

PARTE I  
**SERVICIOS DE RED**



## 1. TECNOLOGÍAS WEB BÁSICAS

En sus orígenes, la tecnología Web, basada en el protocolo HTTP para la transmisión de hipertexto sobre protocolos de comunicaciones TCP/IP, sirvió para proporcionar una nueva visión de Internet. Si antes, con servicios como FTP, la Red se concebía como un conjunto de nodos servidores cuya topología y estructura física había que conocer, ahora se logra un nivel de abstracción mayor: lo que importa es la información —*los contenidos* en términos del WWW— en lugar de su ubicación, de las características de los servidores en los que está ubicada o de los dispositivos empleados para acceder a la misma: computador personal, PDA, WebTV, teléfono móvil, etc.

Estas características de independencia entre clientes y servidores, unida al auge de la generación de contenidos en Internet y la enorme heterogeneidad reinante en el entorno de las TIC, facilitan que prospere rápidamente este nuevo modelo cliente/servidor.

Sin embargo, debido en parte a su paulatina participación en el mundo de los negocios y a la necesidad, por tanto, de generación de contenidos dinámicos, cada vez son más las tecnologías que en los últimos tiempos han venido a acompañar a los tradicionales *servidores Web* y, porqué no, a *engordar* nuestros *navegadores Web*. Está claro que se necesita una mejor gestión de recursos y una mayor organización de las tecnologías implicadas.

Atrás quedaron los tiempos en que la Web era un mero escaparate de consulta de información estática, dónde las empresas presentaban su catálogo de productos o su oferta de servicios. En la actualidad, se tiende progresivamente a hacer efectivos los diferentes servicios que ya se prestan por otros medios, en esa red universal que es Internet. Para ello, las aplicaciones convencionales cliente/servidor para redes locales o corporativas, escritas pensando en un entorno determinado, deberán ser

adaptadas a las nuevas características e idiosincrasia que imponen las tecnologías de Internet en general y de la Web en particular.

Durante los próximos cuatro capítulos revisaremos su evolución: partiendo en este capítulo del concepto de HTTP básico y de aplicación Web, en los capítulos 2 y 3 analizaremos las tecnologías más importantes que se han ido incorporando tanto en la parte del cliente —DTML, *ActiveX*, *applets java*, *plug-ins*— como en la del servidor —*páginas activas*, componentes en el servidor, contenedores de componentes; finalmente, en el capítulo 4 veremos cómo, de alguna manera, tanto por la complejidad que están adquiriendo las actuales aplicaciones como por la sobrecarga que todas estas tecnologías infligen sobre los servidores Web convencionales, en la actualidad, la tendencia es volver a aligerar a nuestro servidor web, dejándolo que se ocupe de aquello para lo que fue concebido —la gestión del nivel de presentación o interfaz con el usuario final— y organizar un sistema paralelo complementario capaz de dar soporte adecuado a las aplicaciones.

## 1.1. MODELO CLIENTE-SERVIDOR

El Modelo Cliente-Servidor es uno de los más empleados por un equipo o aplicación (cliente) para acceder a recursos ubicados en otro equipo (servidor) u ofrecido por otra aplicación (servicio).

En la figura 1.1 se muestra un diagrama en el que se destacan los principales elementos que intervienen en un entorno preparado para emplear el modelo cliente-servidor a través de Internet.

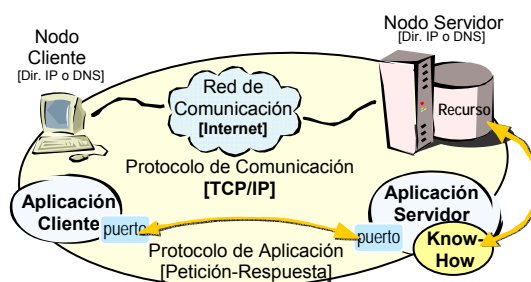


Figura 1.1. Principales elementos del modelo Cliente/Servidor sobre TCP/IP.

El primer elemento a destacar es el *nodo cliente*. El nodo cliente está formado por el equipo computacional —ya sea un PC convencional, un

portátil, una agenda electrónica o un teléfono móvil— con su respectivo sistema operativo y con capacidad para conectarse a una red de computadores. Cada nodo cliente debe disponer de, al menos, una aplicación que denominaremos *aplicación cliente*. Esta aplicación es la responsable de solicitar el recurso o servicio deseado y, en caso de ser interactiva, actuar como interfaz con el usuario final. En la práctica, tanto el nodo cliente como la aplicación cliente se suelen denominar, sencillamente, clientes.

