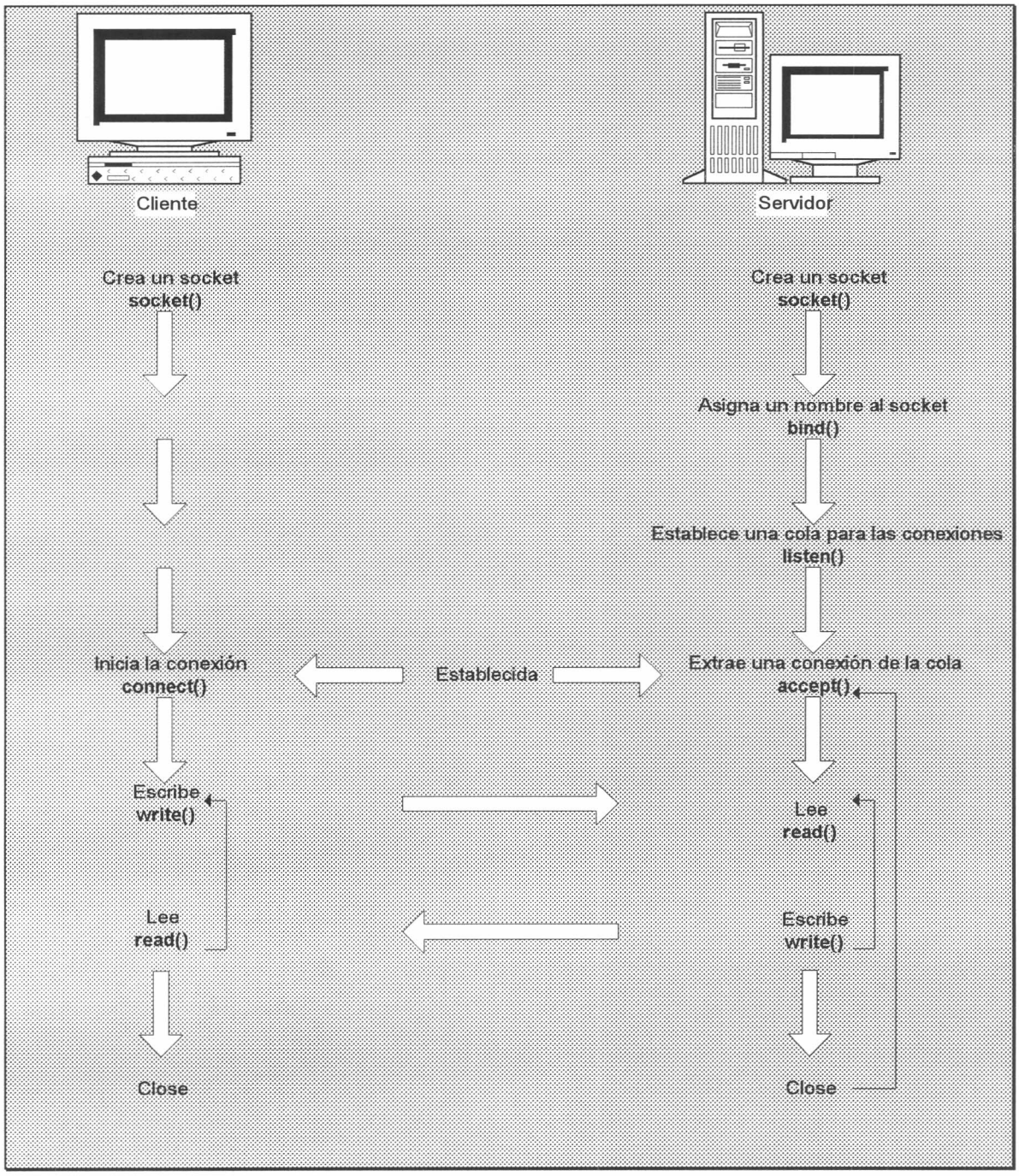


Figura 37: Conexión mediante sockets

El cliente crea un socket, llama a *connect* para conectarse al servidor, y luego interactúa usando *write* para enviar requerimientos y *read* para recibir respuestas. Cuando termina de usar la conexión hace un *close*.

Un servidor usa *bind* para especificar el port local bien conocido que usará, llama a *listen* para setear la longitud de la cola de la conexión, y luego entra en loop. Dentro del loop, el servidor hace *accept* para esperar hasta que el siguiente requerimiento de conexión llegue, usa *read* y *write* para interactuar con el cliente, y finalmente usa *close* para terminar la conexión. El servidor luego retorna al *accept* donde espera para la siguiente conexión [COM/93].

Apéndice B: La World Wide Web

La World Wide Web, es un servicio ofrecido por una gran cantidad de servidores conectados a Internet, y, básicamente, es un sistema de *hipertexto distribuido*.

Se llama hipertexto a un conjunto de documentos relacionados entre sí. Cada documento contiene partes de texto y un conjunto de referencias (comúnmente llamadas *links*) a otros documentos.

Un ejemplo muy común de hipertexto es la ayuda que hay en muchos de los programas utilizados actualmente. Cuando uno pide ayuda de un determinado tema, se despliega una ventana con la información de dicho tema y aparecen referencias (generalmente partes de texto subrayadas) a otros documentos que tienen algo que ver. Este es un ejemplo de hipertexto centralizado, ya que todos sus documentos se encuentran ubicados en la misma máquina.

En cambio, la Web, como dijimos anteriormente, es un sistema de hipertexto distribuido. Los documentos, dentro de ella, están distribuidos en miles de máquinas a través de todo el mundo.

La Web no es sólo un sistema de hipertexto. También es un sistema de *hipermedia*, ya que los documentos contienen texto, imágenes, sonidos, etc. De todos modos nos concentraremos solamente en el aspecto hipertextual de la Web.

Se dice que uno navega por la Web, por el hecho de que va pasando de un hipertexto a otro a través de los links.

Existen programas llamados *browsers* (navegadores), que nos permiten obtener los documentos insertos en la Web e ir navegando fácilmente de unos a otros.

La Web es un servicio que ha hecho que Internet crezca enormemente en cuanto a cantidad de usuarios, servidores y recursos que estos ofrecen. La Web crece en mayor proporción que la Internet misma [GRA/96].

