Tema 1
HTTP y sus limitaciones

J. Gutiérrez

Departament d'Informàtica
Universitat de València

DAW-TS (ISAW).
Curso 14-15

Índice

URI (1)

Un URI (*Uniform Resource Identifier*) es un texto que identifica a un recurso en la Web. El recurso puede ser un documento, una imagen, un fichero (pdf, zip, ...) , una dirección de correo electrónico o un procedimiento remoto.

Un URI tiene la siguientes estructura:

```
<scheme>:<scheme-specific-part>
```

Un subconjunto de los URI se representa del siguiente modo:

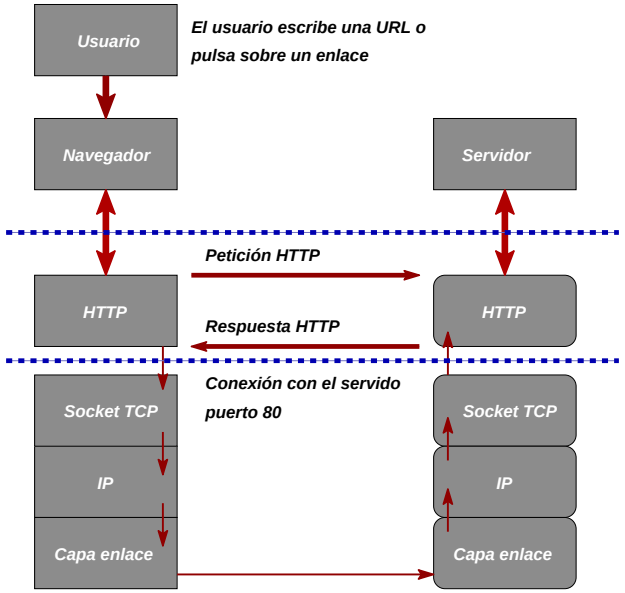
```
<scheme>://<authority><path>?<query>
```


URI (2)

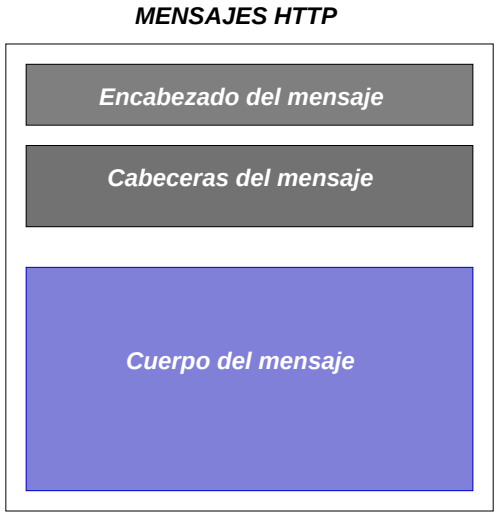
Ejemplos:

```
mailto:gujrreit@uv.es

ftp://ftp.is.co.za/rfc/rfc1808.txt

http://www.math.uio.no/faq/compression-faq/part1.html

telnet://melvyl.ucop.edu/

http://www.google.es/search?q=URI&ie=utf-8
```

Funcionamiento del protocolo HTTP



HTTP: Estructura de los mensajes



HTTP: Mensajes de petición (1)

Los mensajes de petición pueden ser: GET, POST, HEAD, DELETE o PUT.

El tipo de petición se incluye en el encabezado del mensaje.

GET

Codifica los parámetros de la petición en la URL.
Creado para peticiones que no modifican el estado del servidor (operaciones idempotentes).

POST

Codifica los parámetros de la petición en en el cuerpo del mensaje.
Creado para peticiones que modifican el estado del servidor.

HTTP: Mensajes de petición (2)

HEAD

Para pedir al servidor que nos devuelva sólo la cabecera del recurso.

PUT

Para poner un recurso en el servidor.

DELETE

Para borrar un recurso del servidor.

HTTP: Mensajes de respuesta

En el encabezado del mensaje de respuesta se encuentra el código de respuesta.