Al otro lado del escenario nos encontramos el *nodo servidor* o *servidor*. En este caso, se trata del equipo —la combinación de hardware más sistema operativo— que posee el *recurso* hardware o software objeto del servicio. El servidor deberá estar conectado de alguna forma a la misma red que el cliente. Sobre él se ejecutará la *aplicación servidora* o *servicio* capaz de atender las solicitudes del cliente y mediante su *know-how* —su *saber cómo*— para acceder y gestionar dicho recurso.

Entre ambos nodos encontramos la *red de comunicación*. Sea cual sea su tecnología física, deberá emplear como protocolo de comunicación (niveles de red y transporte), la *pila TCP/IP*, es decir, el conjunto de protocolos empleado por *Internet*.

En este modelo cliente-servidor resulta imprescindible que ambas partes tengan alguna forma de referenciarse. Un cliente debe poder invocar a un servidor, y este servidor tendrá que ser capaz de devolverle su solicitud. Puesto que la red (lógica) en la que nos basamos es Internet, el mecanismo de identificación que cada nodo conectado a la misma debe poseer —ya sea cliente o servidor—, es lo que denominamos una *dirección IP*. Sin embargo, una dirección IP está compuesta por un número de cuatro bytes que resulta muy incómodo de recordar y manipular por los usuarios humanos. Por esta razón, lo más común es emplear un sistema de traducción de direcciones IP a un sistema de nombres jerárquicos denominando DNS (*Domain Name Server*), en el que el nombre del nodo se expresa mediante una combinación de *nombre* más el *dominio* al que pertenece. La dirección así expresada se denomina *nombre DNS*.

Aunque una dirección IP o un nombre DNS permiten identificar el nodo de red de manera unívoca, todavía falta identificar la aplicación concreta, de entre todas las que se están ejecutando sobre cada nodo, con la que nos queremos comunicar. En este caso, el protocolo TCP/IP proporciona como mecanismo un *número entero* diferente a cada aplicación en ejecución que desee acceder a la red. Este número se denomina *puerto*. Por lo tanto, el par *direcciónIP:puerto* o *nombreDNS:puerto* será el que realmente identifique cliente y servicio.

Finalmente, aunque los nodos de red cliente y servidor se comunican mediante el protocolo de comunicaciones TCP/IP que les permite entenderse a nivel de red y de transporte de datos, las aplicaciones cliente y servidor también deben poseer un mecanismo que les permita hablar entre ellas, es lo que denominamos *protocolo de aplicación*.

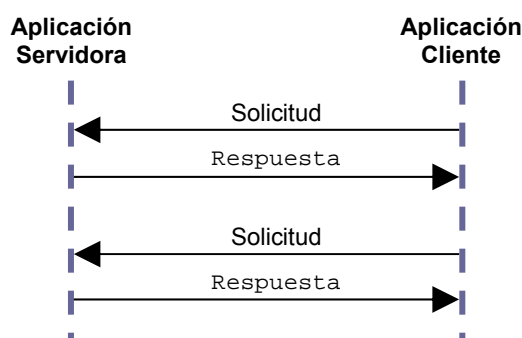


Figura 1.2. Dos secuencias Solicitud-Respuesta genéricas.

En el caso de este tipo de aplicaciones, este protocolo suele ser del tipo *petición-respuesta* (ver figura 1.2). Son protocolos en los que la *respuesta* del servidor a la *solicitud* del cliente se entiende como el justificante de haber sido recibida. Esto permite disminuir el tráfico de red provocado por el envío de justificantes y que, en muchas ocasiones, son el motivo del colapso de los sistemas de comunicación. En la mayor parte de los casos, el protocolo de aplicación es el que proporciona su nombre al servicio: FTP, HTTP, *Telnet*, etc.

## 1.2. SERVICIO HTTP

El servicio HTTP es un servicio basado en el modelo cliente-servidor sobre Internet, donde el cliente es un *navegador Web* y el servidor es un *servidor Web* (ver figura 1.3), utilizando ambos para su entendimiento el *protocolo de aplicación HTTP* (*HyperText Transfer Protocol*). En este caso el servidor Web se configura por defecto para escuchar solicitudes en el *puerto 80*, de forma que cualquier cliente pueda encontrarlo fácilmente. El recurso que se ofrece lo constituyen documentos de texto denominados *páginas HTML* (*HyperText Markup Language*), por ser éste el lenguaje que se emplea para su codificación (ver figura 1.3).

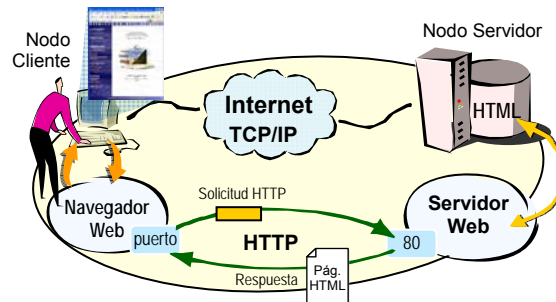


Figura 1.3. Elementos del servicio HTTP.

