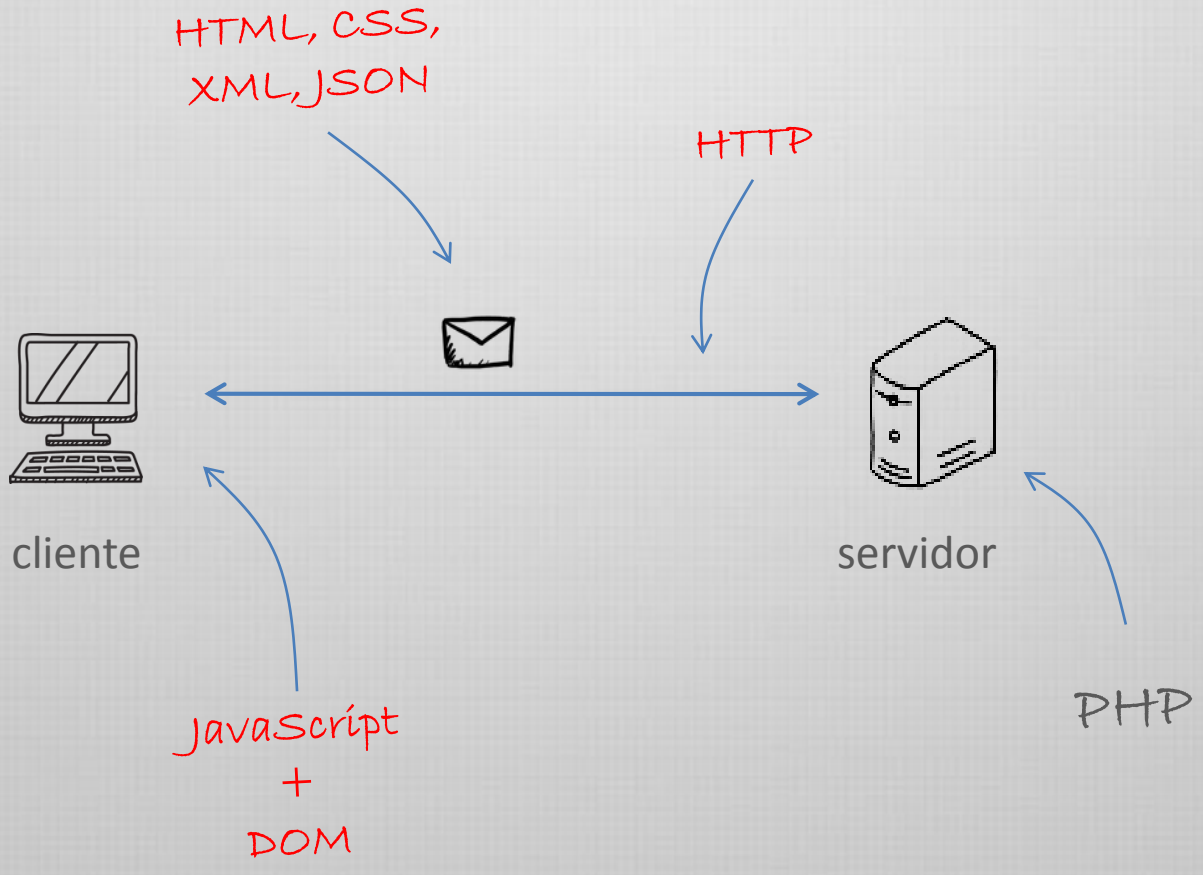
Ingeniería de Aplicaciones Web

Diego C. Martínez

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Tecnologías web



modelo de interacción para aplicaciones web,
basado primordialmente en dos tecnologías:
JavaScript y *XML*

AJAX

Asynchronous **J**avaScript **a**nd **X**ML

Es básicamente una técnica para implementar ciertos comportamientos en aplicaciones web.

La idea general es solicitar en background información a un servidor mientras el cliente opera sobre el navegador (por eso asincrónica).

Antes "Inner-Browsing"

no es una tecnología, sino una agrupación de varias técnicas y tecnologías unidas para lograr un mismo efecto.