- *1xx* : Informan al navegador
- *2xx* : Indican que la respuesta es correcta (200: respuesta correcta).
- *3xx* : Indican redirección (300 y 301: Indican el que navegador debe ir a otra página).
- *4xx* : Indican errores en la petición (400: error en la petición, 401: no autorizado, 404: no encontrado)
- *5xx* : Indican errores en el servidor (500: error interno en el servidor, 501: no implementado).

Cabeceras (1)

Content-Type (petición y respuesta)
Describe el contenido del cuerpo del mensaje

Date (petición y respuesta)
Fecha en la que se ha creado el mensaje

Expires (respuesta)
Fecha y hora en la cual el contenido debe considerarse obsoleto.

Cabeceras (2)

If-Modified-Since (petición)
La página debe entregarse si ha sido modificada con posterioridad a la fecha indicada.

Location (respuesta)
URL absoluta de la página

Referer (petición)
URL desde la que se inicia la operación

User-Agent (petición)
Información sobre el software que usa el cliente para realizar la petición.

Cabeceras (3)

Set-Cookie (respuesta)
El servidor proporciona un identificador (cookie) para la sesión

Cookie (petición)
El navegador envía la cookie que le ha proporcionado el servidor

WWW-Authenticate (respuesta)
El servidor requiere usuario y contraseña para acceder al recurso

Authorization (petición)
El navegador envía el usuario y contraseña al servidor (ojo, esta información no va encriptada).

Practicar con peticiones, cabeceras y respuestas

Ejemplo

- Guardar el fichero `RespuestaHTTP.jar` (que está en Aula Virtual) en un directorio.
- Abrir una ventana de órdenes (`cmd`) e ir al directorio donde se ha guardado el fichero (`cd`).
- Ejecutar: `java -jar RespuestaHTTP.jar`
- En el primer campo de texto hay que indicar el `host`
- En el segundo campo de texto hay que indicar el `puerto`
- En el tercer campo de texto hay que indicar la `petición`
- En el área de texto hay que poner las `cabeceras` de la petición (cada cabecera en una linea).
- La respuesta del servidor se muestra en el área de texto inferior.

Índice

Sesión y estado (1)

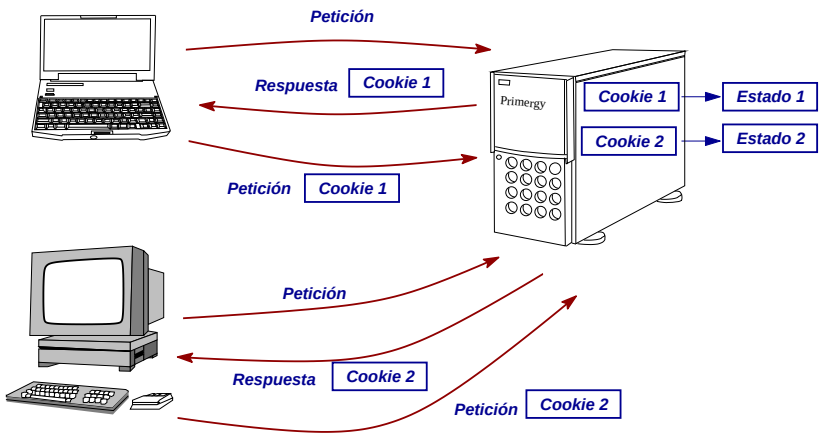
Problema: paradigma petición / respuesta.

Es necesario identificar las operaciones que realiza cada usuario (*sesión*) para almacenar el *estado* el resultado de las operaciones (por ejemplo: carro de la compra).

Solución: *cookies* y un objeto en el servidor que se mantiene mientras está activa la sesión.

Las cookies de sesión están en la memoria del navegador y no se guardan en disco

Sesión y estado (2)



Eventos (1)

El protocolo HTTP está diseñado para que la comunicación la inicie el cliente con una petición y la respuesta es un recurso enviado al cliente.