En la figura 1.4 se puede apreciar un ejemplo de solicitud de página HTML utilizando el protocolo HTTP. La respuesta será la propia página solicitada.

| Método | Recurso     | Protocolo | Cabecera |
|--------|-------------|-----------|----------|
| GET    | /index.html | HTTP/1.1  |          |

Figura 1.4. Ejemplo de una *Solicitud* HTTP.

### 1.2.1. Protocolo de aplicación HTTP

Se trata de un protocolo de aplicación, basado en texto y del tipo solicitud-respuesta. El formato de una solicitud HTTP responde a la siguiente sintaxis:

```
<Método HTTP> <URI> <Protocolo>
<Cabecera>
<Línea en blanco>
[<Cuerpo>]
```

Donde:

- El *Método HTTP* indica la acción que se solicita al servidor. Alguno de los principales métodos HTTP son los siguientes:
  - HEAD Solicita la cabecera del recurso referenciado.
  - GET Solicita un recurso mediante una URI.
  - POST Solicita un recurso y pasa información adicional en el *cuerpo* de la solicitud.

PUT Solicita que el servidor almacene la información enviada con el nombre indicado en la URI

- La *URI* (*Uniform Resource Identifier*) es la forma normalizada de referenciar un recurso. La diferencia con una *URL* (*Universal Resource Locator*) es que en el último caso, además de la URI, también se incluye el tipo de servicio al que se quiere acceder, la dirección del nodo servidor y el puerto en el que el servicio espera las solicitudes.
- El *Protocolo* indica el nombre y versión del protocolo que espera el cliente.
- La *Cabecera* de la solicitud sirve para indicar diferentes parámetros que modifican la forma en la que se realiza la solicitud o la respuesta. La cabecera finaliza con una línea en blanco. El formato de cada parámetro es el siguiente:  
`<etiqueta>: <valor>\r\n`
- El *Cuerpo* de la solicitud es opcional y permite enviar información adicional junto a la solicitud.

En la figura 1.5 se presenta un sencillo ejemplo en el que se solicita al servidor el archivo `miAplicacion.cgi` situado en la carpeta `/cgi`, también se le indica mediante la cabecera algunas condiciones —como que se acepta cualquier tipo de datos, que no cierre la sesión inmediatamente y que nuestro navegador es del tipo genérico—. Además, se emplea el cuerpo de la solicitud para pasar información adicional (que se detallará en los próximos apartados) al servidor Web.



|           |   |
|-----------|---|
| Solicitud | POST /cgi/miAplicacion.cgi HTTP/1.0   |
| Cabecera  | Accept: */*<br>Connection: Keep-Alive<br>User-Agent: Generic<br>[línea en blanco] |
| Cuerpo    | Nombre=Paco&eMail=pmacia@dtic.ua.es   |

Figura 1.5. Ejemplo de solicitud HTTP.

Así mismo, la Respuesta también sigue un formato establecido que se puede sintetizar en la siguiente sintaxis:

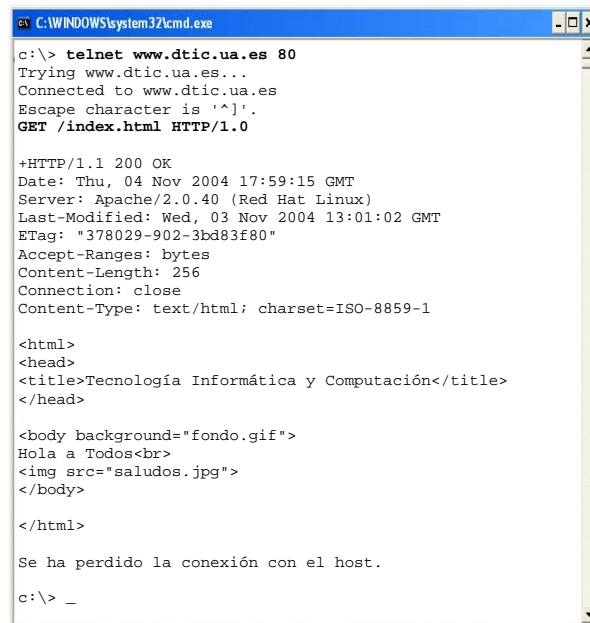
```
<Línea de estado>
<Cabecera>
<Línea en blanco>
[<Cuerpo>]
```

Donde:

- La *Línea de estado* comienza con un símbolo más (+) si todo ha ido bien o con un símbolo menos (-) si se ha producido algún incidente. A continuación se especifica el protocolo que emplea el servidor (por ejemplo: HTTP/1.0), el código de estado informando sobre el desarrollo, negativo o positivo, de la solicitud y, finalmente, una descripción textual del mismo: si ha sido exitosa, si se ha producido un error o, sencillamente, con información de carácter más general.  
+|-protocolo código texto
- La *Cabecera* de la respuesta sirve para indicar diferente información sobre el servidor y sobre el objeto solicitado: la fecha, el nombre del servidor, el tipo MIME del objeto, etc. El formato de cada parámetro es el siguiente:  
<etiqueta>: <valor>
- El *Cuerpo* es la parte más importante de la respuesta, pues está formado por el contenido del recurso web solicitado, generalmente

una página en formato HTML o un archivo en formato MIME (ambos se describirán más adelante).