FECHA	SERVIDORES WEB
Junio 93	130
Diciembre 93	623
Junio 94	2738
Diciembre 94	10022
Junio 95	23500
Enero 96	100000
Diciembre 96	230000
Enero 97	650000

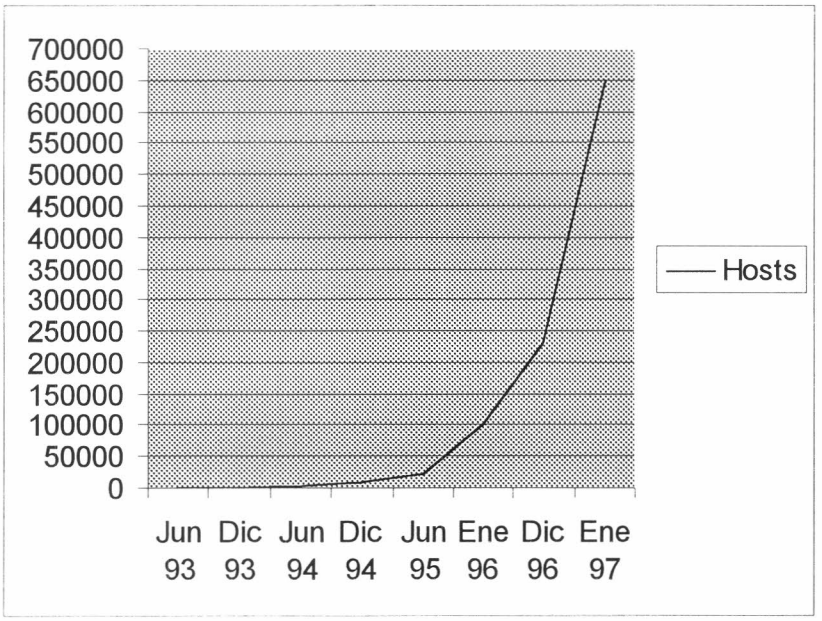


Figura 38: Crecimiento de los servidores Web

Otro modo de ver el crecimiento de la Web es comparándolo con el uso de otros servicios [GRA/96].

Fecha	%FTP	%TELNET	%NEWS	%IRC	%GOPHER	%EMAIL	%WEB
6/93	42,9	5,6	9,3	1,1	1,6	6,4	0,5
12/93	40,9	5,3	9,7	1,3	3,0	6,0	2,2
6/94	35,2	4,8	10,9	1,3	3,7	6,4	6,1
12/94	31,7	3,9	10,9	1,4	3,6	5,6	16,0
3/95	24,2	2,9	8,3	1,3	2,5	4,9	23,9

A continuación se muestra la arquitectura de la Web.

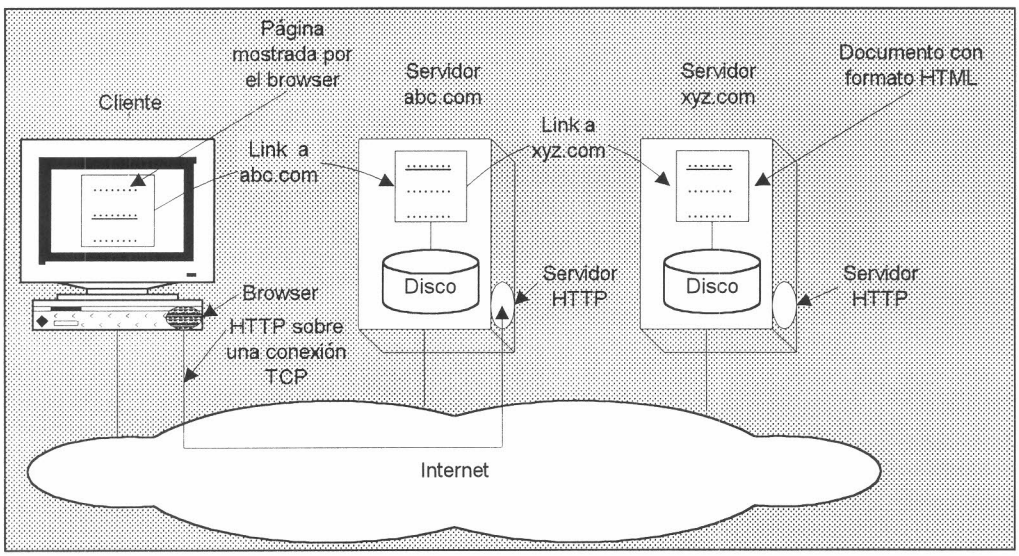


Figura 39: Elementos de la Web

URL´s

Cada documento en la Web (y en general, en cualquier servicio de Internet) tiene una única dirección. Esta dirección es conocida con el nombre de *URL*, que significa Ubicador Uniforme de Recursos (Uniform Resource Locator).

Un URL tiene la siguiente estructura:

esquema: parte específica del esquema

El esquema describe el tipo de objeto al que hace referencia el URL. La parte específica del esquema es particular a los distintos tipos de esquemas.

Algunos de los esquemas más utilizados son:

http	HyperText Transfer Protocol
ftp	File Transfer Protocol
gopher	Protocolo Gopher
mailto	Direcciones de e-mail
news	News de USENET
telnet	Protocolo Telnet
file	Archivos locales

El esquema más utilizado en la Web es *HTTP*. Este es usado para acceder a documentos guardados en un servidor HTTP y tiene dos formatos posibles:

http://servidor:port/camino#fragmento
http://servidor:port/camino?cadena_de_búsqueda

El servidor

El servidor es el nombre único dentro de Internet o la dirección numérica IP de la computadora que posee el recurso de la Web. El nombre consiste de varias partes, incluyendo el nombre actual del servidor y los sucesivos nombres de sus dominios. Nombres típicos de Internet pueden ser *www.uba.ar* o *hoohoo.ncsa.uiuc.edu*. Ejemplos de direcciones IP son 125.234.1.86 o 192.249.1.35.

El port

El port es el número del port de comunicación al que el cliente se conecta dentro del servidor. Los servidores realizan muchas tareas además de servir documentos de la Web: e-mail, FTP, telnet, compartimiento del file system, etc. Esta actividad se divide en ports atendidos por diferentes softwares. La mayoría de los protocolos asignan un port por default en el que se realiza la conexión en caso de no especificarlo. El port por default del protocolo HTTP es 80. En caso de que el servidor no use este número de port se debe añadir luego del nombre del servidor, por ejemplo: www.serv.com: 81.

El camino

El camino del documento es la ubicación jerárquica del archivo dentro del servidor. Consiste en uno o más nombres separados por (/). Todos, excepto el último nombre representan los directorios sucesivos en los que se encuentra el documento. El último nombre es generalmente el mismo documento.

Se utilizan ciertas convenciones. Por ejemplo, si el último nombre dentro del camino indica un directorio en lugar de un documento, el servidor le envía al cliente un documento por defecto de ese directorio. Si no se indica ningún nombre de directorio el servidor envía el primer documento que se encuentra en el directorio raíz del servidor, llamado "*home page*".

El fragmento

El fragmento es un nombre que apunta a una sección de un documento. Le indica al browser que muestre el documento obtenido a partir de ese nombre de fragmento. El fragmento sólo se aplica a documentos con formato HTML.

Los fragmentos son útiles para documentos largos. Dividiendo un documento en fragmentos se hace más fácil para los lectores leer solamente una porción del texto.

La cadena de búsqueda

La cadena de búsqueda es un parámetro opcional dentro de un URL HTTP. Indica que el documento es un recurso de búsqueda o ejecutable en el servidor. El contenido de la cadena es pasado al servidor como un parámetro que controla la función de búsqueda o de ejecución.