Eventos generados en el cliente

¿Qué ocurre si deseamos notificar al servidor que ha ocurrido algo (un evento) y que debe hacer alguna tarea que actualiza *una parte* de la página?

Eventos generados en el servidor

-Qué ocurre si deseamos que servidor notifique sobre algo (un evento) al cliente?

Eventos (2)

Ajax soluciona el primer problema

1. El cliente, usando JavaScript envía una petición al servidor,
2. La respuesta es capturada por JavaScript que ejecuta una función
3. En esta función es puede actualizar parte del contenido de la página.

¿Qué puede enviar el servidor?

- XML (lo recomendado pero costoso para el cliente)
- XHTML (el código XHTML que debe insertar el cliente)
- JSON (JavaScript Object Notation)

Hay frameworks JavaScript para trabajar con Ajax, acceder al DOM, y ofrecer funcionalidad avanzada (como por ejemplo drag and drop). Ejemplos: prototype, scriptacolous, JQuery, Dojo, Yahoo UI, etc.

Eventos (3)

Las primeras soluciones al segundo problema aparecieron con Bayeux.

El objetivo es soportar interacciones bidireccionales entre el cliente web y el servidor web.

Los mensajes se encaminan a través de canales y pueden ser enviados desde

1. el servidor al cliente (*server-push*).
2. el cliente al servidor.
3. un cliente a otro cliente (vía el servidor).

Eventos (4)

¿Cómo funciona?

El protocolo HTTP no especifica que el servidor deba enviar la respuesta inmediatamente, esto se puede aprovechar del siguiente modo:

1. El cliente realiza una petición
2. El servidor no contesta inmediatamente sino que espera a que se produzca algún evento en el que el cliente ha manifestado su interés (pueden haber varios canales).
3. El cliente procesa la respuesta y a continuación vuelve a realizar otra petición.

Eventos (5)

HTML5 incluye dos mecanismos para propagar eventos desde el servidor hasta el cliente:

- *Server-Sent Events* permite enviar información desde el servidor hasta el cliente.
- *WebSockets* establece un canal de comunicación bidireccional via *socket*.

La versión GlassFish 3.1 soporta ambas aproximaciones.

Otros elementos introducidos en HTML5

HTML5 incluye también:

Posibilidad de almacenar información en el cliente usando JavaScript. Para ello ofrecen una base de datos SQL en el cliente que ofrece almacenamiento estructurado de datos y almacenamiento de pares clave/valor.

```
// Los siguientes ejemplos son código JavaScript

// Almacenamiento estructurado:
if (window.openDatabase){
   db = openDatabase(...);
  ...
}

// Ejemplo de almacenamiento persistente del tipo clave/valor:
if (window.localStorage){
   localStorage.usuario=''Juan'';
   var user = localStorage.usuario;
}

// Ejemplo de almacenamiento asociado a la sesión del tipo clave/valor
if (window.sessionStorage){
   sessionStorage.usuario=''Juan'';
   var user = sessionStorage.usuario;
}
```


Índice

Índice

Responsabilidades principales

1. Generar y enviar peticiones a los servidores Web (el proceso lo inicia el usuario).
2. Procesar la respuesta e interpretarla para determinar las acciones necesarias para poder mostrar la página y su contenido.
3. Mostrar los resultados en la ventana del navegador o delegar en extensiones instaladas (plugins).