En la figura 1.6 se presenta un ejemplo de conexión remota mediante un sencillo cliente *telnet*. En primer lugar se realiza una conexión indicando el nombre DNS del servidor y el puerto en el que se encuentra escuchando (puerto 80). Una vez establecida la conexión, el servidor Web queda a la espera de una solicitud que siga el protocolo HTTP. En el caso del ejemplo se trata de una solicitud para recuperar el archivo `index.html` que se debe encontrar en el directorio raíz del servidor Web. El servidor Web responde indicando con un símbolo más (+) que todo ha ido bien, informa sobre el protocolo con el que trabaja (HTTP/1.1), el código de estado (200) y una breve descripción que indica que todo ha ido bien (OK). A continuación envía la cabecera de la respuesta con algunos datos sobre la conexión y la información a transmitir (fecha, servidor, última modificación del archivo, una marca, el tamaño de los datos y del archivo, el estado de la conexión y el tipo MIME del archivo). Finalmente, tras una línea en blanco que señala el final de la cabecera, utiliza el cuerpo de la respuesta para transmitir el contenido del archivo de texto, en formato HTML, que se le había solicitado. Una vez enviada esta información y puesto que HTTP es un protocolo de tipo petición-respuesta, el servidor da por finalizada la conexión, es decir, si deseamos un nuevo recurso de este servidor, tendremos que realizar una nueva conexión siguiendo nuevamente todos los pasos anteriormente detallados. Por supuesto, un cliente *telnet* no sabe interpretar las etiquetas de formato que ha recibido, por lo que mostrará el contenido del archivo de forma literal.



```

c:\> telnet www.dtic.ua.es 80
Trying www.dtic.ua.es...
Connected to www.dtic.ua.es
Escape character is '^]'.
GET /index.html HTTP/1.0

+HTTP/1.1 200 OK
Date: Thu, 04 Nov 2004 17:59:15 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Wed, 03 Nov 2004 13:01:02 GMT
ETag: "378029-902-3bd83f80"
Accept-Ranges: bytes
Content-Length: 256
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<html>
<head>
<title>Tecnología Informática y Computación</title>
</head>

<body background="fondo.gif">
Hola a Todos<br>

</body>

</html>

Se ha perdido la conexión con el host.

c:\> _

```

Figura 1.6. Ejemplo de una sesión remota con una solicitud-respuesta HTTP.

En principio, un servidor Web *elemental* sólo está preparado para servir información estática, normalmente en forma de páginas HTML que el navegador Web del cliente se encargará de interpretar y representar, generalmente, de forma gráfica en su monitor. En el ejemplo de la figura 1.6 se había solicitado una sencilla página HTML que implementa el típico caso de ejemplo «Saludos». El contenido del archivo `/index.html` con dicho contenido podría ser el mostrado en la figura 1.7.

```

<HTML>
  <HEAD>
    <TITLE>Saludos</TITLE>
  </HEAD>

  <BODY background="fondo.gif">
    Hola a todos <BR>
    <IMG src="saludos.jpg">
  </BODY>
</HTML>

```

Figura 1.7. Listado del archivo `index.html` con un sencillo documento HTML.

Un documento HTML, tal y como sus siglas indican, está formado por los siguientes elementos:

- Un documento de texto.
- Etiquetas de marcado o de formato:  
`<etiqueta [arg1 arg2 ...]> ... [</etiqueta>]`
- Referencias a otros recursos:  
`<A HREF="URL"> texto descriptivo </A>`

Aunque las páginas o archivos en formato HTML son los recursos más básicos de un servicio Web, pueden requerir otros recursos adicionales para que el navegador Web pueda mostrar todo su contenido (en el ejemplo de la figura 1.7, tras el mensaje «*Hola a todos*», se presentará una imagen en formato *JPEG* (*Joint Photographic Experts Group*), ubicada en el servidor, con el nombre de `saludos.jpg`. Por supuesto, el navegador Web tendrá que recuperar este archivo mediante una nueva solicitud al servidor Web. El problema es que estos recursos suelen ser archivos gráficos, de vídeo o de sonido, que no se encuentran en formato de texto, que es para lo único que está preparado el protocolo de texto HTTP. Para subsanar esta deficiencia, cada archivo binario deberá ser convertido previamente a un archivo de texto mediante un mecanismo de representación externa de datos denominado *MIME* (*Multipurpose Internet Mail Extension*). La figura 1.8 muestra el contenido de un archivo MIME correspondiente a la conversión a texto de una imagen binaria en formato JPEG. Este mecanismo fue originalmente concebido para poder adjuntar archivos en formato binario a los mensajes de correo electrónico mediante el servicio SMTE (que se estudiará en el capítulo 6).