Ejemplos de URLs HTTP

Algunos ejemplos de URLs HTTP son:

```
http://www.ora.com/catalog.html
http://www.ora.com/
http://www.kumquat.com:8080/
http://www.kumquat.com/planting/guide.html#soil_prep
http://www.kumquat.com/find_a_quat?state=Florida
```

El primer ejemplo es una referencia explícita a un documento llamado "catalog.html" que se encuentra en el directorio raíz del servidor www.ora.com.

El segundo se refiere a la "home page" del mismo servidor. Esa "home page" puede ser o no el archivo "catalog.html".

El tercer ejemplo, indica la "home page" del servidor www.kumquat.com, y asume que la conexión a la Web se realiza en el port 8080 en lugar del port estándar.

El cuarto ejemplo es un URL para obtener el documento llamado "guide.html" ubicado en el directorio "planting" del servidor www.kumquat.com. Una vez recuperado, se le indica al browser que muestre el documento a partir del fragmento llamado "soil_prep".

El último ejemplo invoca a un recurso ejecutable llamado "find_a_quat" con el parámetro "state" seteado con el valor "Florida". Lo más probable es que dicho recurso genere automáticamente un documento con formato HTML y lo envíe como respuesta.

Protocolo HTTP

HTTP (Hypertext Transfer Protocol o Protocolo de Transferencia de Hipertextos) es el protocolo usado por la Web para obtener documentos de servidores distribuidos. Se basa en el modelo cliente/servidor. El cliente establece una conexión con el servidor y le pide un documento. El servidor procesa el pedido, le envía al cliente una respuesta, y cierra la conexión [LEE/94b].

Dentro de Internet, la comunicación generalmente se lleva a cabo sobre una conexión TCP/IP. HTTP es un protocolo perteneciente a la capa de aplicación ubicado sobre un nivel de transporte confiable como lo es TCP.

El protocolo HTTP se ubica dentro de los protocolos orientados a conexión. Una vez activado, el protocolo comienza un proceso de 4 etapas que son: conexión, requerimiento, respuesta y desconexión.

Al ser un protocolo de requerimiento-respuesta, HTTP requiere tanto de un cliente como de un servidor para completar una transacción. En general, aunque no siempre, los clientes HTTP son los programas llamados Browsers.

La conexión

En esta etapa, el cliente intenta establecer una conexión con el servidor. El cliente obtiene un nombre de dominio o la dirección IP contenida en el URL y trata de conectarse con esa máquina. A menos que se especifique otro port en el URL, HTTP intenta usar el port 80 en el servidor.

El requerimiento

Una vez establecida la conexión empieza la etapa de requerimiento. El cliente hace un requerimiento al servidor enviándole un mensaje ASCII con ciertos comandos específicos.

El mensaje comienza con la línea de requerimiento, que contiene el método a utilizar, el URL y la versión del protocolo.

Los tres métodos más comunes son *Get*, *Head* y *Post*, y, en menor grado, *Put* y *Delete*.

Get, el método más usado, le indica al servidor que envíe un documento u objeto al cliente.

Head le indica al servidor que envíe el documento pero sin el cuerpo de éste, es decir, que envíe información de éste pero no su contenido. Head es usado generalmente para determinar si un link es válido o si un documento fue recientemente modificado.

Post le permite al cliente el envío de información al servidor, por ejemplo para actualizar una base de datos. Lo que se hace con los datos depende del servidor.

Put es similar a Post. El cliente le indica al servidor que guarde cierta información en un archivo. La principal diferencia entre los dos métodos es que con Put el servidor no determina la acción a seguir, por lo que es menos seguro que Post.

Delete le indica al servidor que borre un objeto. Es poco admitido por los servidores, al igual que el método Put, por razones de seguridad.

Luego de la línea de requerimiento vienen las líneas de cabeceras del requerimiento. La cabecera *Accept* identifica que tipo de datos, tal como sonido o video, debe contener la respuesta. *Accept-Charset* especifica el conjunto de caracteres aceptado y *Accept-Encoding* la codificación requerida. *User Agent* ofrece al servidor información acerca del cliente, *If-modified-since* se usa con el método Get, incluye un parámetro de tipo fecha, el recurso es transferido sólo si ha sido modificado luego de dicha fecha.

Un ejemplo de un mensaje de requerimiento es:

```
GET /docs/manual.html HTTP/1.0
Accept: text/plain
Accept: text/html
User-Agent: Browser
```

La respuesta

Una vez que el browser estableció la conexión con el servidor y envió el requerimiento, se queda esperando un mensaje de respuesta. La primera es la línea de estado, seguido por un conjunto de cabeceras y a continuación el documento u objeto requerido.

La línea de estado contiene la versión de HTTP usada por el servidor, un código de estado y un mensaje. Los códigos de estado son números de tres dígitos que indican como fue manejado el requerimiento (si el servidor pudo responder el pedido), mientras que el mensaje es una aclaración del código.

Hay cinco clases de códigos de estado:

1.xx	Reservado para detalles de información
2.xx	Indica éxito
3.xx	El URL fue movido
4.xx	Error del lado del browser
5.xx	El servidor tiene problemas

Algunos de los códigos más frecuentes son:

401	Desautorizado
405	El método no está permitido
503	El servicio no está disponible
200	OK

Las cabeceras indican al cliente qué métodos están permitidos (*Allow*), proveen información acerca del contenido del mensaje (por ejemplo, *Content-Language, Content-Length* y *Content-Type*) y dan detalles acerca de fechas de modificación (*Last-modified*). Estas cabeceras le permiten a los browsers interpretar al archivo o al recurso en forma correcta.

Un ejemplo de una respuesta es:

```
HTTP/1.0 200 OK
Server: MDMA/0.1
MIME-version: 1.0
Content-type: text/html
Last Modified: Thu Jul 7 00:25:33 1994
Content-Length: 2003

****Contenido del archivo***

[Connection closed by foreign host]
```

La desconexión

Esta etapa ocurre al final de la transacción HTTP. Puede ser iniciada por el servidor, una vez que fue transferido el archivo o por el cliente, si el usuario realizó un Stop o inició una nueva transacción HTTP seleccionando un nuevo link.

HTTP no mantiene el estado entre conexiones sucesivas, es decir que una vez que el servidor responde el pedido del cliente, la conexión se corta y ninguno de los dos recuerda haber estado en contacto con el otro. Si el cliente debe efectuar otro pedido al mismo servidor, ambas partes deben comenzar desde el principio todos los pasos del protocolo.

Formato HTML

HTML (HyperText Markup Language) es un lenguaje para el formateo de documentos y especificación de links entre ellos. Define la sintaxis y ubicación de ciertas directivas que no son mostradas por el browser pero que le indican como visualizar el contenido del documento, incluyendo texto, imágenes y otros objetos. El lenguaje también indica como hacer un documento interactivo mediante la creación de links, que conectan a un documento con otros [MUS/96].

HTML es un lenguaje en el que se mezclan el texto de un documento con *tags*, que son los comandos que indican cómo mostrar dicho documento. La mayoría de estos tags definen un bloque de un documento HTML. El bloque comienza con el tag de comienzo y sus atributos y continúa hasta el tag de fin. El tag de fin es igual que el tag de comienzo pero precedido por el signo (/).