AJAX

- Presentación basada en estándares: *XHTML*, *CSS*.
- Páginas dinámicas, utilizando *DOM*.
- Intercambio y manipulación de datos *XML y JSON*
- Recuperación asincrónica de datos, usando *XMLHttpRequest*

JavaScript

El corazón de AJAX

Chrome, Internet Explorer, Opera, Safari, *Firefox, Konqueror* y otros proveen todos un método para que los scripts JavaScript puedan realizar requerimientos HTTP.

Se realiza por medio de un objeto denominado **`XMLHttpRequest`** *("XHR")*

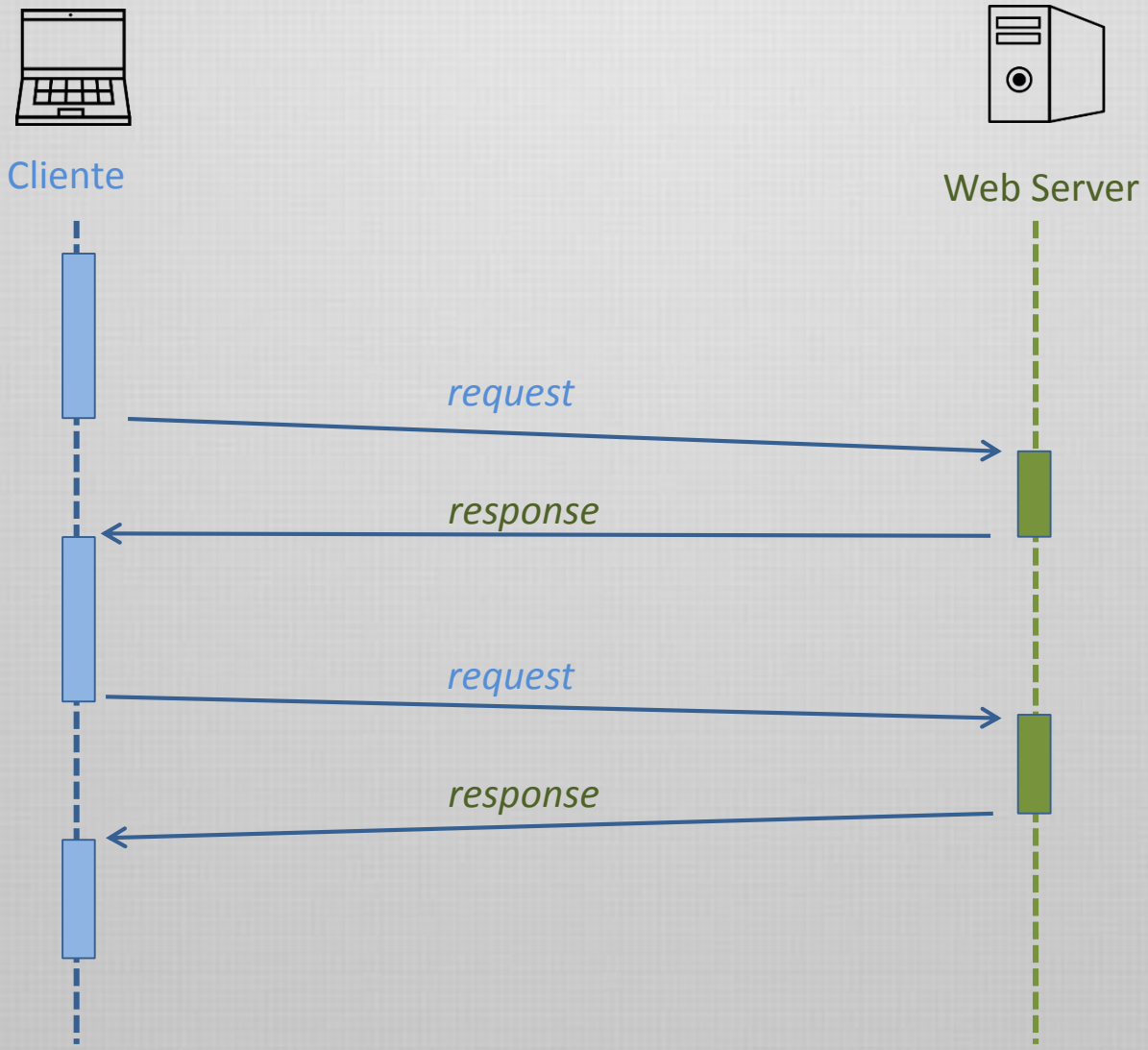
Aunque XML aparece en el nombre, no se limita sólo a ese formato.

```
var req = new ActiveXObject("Microsoft.XMLHTTP")
var req = new XMLHttpRequest()
var req = window.createRequest()
```