### 1.2.2. Arquitectura HTTP

Podemos considerar que la arquitectura técnica, física, de un sistema distribuido para la prestación del servicio HTTP tiene como fundamento una red de comunicaciones. Sobre ésta, (a la derecha en la figura 1.9), encontramos la estructura del cliente. Subiendo por las diferentes capas de dicha estructura hallamos el hardware del nodo cliente en el que destaca la tarjeta adaptadora de red (NIC) que le permite acceder al canal de comunicación. A continuación encontramos el sistema operativo que gestiona el nodo y en el que se ubica la pila TCP/IP que implementa los protocolos de comunicaciones. Finalmente llegamos a la aplicación cliente que, en este caso, es un navegador Web. Como ya se ha podido estudiar en el apartado anterior, el navegador Web tendrá capacidad para comunicarse con el servidor mediante el protocolo HTTP y para interpretar los archivos



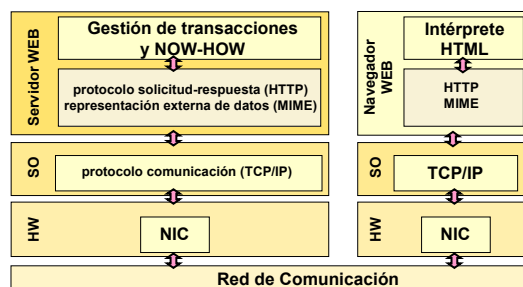


Figura 1.9. Arquitectura técnica física de una aplicación Web sobre Internet.

### 1.3. INTERFAZ CGI Y APLICACIONES WEB

Básicamente, una *aplicación Web* es cualquier aplicación basada en la arquitectura cliente-servidor, donde el cliente es un *navegador Web* y el servidor es un *servidor Web* (ver figura 1.3), utilizando ambos para su entendimiento el *protocolo de aplicación HTTP (HyperText Transfer Protocol)*. Sin embargo, con las características del servidor Web que hemos estudiado hasta el momento, el servicio HTTP tan solo es capaz de proporcionar una serie de páginas estáticas, previamente definidas, que difícilmente servirían para alcanzar un objetivo tan ambicioso.

Un requisito mínimo para poder construir una aplicación a partir del servicio HTTP es contar con la capacidad de generar páginas HTML de forma dinámica en función de determinadas condiciones establecidas por el cliente. Por suerte, el servicio HTTP básico cuenta con una característica adicional que permite hacer esto: un servidor Web puede invocar una aplicación externa a petición del cliente, pasarle una serie de parámetros que éste haya indicado y, tras la ejecución de dicha aplicación, recoger su salida para, seguidamente, retransmitírsela al cliente.

Estas aplicaciones se ejecutan en el nodo servidor y pueden estar desarrolladas en cualquier lenguaje de programación, compilado o interpretado, soportado por dicho servidor. La única restricción que se les impone es que implementen un mecanismo que les permita comunicarse con el servicio Web; concretamente: acceder a los parámetros de ejecución —en caso de que éstos existan— determinados por el cliente y que su salida —en formato textual y preferiblemente en formato HTML— pueda ser recuperada por el servidor Web para pasársela al cliente que invocó la aplicación. Este mecanismo está perfectamente definido para que ambas partes la conozcan de antemano y se denomina *CGI (Common Gateway*

*Interface*); razón por la cual, este tipo de aplicaciones se denominan también *aplicaciones CGI*.

En la figura 1.10 se presenta un sencillo ejemplo de solicitud HTTP GET en la que un cliente solicitaría al servidor la ejecución de la aplicación `busca.php` ubicada en la carpeta `/cgi-bin` y le pasa como argumento un parámetro identificado como `clave`, cuyo valor será `pdi`.

| Método | URI                          | Cabecera |
|--------|------------------------------|----------|
| GET    | /cgi-bin/busca.php?clave=pdi |          |

Figura 1.10. Formato de invocación de una aplicación CGI.