Por ejemplo: un ejemplo , tiene como tag de comienzo a y como tag de fin a y le indica al browser mostrar "un ejemplo" en itálicas.

Estructura de un documento HTML

Un documento HTML consiste de texto, que define el contenido del documento y tags, que definen la estructura y apariencia del documento. La estructura de un documento HTML consiste de un tag *<html>* y uno *</html>* encerrando a la cabecera y el cuerpo del documento.

```
<html>
<head>
<title> Ejemplo de un documento HTML </title>
</head>
<body>
Este documento muestra la estructura de un documento HTML
</body>
</html>
```

Cada documento tiene una cabecera (delimitada por el tag *head*) y un cuerpo (delimitado por el tag *body*). En la cabecera se indican el título y otros parámetros que el browser utiliza para mostrar el documento. En el cuerpo se pone el contenido del documento HTML, incluyendo el texto y los tags de control que le indican al browser cómo mostrar el texto. Los tags dentro del cuerpo también hacen referencia a archivos de gráficos o de sonido e indican los links del documento a otros documentos.

Tags

Cada tag consiste de un nombre de tag, a veces seguido de una lista de atributos del tag, todos ubicados entre los signos "<" y ">".

Algunos ejemplos de tags son:

```
<a href="http://www.ora.com/catalog.html">
<ul compact>
<input name=filename size=24 maxlength=80>
```

El primer ejemplo corresponde al tag <a> que indica un link especificado en el parámetro href.

El segundo ejemplo muestra el tag que formatea texto en una lista desordenada de ítems.

El último ejemplo muestra un tag con múltiples atributos.

La cabecera

La cabecera de un documento HTML describe las propiedades del documento, incluyendo su título, su ubicación dentro de la Web y su relación con otros documentos. La mayor parte de los datos contenidos en la cabecera no es mostrado a los usuarios.

Los principales tags incluidos en la cabecera son:

<head>	Define la cabecera del documento
<title>	Define el título del documento
<link> y <base>	Definen la ubicación del documento y su relación con otros documentos
<isindex>	Crea formularios de índices automáticamente
<meta>	Provee datos adicionales del documento

El cuerpo

El cuerpo del documento es donde se guarda el contenido del documento. Se encuentra delimitado por el tag *<body>* y algunos de los tags permitidos son:

<p>	Indica el comienzo de un párrafo
<h1>, <h2>...<h6>	Define seis niveles de encabezamientos
<i>	Itálicas

Creación de links

Uno de los tags más importantes dentro del cuerpo de un documento HTML es el tag *<a>*, que indica un link hacia otro lugar del documento o hacia otros documentos.

La referencia a otros documentos se realiza mediante el atributo *href*. Un ejemplo del tag <a> haciendo referencia a otro documento es:

El *<a href="http://www.serv.com/cata.html" catálogo se encuentra en preparación*, que aparece en un browser como:

El catálogo se encuentra en preparación.

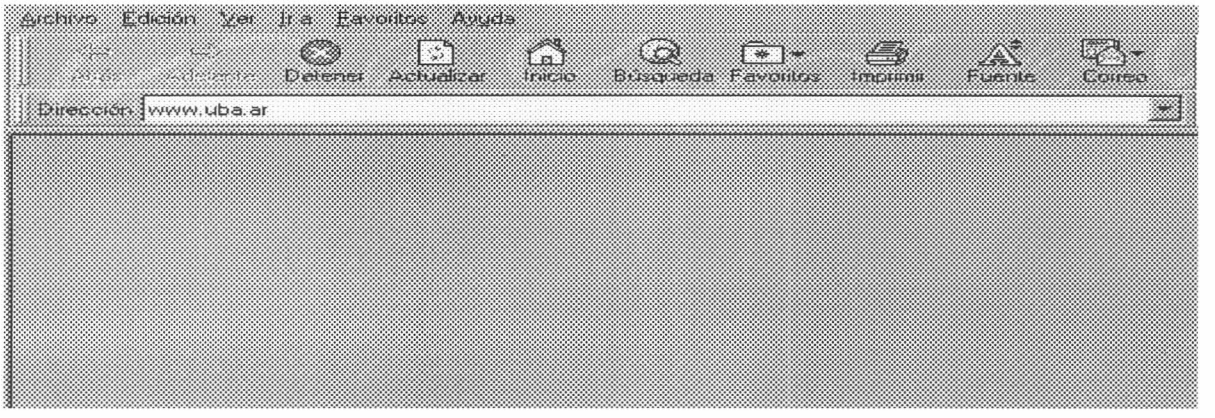
Clientes de la Web

Desde el punto de vista de los usuarios, la Web consiste en un gran conjunto de documentos. Cada documento puede contener punteros (links) a cualquier otro recurso de la Web. Los usuarios pueden seguir un link simplemente seleccionándolo con el mouse, obteniendo así el nuevo documento. Este proceso se puede repetir indefinidamente.

Un cliente de la Web es un programa que envía solicitudes a los servidores de la Web mediante el protocolo HTTP, y espera respuestas de estos. A nivel usuario se utilizan programas llamados browsers, que son clientes de la Web que realizan las siguientes tareas:

1) Le permite al usuario ingresar el URL del recurso que quiere obtener.

2) El browser se conecta al servidor indicado en el URL y le envía un mensaje como este:



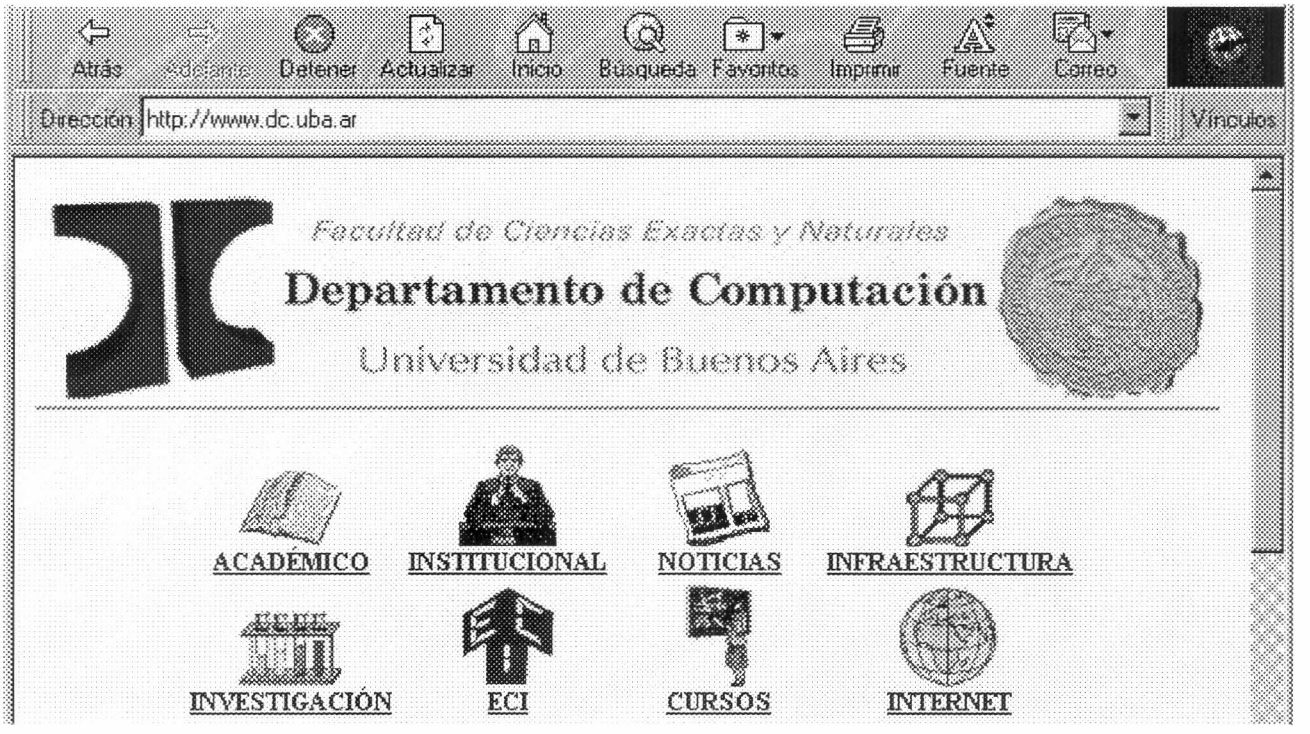

```
GET / HTTP 1.0
User-Agent: Browser xxx
Host: www.uba.ar
Accept: *
```

3) Recibe una respuesta del servidor, que comienza así:

```
HTTP 1.0 OK
Date: Sun, 7 Sep 1997 15:00
Server: Apache/1.1.1
Content-type: text/html
Content-length: 1239
Last-modified: Fri, 5 Sep 1997 19:15

<HTML>
<HEAD>
<TITLE>Departamento de Computaci&oacute;n - FCEyN - UBA</TITLE>
</HEAD>
<BODY text=#000000 bgcolor=#FFFFFF>

<TABLE width="100%" Border=0>
<TR>
<TD width="25%" Align=right><img src="/graficos/0ldc.gif" alt="[DC logo]"></TD>
<TD width="50%" Align=center><img src="/graficos/0ltexto.gif" alt="[FCEyN DC UBA]"></TD>

.......................
.......................
```

4) Si el documento tiene referencias a gráficos, el browser se los solicita al servidor, estableciendo una conexión para cada uno de los pedidos.

5) Muestra el documento, previamente interpretado, en la pantalla.



6) Ya teniendo una página en pantalla, el usuario puede solicitar otro documento, ingresando un nuevo URL o seleccionando uno de los links contenidos en el documento. En este último caso el browser obtiene el URL contenido en el tag <A> de HTML que identifica al link.

Algunos browsers usan el disco local para cachear las páginas obtenidas. Antes que una página sea buscada, se hace un chequeo para ver si ésta se encuentra en la caché local. Si es así, se chequea si la página está actualizada y en ese caso no hace falta cargarla nuevamente.

Servidores de la Web

Se suelen denominar servidores de la Web a las máquinas que contienen los documentos. Sin embargo el servidor no es una máquina sino un proceso que corre en una máquina. Cada servidor escucha en un port TCP (por defecto el 80) y se queda esperando conexiones de los clientes. Por cada nueva conexión se lanza un nuevo proceso que la atiende. Luego de establecerse una conexión el cliente envía un requerimiento y el servidor le devuelve una respuesta. Luego la conexión es liberada. El protocolo que define los requerimientos y respuestas legales es HTTP.

En el siguiente ejemplo se muestra un servidor con cuatro conexiones activas. Hay un proceso maestro que es el que recibe los pedidos de conexión y lanza un nuevo proceso por cada uno de ellos.

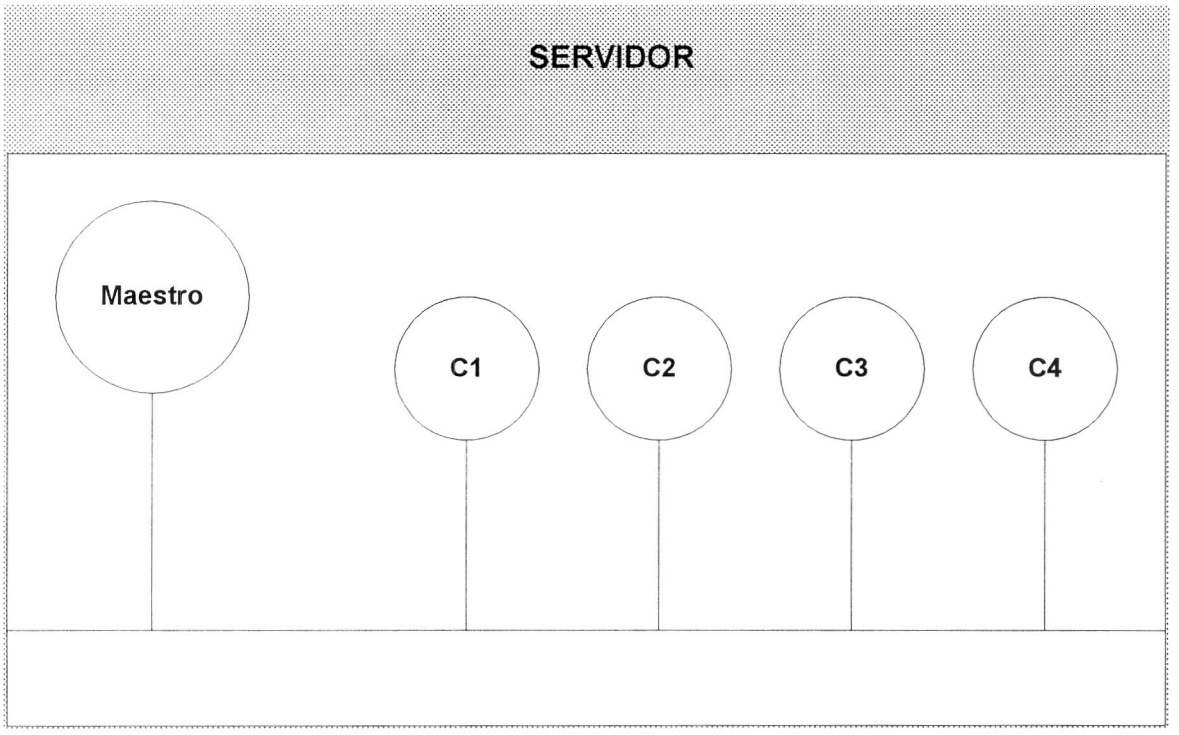


Figura 40: Varias conexiones en un servidor Web

CGI

CGI (Common Gateway Interface) es la parte del servidor Web que le permite a éste comunicarse con otros programas. Mediante CGI, el servidor, ante el requerimiento de un cliente, puede invocar a un programa pasándole argumentos (por ejemplo, el nombre del host del cliente) y enviarle al cliente la salida generada por dicho programa.