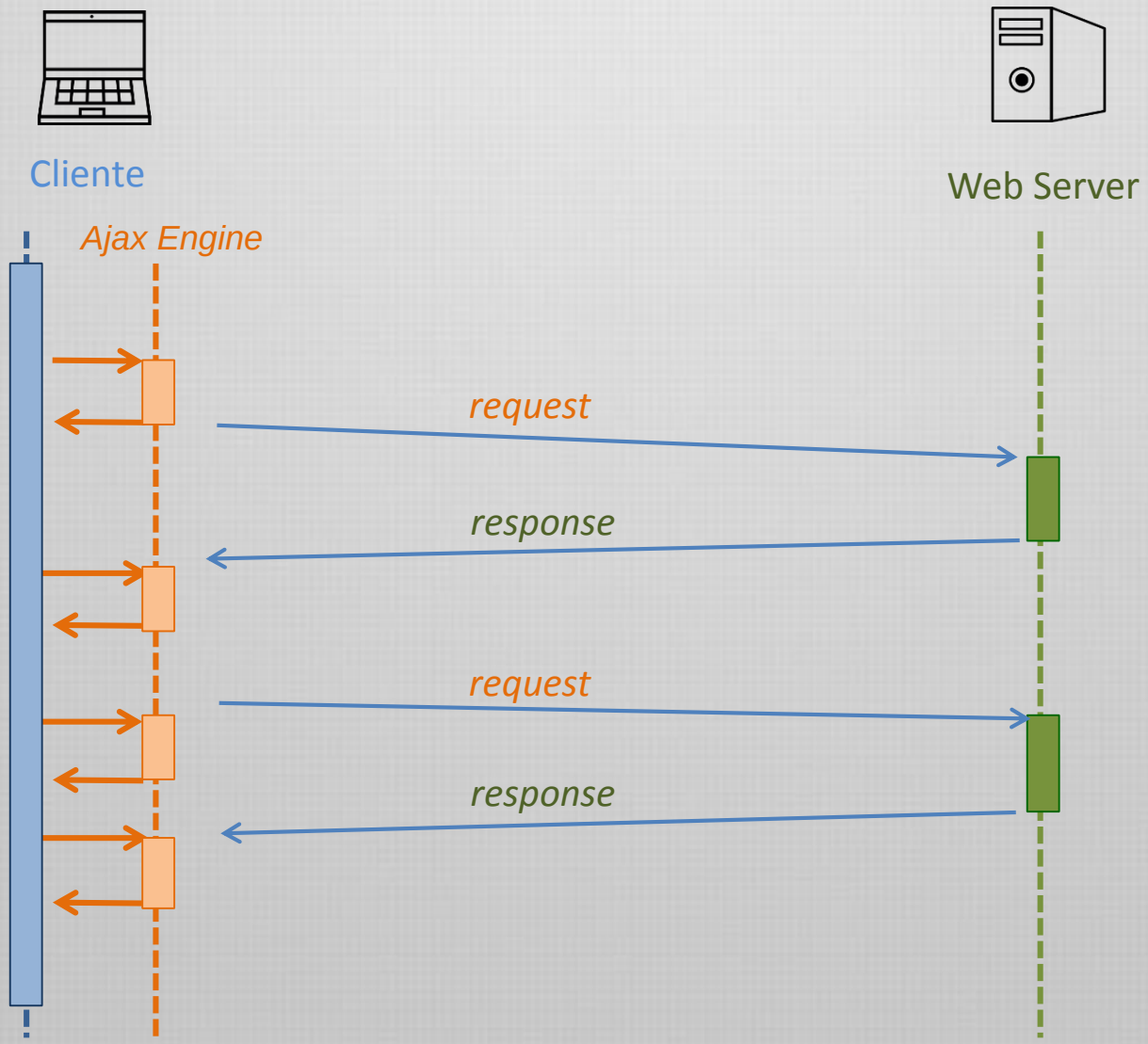
Una vez obtenido el objeto, realizar un pedido HTTP es fácil.