Otro ejemplo de solicitud de ejecución de una aplicación CGI, pero esta vez empleando el método POST, es el mostrado en la figura 1.11. En él puede observarse cómo los parámetros y sus valores, se transfieren ahora en el cuerpo de la solicitud.

|           |  |
|-----------|--|
| Solicitud | POST /cgi/miAplicacion.cgi HTTP/1.0  |
| Cabecera  | Accept: /*/*<br>Connection: Keep-Alive<br>User-Agent: Generic<br>[línea en blanco] |
| Cuerpo    | Nombre=Paco&eMail=pmacia@dtic.ua.es  |

Figura 1.11. Ejemplo de solicitud HTTP utilizando el método POST.

En la figura 1.12 se muestra un sencillo escenario en el que un cliente invoca la ejecución de una aplicación CGI. Su funcionamiento, siguiendo el procedimiento que define la interfaz CGI es el siguiente:

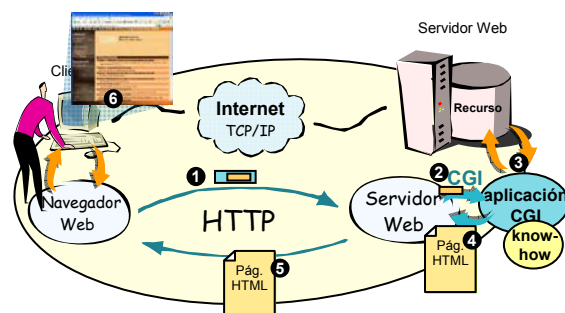


Figura 1.12. Escenario de desarrollo de un servicio HTTP mediante CGI.

- 1 Cliente o navegador Web a Servidor Web. El usuario, a través de su navegador Web solicita la ejecución de una aplicación CGI al servidor. Esta solicitud puede realizarse mediante el método GET o el método POST. En el caso de una solicitud GET, los parámetros y sus valores se incluyen en la propia URI (ver figura 1.10). Con el método POST, estos mismos datos se incluyen en el cuerpo de la solicitud (ver figura 1.11).
- 2 Servidor Web a Aplicación invocada. El Servidor Web copia determinados parámetros en variables de entorno (ver figura 1.13) —en Win-CGI éstas se guardan en ficheros `.INI`— y ejecuta la aplicación. Si el método de invocación era POST, además, escribe los datos del cuerpo de la solicitud (es decir, los parámetros y sus valores) en el descriptor de archivo que se utiliza como entrada estándar de la aplicación.
- 3 La aplicación puede desempeñar cualquier función —como por ejemplo: realizar un cálculo o acceder a una base de datos— y tomar decisiones antes de formatear la información en HTML, para dejársela al servidor Web, el cual la enviará al cliente o navegador. Podemos considerar que el *know-how* ha pasado del servidor Web a la aplicación CGI.
- 4 La aplicación, una vez realizada su labor (o a medida que la realiza), escribe en su salida estándar el resultado, procurando hacerlo en un formato válido para el cliente Web: texto en formato HTML, MIME o XML, por poner algunos ejemplos. Precisamente, el servidor Web tendrá



redirigida esta salida a un descriptor propio, a través del cuál leerá este resultado.

5 El servidor Web enviará la información al cliente Web que realizó la invocación a modo de respuesta HTTP, conformando el cuerpo de la misma a partir de los datos obtenidos en el paso anterior.

6 Finalmente, el navegador Web, una vez recibida la respuesta, la interpretará y la mostrará (posiblemente tras recuperar mediante otras solicitudes otros recursos: gráficos, audio, etc.) al usuario de forma gráfica en su pantalla.

| Variable de Entorno | Valor                        |
|---------------------|------------------------------|
| REQUEST_METHOD      | GET                          |
| QUERY_STRING        | /cgi-bin/busca.php?clave=pdi |
| CONTENT_LENGTH      | 29                           |
| SCRIPT_NAME         | /cgi-bin/busca.php           |
| SERVER_PORT         | 80                           |

Figura 1.13. Contenido de algunas variables de entorno utilizadas por el protocolo CGI.

La propuesta CGI aporta una capacidad de respuesta dinámica al servidor Web, de modo que las respuestas ya no van a ser meras páginas HTML estáticas, sino que pueden estar en «*función de*». Proporciona, además, una razonable libertad de elección del lenguaje de desarrollo de la aplicación CGI, que puede realizarse mediante programación en lenguajes nativos de la plataforma y a los que probablemente estarán acostumbrados los programadores de la compañía, con lo que la curva de aprendizaje es mínima.

Como desventajas, hay que mencionar que con esta tecnología no hay relación entre el programa CGI y el servidor Web, pues ambos se ejecutan en procesos separados. Por tanto, no podemos controlar al programa CGI, es decir, no sabemos si terminó con éxito, si «*se colgó*» o si devolvió un resultado inesperado. Asimismo, como se instancia un programa CGI con cada petición que se realiza al servidor web, incluso aunque sea el mismo cliente el que la efectúe, el rendimiento no es el más óptimo y, además, se sobrecargan los recursos del nodo servidor. Finalmente, con CGI existe poca o nula portabilidad entre plataformas distintas, al escribirse el programa en un lenguaje nativo. Si el CGI se

realiza en lenguaje interpretado, por ejemplo tipo Perl o PHP, aumenta la portabilidad —sólo habría que tener el intérprete correspondiente en la otra máquina— a costa de reducir el rendimiento con respecto a un lenguaje compilado.