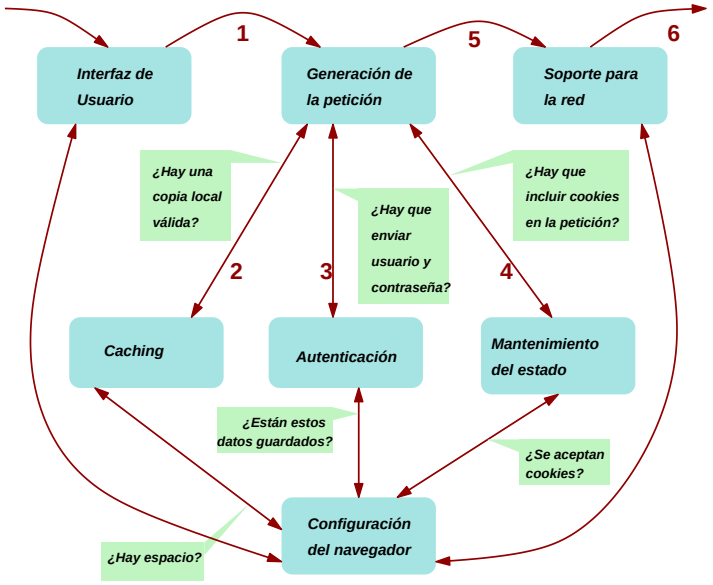
Tareas adicionales
Que dependen del código de estado y de los campos de cabecera de la respuesta

1. *Caching (almacenamiento)*: ¿se almacena la respuesta?, ¿hay una respuesta almacenada y no hace falta solicitarla al servidor?.
2. *Autenticación*: si el servidor pide autenticación el navegador debe mostrar al usuario una ventana en la que introduzca su usuario y contraseña para acceder al recurso.
3. *Mantenimiento del estado*: si el servidor envía una cookie, el navegador debe almacenarla y enviarla en las siguientes peticiones.

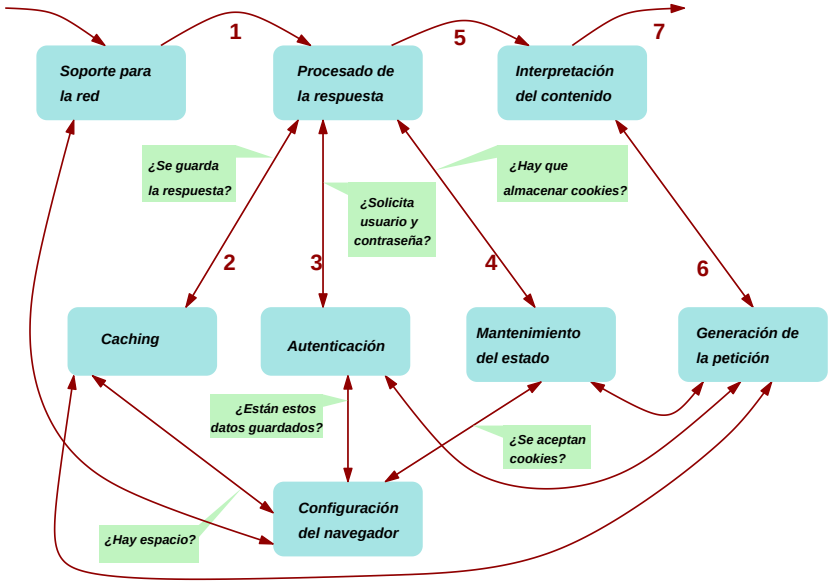
Tareas adicionales
Que dependen del código de estado y de los campos de cabecera de la respuesta

4. *Solicitar datos adicionales*: código (javascript, applets java, flash, ...), imágenes, sonidos, instrucciones para la presentación (css), ...
   El navegador analiza (*parse*) el código HTML y por cada recurso al que se hace referencia debe realizar una petición.
5. *Tomar acciones en respuesta a otros campos de cabecera o códigos de estado*: por ejemplo el servidor puede enviar un código de estado de redireccionamiento.
6. *Mostrar objetos complejos*: los navegadores soportan de forma nativa contenido de diverso tipo: (text/html, text/plain, image/gif, image/jpeg, ...). El contenido que no es soportado se delega en extensiones.
7. *Tratar las condiciones de error*.