Hay que tener en cuenta que el pedido se realiza en background, en forma asincrónica, por lo que no "esperamos" por una respuesta...

Modelo *tradicional*



Modelo *Ajax*


Cliente
Web Server
Ajax Engine
request
response
request
response

XMLHttpRequest

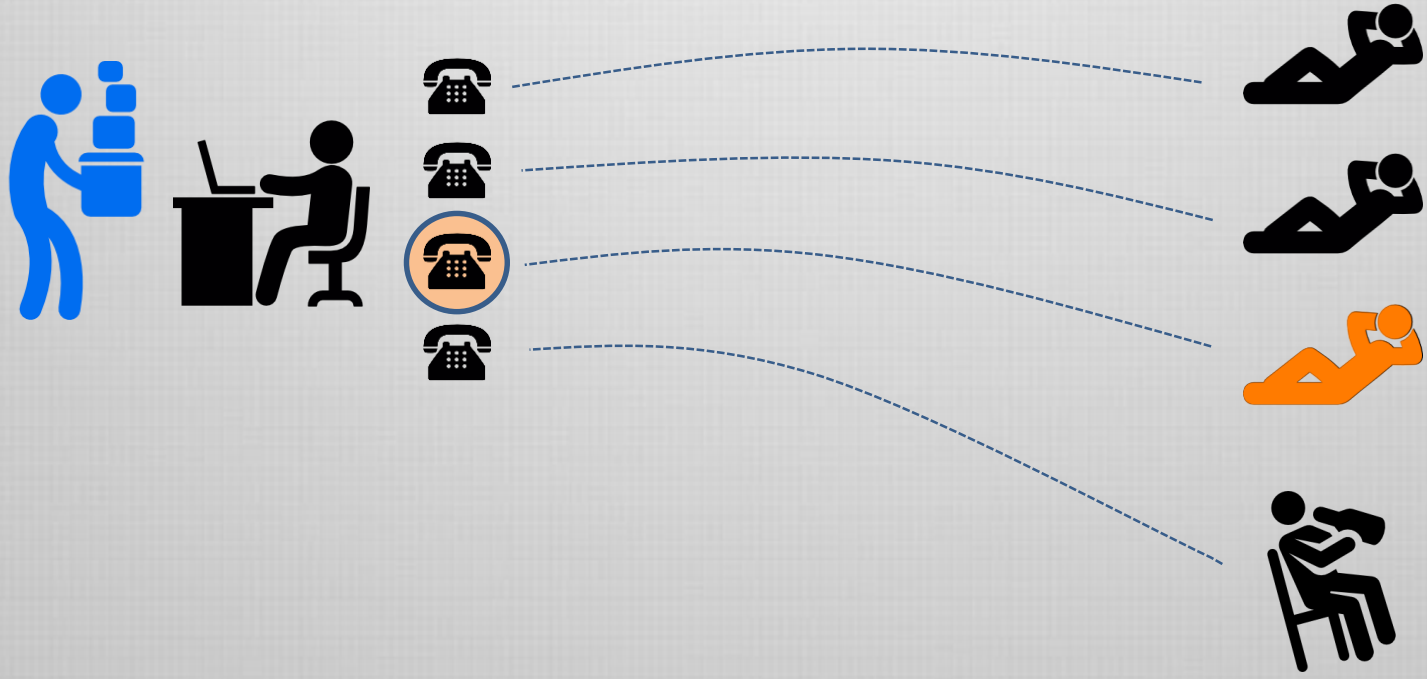
```
if (window.XMLHttpRequest){
      // IE7+, Firefox, Chrome, Opera, Safari
      xmlhttp=new XMLHttpRequest();
} else {
      // IE6, IE5
      xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
```

```
xmlhttp.open(metodo,URL,asincronico);
xmlhttp.send();
```

Si el request es asincrónico, hay que indicar previamente qué sucederá cuando se complete la recepción de la respuesta del servidor...