Analicemos algunos sencillos ejemplos que nos ayuden a comprender mejor el funcionamiento de los CGIs.

En primer lugar, el listado que se muestra en la figura 1.14 corresponde a una aplicación mínima creada mediante un guión de comandos (*script*) que podrá ejecutar casi cualquier intérprete de comandos (*shell*) de Unix. En este ejemplo, el proceso escribirá en su salida estándar el código correspondiente a una página HTML que es equivalente al ejemplo de *saludo* mostrado con anterioridad en la figura 1.7. Si un cliente invoca este script mediante una solicitud HTTP del tipo: GET /cgi/saludos.sh HTTP/1.1, el servidor Web se encargará de que dicho guión se ejecute y recogería el resultado de su salida estándar para devolvérselo al usuario. El resultado final para este usuario coincidirá con el del ejemplo 1.7, sólo que en esta ocasión, la página no reside físicamente en un sistema de archivos, en realidad se ha generado «*al vuelo*» (o de forma dinámica).

```
#!/usr/local/sh
# Comenzamos indicando quién debe interpretar el script.

# Sencillo script de shell que genera por la salida
# estándar una página HTML de saludo.

# Generamos la cabecera de la respuesta HTTP
print "Content-type: text/html"
print

# Generamos el cuerpo de la respuesta HTTP
print "<HTML>"
print "<HEAD>"
print "<TITLE>SALUDOS</TITLE>"
print "</HEAD>"
print "<BODY>"
print "<H1>¡Hola a Todos!</H1>"
print "</BODY>"
print "</HTML>"
```

Figura 1.14. Guión de comandos (*script de shell*) denominado `saludos.sh`, compatible con la mayoría de *shells* UNIX (`ksh`, `bsh`, `bash`), pensado para actuar como un programa CGI.

Otro ejemplo equivalente al anterior es el presentado en el listado de la figura 1.15. En este caso, en lugar de un *script*, se ha optado por una versión escrita en *lenguaje C* que se deberá compilar para generar el correspondiente binario ejecutable. El resultado de invocar este binario por parte de un cliente será, nuevamente, análogo al de ejecutar el anterior *script* (ejemplo 1.14) o invocar la página HTML del ejemplo 1.7.

```
/* Programa C utilizado como programa CGI  *
 * generando como salida una página html  *
 * con el mensaje "Hola a todos"          */

#include <stdio.h>

main(int argc, char *argv[]) {

    printf("Content-type: text/html\n\n");
    printf("<HTML>\n");
    printf("<HEAD>\n");
    printf("<TITLE>SALUDOS</TITLE>\n");
    printf("</HEAD>\n");
    printf("<BODY>\n");
    printf("<H1>;Hola a Todos!</H1>\n");
    printf("</BODY>\n");
    printf("</HTML>\n");

} /* de main */
```

Listado 1.15. Contenido del archivo `saludos.c` con la versión en lenguaje C de la aplicación CGI "Hola a todos".

Un ejemplo un poco más «*interactivo*» es el que se presenta en la figura 1.16. En este caso, vamos a determinar qué método de invocación se ha empleado leyendo la variable de entorno `REQUEST_METHOD`. En caso de haber empleado el método GET, los argumentos se encontrarán en la variable de entorno `QUERY_STRING`. Si, por el contrario, se ha empleado el método POST, el servidor Web nos habrá dejado estos argumentos en la entrada estándar. En cualquier caso, el tamaño de la cadena con los argumentos lo tendremos reflejado en la variable de entorno `CONTENT_LENGTH`.

Como resultado, este programa generará de forma dinámica el código HTML de una página en la que se mostrará cuales fueron los argumentos de la invocación.

```

/* Aplicación C escrita para ser ejecutada como      *
 * programa CGI, pensada para obtener los argumentos *
 * pasados por el navegador Web, ya sea a través de *
 * una invocación mediante el método GET, como el POST */

int main (void) {
    char metodo[100];
    char argumentos[1000];
    int tam;

    /* Obtenemos el método */
    strcpy(metodo, getenv("REQUEST_METHOD"));
    if (strcmp(metodo, "GET") = 0) {
        /* Argumentos en variable de entorno QUERY_STRING */
        strcpy(argumentos, getenv("QUERY_STRING"));
    }
    else {
        if (strcmp(metodo, "POST") = 0) {
            /* Argumentos en la entrada estándar */
            tam = atoi(getenv("CONTENT_LENGTH"));
            fgets(argumentos, tam + 1, stdin);
        }
        else
            printf("Error. Metodo no soportado");
    }

    /* Cabecera */
    printf("Content-type: text/html\n");
    printf("\n");

    /* Cuerpo */
    printf("<HTML>\n");
    printf("<HEAD>\n");
    printf("<TITLE>Argumentos de entrada</TITLE>\n");
    printf("<BODY>\n");
    printf("Argumentos del CGI: %s\n", argumentos);
    printf("</BODY>\n");
    printf("</HTML>\n");
}