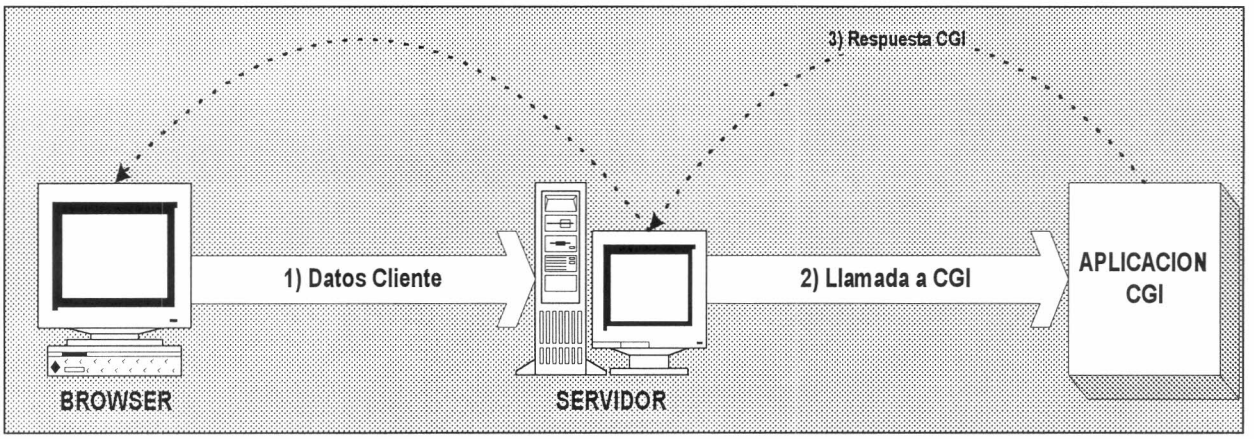


Figura 41: CGI

Apéndice C: Lista de robots

En este apéndice se describen brevemente algunos robots. La lista de robots presentada es un subconjunto de la presentada en [KOS/4].

WWWW

WWWW (World Wide Web Worm) [MCB/94] es una araña que intenta ubicar todos los recursos hipertextuales (documentos con formato HTML) posibles de la Web. Crea una base de datos con los recursos encontrados. Se indexan los títulos de cada documento y también sus links.

WWWW guarda en un archivo la lista de URL´s encontrados, que pueden estar en uno de los siguientes estados:

T	HTML	TITULO	
R	URL	HTML	TEXTO
C	HTML		

El estado T denota a un documento encontrado y abierto para su procesamiento, siendo su segundo argumento el URL correspondiente al documento. El último argumento corresponde al título del documento tomado del tag <TITLE>.

El estado R es utilizado para indicar un URL que es referenciado en un documento HTML que está siendo procesado. En este caso, el segundo argumento es el URL y el tercero es el URL del documento que contiene la referencia. El último argumento es el texto contenido en el URL. Por ejemplo, supongamos que el documento http://mach/home.html contiene la línea:

Haga click Foto de McBryan>

En el archivo quedará la siguiente entrada:

R http://mach/photo.gif http://mach/home.html **Foto de McBryan**

Por último, el estado C indica que un documento que había sido abierto previamente se ha completado. Esto indica que todas las referencias dentro del documento han sido procesadas.

Scooter

Scooter es la araña utilizada por el servidor de índices Alta Vista para la recolección de datos de la Web [DUM/96]. A pesar de su velocidad, Scooter es una araña bien educada, ya que respeta el estándar de exclusión de robots (más adelante lo veremos en detalle, pero básicamente se refiere a la posibilidad de los servidores de indicar a los robots qué documentos no deben obtener).

Además, introduce un tiempo de espera entre dos visitas sucesivas a un mismo sitio, con el fin de no saturarlo. Este tiempo de espera está en relación con el tiempo de respuesta observado en ese sitio; a más lentitud, más tiempo de espera. Sin embargo, Scooter no pasa este tiempo ocioso, ya que accede simultáneamente a miles de páginas en diferentes sitios.

Lycos

La araña utilizada por el servidor de índices Lycos [CHE/96] usa un método probabilístico para saltar de un documento a otro en el espacio de documentos de la Web, haciendo elecciones aleatorias de URL's. Este método evita la sobrecarga de los servidores y permite a Lycos dar preferencia, dentro de los URL's que va a visitar, a los que proveen más información.

Lycos obtiene documentos enteros pero sólo guarda la siguiente información de estos, con el objetivo de reducir la cantidad de información a ser almacenada.

- Título
- Cabeceras y subcabeceras

- Las 100 palabras más importantes
- Las primeras 20 líneas
- Tamaño en bytes
- Número de palabras

Las 100 palabras más importantes son obtenidas por un algoritmo que toma en consideración la ubicación y la frecuencia de las palabras, entre otros factores.

Los pasos básicos del algoritmo son:

1) Cuando se obtiene un URL, Lycos obtiene las referencias a nuevos URL's y los añade a una cola interna.

2) Para elegir el próximo URL a explorar, Lycos se basa en ciertas preferencias (heurísticas). Entre ellas se destacan el hecho de dar prioridad a los documentos más populares (los documentos que más son referenciados desde otros documentos) y a los URL's más cortos (por el hecho de que en general la 'home page' [1]es el URL más corto dentro de un servidor).

De lo anterior se deduce que Lycos usa una estrategia *best-first* [RUS/95], ya que elige el "mejor" URL para su próxima visita.

WebCrawler

La araña utilizada por WebCrawler [PIN/94] accede de a un documento a la vez para su posterior procesamiento local. Funciona utilizando solamente la infraestructura básica de la Web: la habilidad de los clientes de obtener documentos de los servidores a través de la primitiva GET del protocolo HTTP.

[1] Se llama ' home page' a la página inicial dentro de un servidor Web.

Luego de obtener un documento, WebCrawler realiza tres acciones. Marca el documento como recuperado, obtiene los links del documento e indexa el contenido de éste. Todos estos pasos insertan información en una base de datos.

El motor de búsqueda dirige las actividades. Es el responsable de decidir qué documentos explorar y de iniciar su transferencia.

WebCrawler descubre nuevos documentos comenzando con un conjunto conocido de documentos, examinando los links de estos y tomando uno de estos links para recuperar el documento asociado. Luego repite el proceso.

WebCrawler ve a la Web como a un grafo y lo explora mediante un algoritmo de búsqueda en grafos. Usa una variante del algoritmo 'ancho primero' (breadth-first) para decidir qué documentos visitar. Esto permite que se llegue a la máxima cantidad posible de servidores.

Entrando en detalle, cada vez que se encuentra un documento perteneciente a un nuevo servidor, dicho servidor es colocado en una lista de servidores a ser visitados. Antes de visitar a otros documentos, se visita a un documento de cada uno de los servidores de la lista.