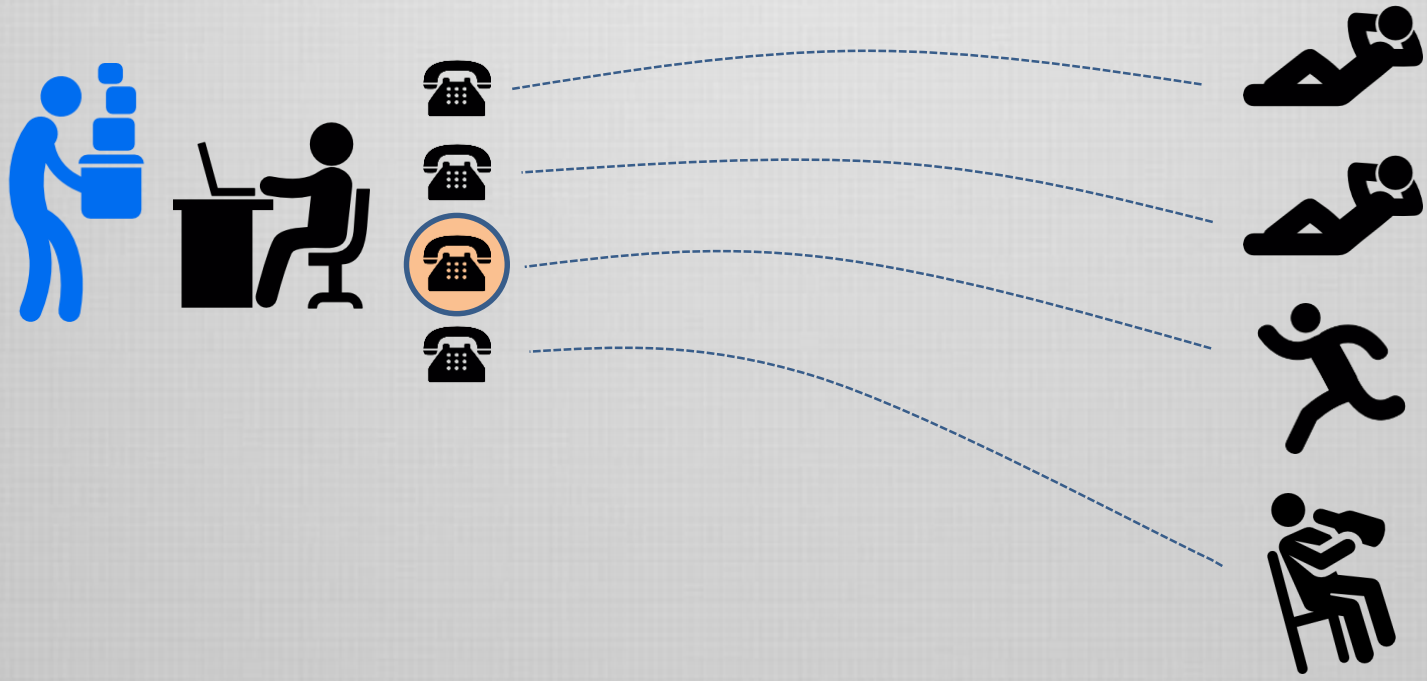
Call-back

Las funciones *call-back* son funciones que se invocan ante la ocurrencia de ciertos eventos.



Call-back

Las funciones *call-back* son funciones que se invocan ante la ocurrencia de ciertos eventos.



XMLHttpRequest en acción

```
req.onreadystatechange = processReqChange;
req.open("GET", url, true);
req.send();
```

onreadystatechange es una propiedad del objeto *XMLHttpRequest* que almacena la función que procesa la respuesta del servidor.

Es el oyente de cambio de estado: la función se invoca cada vez que cambia la propiedad **readyState**.

- después de invocar a *open*, **readyState** = 1.
- después de invocar a *send*, **readyState** = 2.
- mientras se recibe la respuesta, **readyState** = 3
- al finalizar la respuesta, **readyState** = 4.

¡La respuesta está lista para procesar!

XMLHttpRequest en acción

```
req.onreadystatechange = processReqChange;
req.open("GET", url, true);
req.send();
```

La respuesta del servidor se almacena finalmente en un atributo del objeto XMLHttpRequest.

Si la respuesta esperada es *texto plano*, se almacena en **`responseText`**

```
function processReqChange(){
 if(req.readyState == 4){
      document.getElementbyId(id).innerHTML = req.responseText;
 }
}
```

XMLHttpRequest en acción

```
req.onreadystatechange = processReqChange;
req.open("GET", url, true);
req.send();
```

El atributo **responseXML** es similar, pero nos permite encapsular la información para ser interpretada como XML.