Acciones para generar la petición

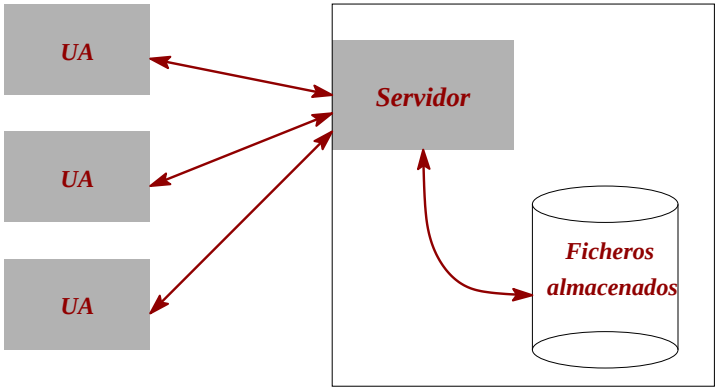


Acciones para procesar la respuesta



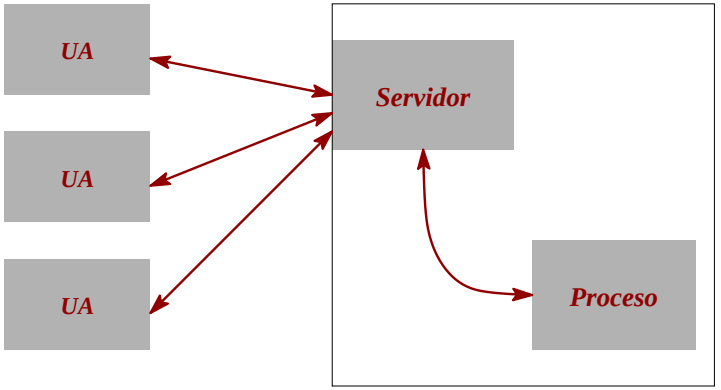
Índice

Esquema (simplificado) servidor estático

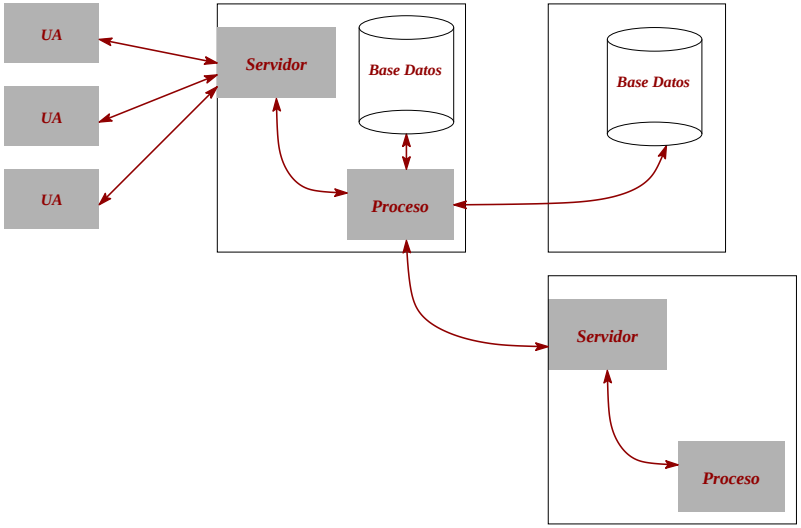


Ejemplo: Httpd (Apache)

Esquema (simplificado) servidor dinámico



Esquema (simplificado) servidor dinámico (II)
¡El proceso puede requerir servicios que están en otros servidores!



¿Qué posibilidades hay para realizar ese proceso?

Aproximaciones que requieren programación

Se realizan por completo usando un lenguaje de programación

- CGI: un proceso que está escrito en algún lenguaje de programación (script shell, Perl, Python, c, c++, ...) y que es llamado cuando se produce la petición.
- Servlet: un objeto Java cuyo ciclo de vida es gestionado por el servidor (a través de un contenedor).

Aproximaciones basadas en plantillas

Mezclan código de algún lenguaje de programación con código HTML:

- JSP
- PHP
- ASP