Fish Search

Esta araña difiere sustancialmente de las anteriores ya que su objetivo es realizar una búsqueda y no generar un índice de la Web [BRA/94].

1) Se debe encontrar un buen punto de partida para la búsqueda. Las dos alternativas más comunes son usar como partida un documento conocido o partir de una búsqueda en un índice generado por un robot.

2) Los documentos son transferidos del servidor al cliente. Una vez en este último, el documento es revisado para obtener la información relevante.

3) Se obtienen los links que aparecen en los documentos.

La efectividad de la búsqueda en este entorno depende de la habilidad para encontrar documentos que brindan información relevante y poder dejar de lado a los que no.

La araña debe inspeccionar gran parte de la Web lo más rápido posible. El mejor resultado es un subconjunto de documentos que contienen la información deseada.

Simula a un cardumen buscando comida. Generalmente el cardumen se mueve en la dirección de la comida. El tiempo que viven los peces y la cantidad de hijos que producen depende de la cantidad de comida que encuentran en el camino. Cuando los peces se desvían del grupo, pueden sobrevivir si encuentran comida. Si no encuentran comida, mueren. Los peces que entran en aguas polucionadas también producen menos hijos y mueren rápidamente.

Haciendo una analogía entre el cardumen y el algoritmo, cada URL corresponde a un pez. Cuando un documento es recuperado se crea un conjunto de hijos dependiendo de la importancia del documento (siguiendo con la analogía, un documento importante implica que hay comida) y de la cantidad de links que contiene.

Cada vez que se encuentra un documento relevante, el pez y sus hijos se hacen más fuertes. Si el documento no contiene información importante, el pez se hace más débil y se generan menos hijos.

Luego de seguir una serie de links que no encuentran documentos relevantes, se suspende la investigación en esa dirección (el pez muere).

Cuando la recuperación de un documento se hace lenta se generan pocos hijos para no ir demasiado en esa dirección.

La implementación actual del algoritmo es secuencial. Los peces son simulados por una lista de URL's. El número promedio de hijos producidos cada vez recibe el nombre de 'ancho'. El número de pasos permitidos sin encontrar documentos relevantes

se llama 'profundidad'. Cada vez que se encuentra un documento relevante se agregan al comienzo de la lista una selección de los links del documento. Esto lleva a un tipo de recorrido 'profundidad-primero'. Cuando se encuentra un documento poco relevante, un subconjunto de sus links es guardado en la lista al final de todos los links seleccionados de documentos relevantes. Se da prioridad a los links que apuntan a diferentes documentos para lograr una mayor atención sobre la Web entera.

Cuando se agrega un URL el algoritmo chequea si éste no se encuentra en la lista. Si ya se encuentra, no es insertado nuevamente, pero puede ser desplazado si es que el documento en el que se encuentra contiene información relevante.

Jumpstation

User-Agent: Jumpstation.

Descripción: Su propósito es generar una base de datos de los recursos de la Web. Se respeta el estándar para la exclusión de robots. Es un conjunto de programas escritos en Perl.

RBSE

User-Agent: RBSE-Spider/0,1 a

Descripción: Su propósito es generar una base de datos de los recursos de la Web y generar estadísticas. Se respeta el estándar para la exclusión de robots. Es un programa escrito en C.

NorthStar

User-Agent: NorthStar

Descripción: Se utiliza con fines estadísticos

W4 (World Wide Web Wanderer)

User-Agent: WWWWanderer v3.0 by Mathew Gray

Descripción: Su objetivo es medir el crecimiento de la Web.

MOMspider

User-Agent: MOMspider/1.00 libwww-perl/0.40

Descripción: Su propósito es validar links y generar estadísticas. Soporta el estándar de exclusión de robots.

WM32

User-Agent: W3M2/x.xx

Descripción: Su propósito es validar links, obtener estadísticas y generar una base de datos de recursos de la Web. Soporta el estándar de exclusión de robots.

WebWalk

User-Agent: webwalk

Descripción: Su propósito es validar links, obtener estadísticas y generar una base de datos de recursos de la Web. Soporta el estándar de exclusión de robots.

InfoSeek

User-Agent: InfoSeek Robot 1.0

Descripción: Su propósito es generar una base de datos de recursos de la Web. Soporta el estándar de exclusión de robots. Utiliza un algoritmo que determina las páginas más populares. Es muy rápido pero nunca hace más de un pedido a la vez a un mismo servidor.

Apéndice D: Manual del Usuario

Requerimientos

Tanto el servidor EROS como el robot que lo utiliza se implementaron y probaron bajo el sistema operativo LINUX versión 1.2.8. Se programó en lenguaje C y se utilizó como mecanismo de comunicación la interface BSD (Berkeley Socket Interface). Se tuvo cuidado de utilizar funciones compatibles con UNIX.

Para compilar el servidor EROS hace falta contar con el analizador lexicográfico Flex (nosotros trabajamos con la versión 2.4). Se necesitan los utilitarios Gzip y Tar para poder instalar tanto el servidor como el robot.

Instalación

El diskette entregado consta de dos archivos. El archivo eros.tar.gz contiene los fuentes del servidor EROS mientras que robot.tar.gz contiene los fuentes del robot.

Para instalar el servidor EROS se deben seguir los siguientes pasos:

1) cd /
2) gzip –d eros.tar.gz
3) tar –xvf eros.tar

Los fuentes del servidor EROS quedan ubicados en el directorio /eros

4) cd /eros
5) Compilar el servidor EROS mediante el script c_eros

Para instalar el robot se deben seguir los siguientes pasos:

1) cd /
2) gzip –d robot.tar.gz
3) tar –xvf robot.tar

Los fuentes del robot quedan ubicados en el directorio /robot

4) cd /robot
5) Compilar el robot mediante el script c_robot

Funcionamiento

Servidor EROS

Para iniciciar el servidor EROS se debe ejecutar el script i_servidor, que básicamente se encarga de iniciar el proceso servidor, con el siguiente comando:

Servidor port_servidor port_robot raíz_web&

Donde:

Port_servidor es el port donde atiende el servidor EROS los pedidos de los robots.
Port_robot es el port donde esperan los robots las respuestas de los servidores EROS.
Raíz_web es el directorio raíz del servidor Web.

Ejemplo:

servidor 2100 2200 /usr/local/www&

Robot

Para iniciar el robot se deben realizar dos tareas.