Al ser un XML, tiene una representación DOM y podemos usar operaciones para obtener los datos contenidos en ese documento!

```
function processReqChange(){
 if(req.readyState == 4){
    var xmldoc = req.responseXML;
    var nodo = xmldoc.getElementsByTagName('root').item(0);
 }
}
```

XMLHttpRequest en acción

```
req.onreadystatechange = processReqChange;
req.open("GET", url, true);
req.send();
```

Las operaciones ***open*** y ***send*** inician el proceso de pedido al servidor.

recibe como parámetros:

el método (GET, POST, HEAD, etc)

+

el URL (que puede ser absoluto o relativo) y

+

un indicativo de si el pedido es asincrónico o no.

El valor "true" como último argumento indica que el script no esperará la respuesta luego de la invocación a la operación send, la cual comienza el pedido antes configurado

Ejemplo Ajax

Con lo visto hasta el momento es fácil crear una simple aplicación que utilice esta técnica.

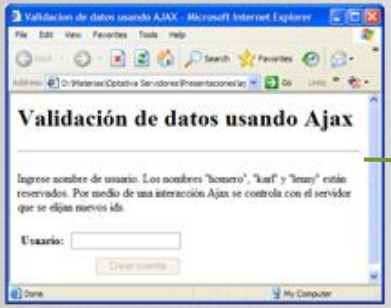
Del lado cliente, haremos pedidos *HTTP* a un servidor por medio del objeto *XMLHttpRequest*. Recibiremos la información y la procesamos.

Del lado servidor, una aplicación (por ejemplo Java) o script (por ejemplo PHP) generará la respuesta solicitada por el cliente.

Los dos extremos colaboran creando el efecto deseado.

Es importante destacar que las tecnologías usadas tanto del lado cliente como del lado servidor son independientes y por lo tanto pueden cambiar.

Ejemplo Ajax – validación de datos



onkeyup

```
validateUserID(){
    // controlar id
}
```

JavaScript

validar.php?userid=...

Server de Validación

XMLHTTPRequest

```
function callback(){
  //actualizar DOM
}
```

JSON

La tendencia es utilizar JSON como estructura de datos, en lugar de XML:

```
<albums>
      <album>
        <titulo>
            Master of Puppets
        </titulo>
        <artista>
            Metallica
        </artista>
      </album>
      <album>
        <titulo>
            The Division Bell
        </titulo>
        <artista>
            Pink Floyd
        </artista>
      </album>
<albums>
```

171 bytes

```
{
  'album':
       [
       {
       'titulo' : 'Master of Puppets',
       'artist' : 'Metallica',
        },
       {
       'titulo' : 'The Division Bell',
       'artist' : 'Pink Floyd',
       }
       ]
}
```

120 bytes

La principal ventaja es que al ser un formato de datos propio de JavaScript, es fácil interpretarlo. No hace falta parsing como en XML.

Algunos usos de Ajax

Validaciones de datos de formularios en tiempo real.

Identificaciones de usuario, números de serie, códigos postales, y otros datos pueden ser validados en el formulario antes del envío de los datos.

Autocompletamiento

Algunos datos pueden ser autocompletados mientras el usuario escribe.

Operaciones de detalles de datos

Información detallada puede ser obtenida por medio de eventos simples del cliente.

Controles de interfaz de usuario

Arboles, menúes, barras de progreso se implementan sin necesidad de refresh de página completa.

Refresh de datos en la página.

Algunos datos son obtenidos del servidor y mostrados en la página.

Algunas desventajas de Ajax

Complejidad

Se necesita lógica de presentación en el cliente y generación y envío de documentos XML en el servidor. Requiere conocimientos de JavaScript.

Depuración

El proceso es complicado porque la lógica se distribuye en el cliente y en el servidor.

Código expuesto

Parte del código de la aplicación queda expuesto al ejecutarse en el cliente.

Indexado

Muchos web crawlers no encontrarán el contenido obtenido via AJAX.

Bookmarks

Probablemente los *bookmarks* no registren el último estado de la página

Websockets

¿en qué se diferencia de AJAX?

¿en qué casos conviene usar cada uno?