```

Listado 1.19. Ejemplo en lenguaje C del tratamiento de las peticiones HTTP GET y POST desde una aplicación CGI.

### 1.3.1. Formularios Web

Tal y como ya se ha explicado, un cliente puede invocar una aplicación CGI mediante una solicitud HTML a través del método GET o POST. Sin embargo, si cada vez que un usuario desea invocar una función, debe

construir a mano una de estas solicitudes, todo el sistema iría en contra de su filosofía global de transparencia y facilidad de uso de cara al usuario. Además, de forma directa, un usuario sólo podría generar solicitudes del tipo GET en la que los parámetros y valores se pasan directamente en la línea de solicitud.

Una primera alternativa consistiría en generar URIs de forma manual e incluirlas mediante etiquetas HREF. El problema es que mediante este mecanismo no se pueden modificar los valores de los parámetros de las llamadas y, por lo tanto, se convierte en una solución parcial, adecuada sólo para casos muy concretos.

Por suerte, HTML prevé esta necesidad y proporciona una serie de etiquetas que permiten definir diferentes campos para la introducción de datos dentro de nuestra página, recoger la información introducida por el usuario en los mismos y generar automáticamente la solicitud HTTP adecuada para invocar una aplicación CGI con los valores introducidos por el usuario. Este tipo de páginas HTML que incluyen esta posibilidad se denominan *formularios HTML* y las etiquetas que lo hacen posible son: `<FORM>`, `</FORM>` e `<INPUT>`.

La etiqueta FORM es la que determina el alcance del formulario, permite definir el método de invocación a utilizar y la aplicación CGI a la que se hará referencia.

Las etiquetas INPUT definen los diferentes campos y botones que presentará el formulario para la introducción de información. Se pueden definir campos de tipo texto, desplegables o de selección. También se pueden definir botones genéricos y, sobre todo, se puede definir un tipo de botón especial denominado `submit`. Este botón es uno de los más importantes del formulario pues, cuando el usuario pulsa sobre él, desencadena que el navegador Web dé por concluida la introducción de datos, los recoja y genere con ellos la solicitud GET o POST adecuada para el servidor Web.

Según esto, podríamos definir un *formulario Web* como una página Web que incluye una serie de botones y campos para la recogida de datos junto con un botón para realizar la invocación.

La figura 1.20 muestra el código HTML de una página Web que incluye un sencillo formulario de búsqueda con un campo de introducción de texto y un botón para comenzar con la misma. En la figura 1.21 se puede observar cómo presentaría un navegador Web convencional este formulario. En este ejemplo, al pulsar sobre el botón «buscar» que es del tipo especial «`submit`», el navegador generaría una solicitud similar a:

```
GET /cgi-bin/busca.cgi?Nombre=Paco&Maciá
```

```
<HTML>
<HEAD>
<TITLE>Búsqueda...</TITLE>
</HEAD>

<BODY>
<H1>Formulario de búsqueda</H1>
<HR>

<FORM method="GET" action="buscar.cgi">
  Introduzca un nombre:<br>
  <INPUT NAME="Nombre"><P>
  <INPUT type="submit" value="buscar"><P>
</FORM>

<HR>
</BODY>
</HTML>
```

Listado 1.20. Formulario HTML con invocación a CGI.

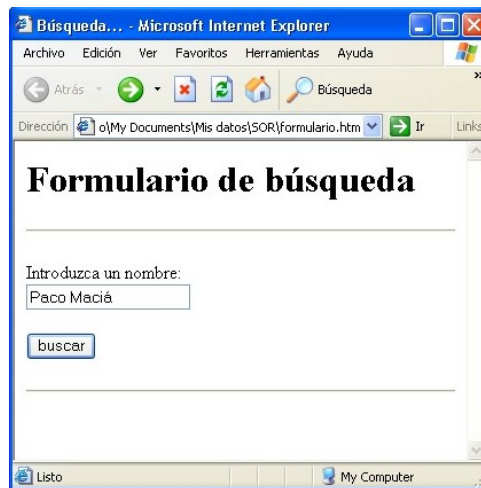


Figura 1.21. Representación gráfica por un navegador de la página HTML de la figura anterior y que incluye un formulario.