1) Insertar los url´s iniciales a partir de los cuales el robot empezará a recorrer los servidores EROS. Esta tarea se realiza mediante el proceso insertar_link URL. Por ejemplo, insertar_link www.uba.ar/index.html
2) Iniciar el robot mediante el script i_robot, que principalmente inicia el proceso destUDP con el siguiente comando:

   destUDP u#dir_retorno port_servidor port_robot tipo_respuesta palabras_claves palabras_iniciales headers &

   Donde:

   dir_retorno: dirección IP de la máquina en dónde se reciben las respuestas.
   port_servidor: port en donde escuchan los servidores EROS.
   port_robot: port en donde se esperan las respuestas de los servidores EROS.
   tipo_respuesta: L (links), A (links + palabras claves), C (palabras claves), T (todo el documento)
   palabras_claves: cantidad de palabras claves que debe incluir el servidor.
   palabras_iniciales: cantidad de palabras iniciales que debe incluir el servidor.
   headers: los códigos de los headers que debe incluir el servidor. Los códigos para los headers son: 0 (títulos), 1 (negritas), 2 (itálicas), 3 (tag H1), 4 (tag H2), 5 (tag H3), 10 (tag META) y 11 (links).
   fecha: la fecha de actualización

   Ejemplo:

   destUDP u#robot.com 2100 2200 A 20 50 001011 &

Los datos que se reciben en la dirección indicada (en el ejemplo, robot.com) son grabados en los directorios links (cuando se recibe un mensaje que contiene links) y texto (cuando se recibe un mensaje que contiene el resumen de un documento html).

BIBLIOGRAFIA

[BOW/94] Bowman M., Danzig P., Hardy D. Manber., Schwartz M., "Harvest: Scalable, Customizable Discovery and Access System". Technical Report CU-CS-732-94, Universidad de Colorado, Julio 1994.

[BRA/94] De Bra, Post, "Information Retrieval in the World Wide Web: Making Client-based searching feasable". Proceedings on the First International World Wide Web Conference, Geneva, Suiza, Mayo 1994.

[CHE/96] Cheong F., "Internet Agents: Spiders, Wanderers, Brokers, and Bots", New Riders Publishing, 1996, U.S.A.

[COM/91a] Comer D., "Internetworking with TCP/IP Volume I; Principles, Protocols and Architecture", Prentice Hall, 1991, U.S.A.

[COM/91b] Comer D., "Internetworking with TCP/IP Volume II; Design, Implementation and Internals", Prentice Hall, 1991, U.S.A.

[COM/93] Comer D., "Internetworking with TCP/IP Volume III; Client - Server Programming and Applications (BSD Socket Version)", Prentice Hall, 1993, U.S.A.

[COM/95] Comer D. "El libro de Internet", Prentice Hall, 1995, U.S.A.

[DUM/96] Du Mortier G., "Internet al Máximo", MP Ediciones S.A., 1996, Argentina.

[EIC/94] Eichmann D., "The RBSE Spider - Balancing Effective Search against Web Load". Proceedings of the First International World Wide Web Conference, Geneva, Suiza, Mayo de 1994.

[FIE/94] Fielding R., "Maintaining Distributes Hypertext Infoestructures: Welcome to the MOMspider's Web", Proceedings of the First International World Wide Web Conference, Geneva, Suiza, Mayo 1994.

[FLE/1] Fletcher, J., "Internet Robots - Structure from Anarchy?", http://lorne.stir.ac.uk:80/~jfl/papers/signidr.html.

[GRA/96] Gray M. "Web Growth Summary", 1996, http://www.mit.edu:8001/people/mkgray/net/web-growth-summary.html

[GRA/97] Gray J. S., "Interprocess Communications in Unix", Prentice Hall, 1997, U.S.A.

[HAR/96] Hardy D. R., Schwartz M. F., Wessels D., "Harvest User´s Manual", Version 1.4 patchlevel 2, 1996.

[KOS/1] Koster M., "Robots in the Web: threat or treat?"

http: //info.webcrawler.com/ mak/projects/robots/threat-or-treat.html

[KOS/2] Koster M, "A proposed standard for robot exclusion", http://web.nexor.co.uk/mak/doc/robots/norobots.html

[KOS/3] Koster M., "Guidelines for robots writers", http://web.nexor.co.uk/mak/doc/robots/guidelines.html

[KOS/94] Koster M., "ALIWEB - Archie-Like Indexing in the Web", Proceedings of the First International World Wide Web Conference, Geneva, Suiza, Mayo 1994.

[KOS/4] Koster M., "List of robots", http://web.nexor.co.uk/mak/doc/robots/active.html

[KOS/5] Kosmynin Arkadi, "Distribution in Internet Resource Discovery", Australia, http://www.dstc.edu.au/aw3tc/papers/kosmynin

[LAL/96] La Liberte D., "Indexing the Web", 1996, http://union.ncsa.uiuc.edu/hypernews/get/www/indexing.html

[LEE/94a] Berners-Lee T., Cailliau R., Loutonen A., Nielsen F. Y Secret A., "The World Wide Web", Communications of the ACM, v37, n. 8, Agosto 1994, p. 76-82.

[LEE/94b] Berners-Lee T., Fielding R., Nielsen F., "Hypertext Transfer Protocol - HTTP/1.0", Internetd Draft, Diciembre de 1994.

[LUN/96] Lundberg S., Ardö A., Brummer A. y Koch T., "The European Web Index: An Internet Search Service for the European Higher Education, Reseach and Development Communities", Universidad de Lund, Suecia, 1996.

[MCB/94] McBryan A., "GENVL and WWWW: Tools for Taming the Web", Proceedings of the First International World Wide Web Conference, Geneva, Suiza, Mayo 1994.

[MOG/1] Mogul J., "WRL Research Report 95/4 – The Case for Persistent Connection HTTP", Mayo de 1995, U.S.A.

[MOG/2] Mogul J., "Improving HTTP Latency"

[MUS/96] Musciano C., Kennedy B., "HTML: The Definitive Guide", O'Reilly & Associates Inc., 1996, U.S.A.

[PER/96] Perret S., Duda A., "Mobile Assistant Programming for Efficient Information Access on the WWW", Fifth International World Wide Web Conference, París, Francia, 6-10 de Mayo 1996.

[PIN/94] Pinkerton B., "Finding What People Want: Experiences with the WebCrawler", Proceedings of the Second International World Wide Web Conference, Chicago, E.E.U.U., Octubre 1994.

[POS/80] Postel J., 1980, "User Datagram Protocol", RFC768, 3 páginas.

[POS/81] Postel J., 1981, "Transmission Control Protocol", RFC793, 85 páginas.

[POS/85] Postel J., Reynolds J. K., 1985, "File Transfer Protocol (FTP)", RFC959, 69 páginas.

[RUS/95] Russell S., Norvig P., "Artificial Intelligence, a modern approach", Capítulos 3 y 4, Prentice Hall, 1995, U.S.A.

[STE/94] Stevens W. R., "TCP/IP Illustrated, Volume 1 – The Protocols", Addison-Wesley Publishing Company, 1994, U.S.A.

[SPE/94] Spetka S., "The TkWWW Robot: Beyond Browsing", Proceedings of the Second International World Wide Web Conference, Chicago, E.E.U.U., Octubre 1994.

[SPE/1] Spero S., "Analisys of HTTP Performance problems", HTTP Working Group, World Wide Web Organization, Noviembre de 1994.

[TAN/96] Tanenbaum A., "Computer Networks", 3rd Edition, Prentice Hall,1996,U.S.A.

[WEL/96] Wells D., "Searching and Indexing", 1996, http://www.objs.com/survey/crawl.htm.