2. AMPLIACIONES EN EL CLIENTE

El navegador Web ha sido adoptado por la comunidad de usuarios de Internet como la aplicación cliente por excelencia para este entorno. No obstante, la Web se ha convertido en un medio complejo donde las aplicaciones requieren de más potencia, sin exceder los requisitos deseados por los usuarios finales (aplicaciones ligeras, mínimos requisitos de instalación y configuración, lógica de negocio, separación entre estilo y datos). Son numerosas las tecnologías que han permitido extender el navegador Web inicial para acercarlo a un contenedor en la parte del cliente que cumpla los requisitos exigidos en estas aplicaciones.

En un principio el navegador Web solamente ofrecía al cliente la posibilidad de visualizar información de tipo estática (HTML). Con la evolución de las *.COM* la competencia por presentar páginas Web más atractivas para el usuario y con una mayor funcionalidad han surgido diferentes tecnologías que permiten incorporar características multimedia en las páginas e interactuar con el usuario. De esta forma ha sido necesario ampliar las funcionalidades del navegador Web para que soporte todas estas características.

2.1. SCRIPTS EN EL CLIENTE

Una de las técnicas utilizadas para dotar de dinamismo las páginas Web en la parte del cliente ha sido el uso de código incrustado dentro del código HTML estático. Estos fragmentos de código son interpretados en tiempo de ejecución en el cliente por el navegador Web y se denominan *SCRIPT*. De esta forma permite dotar a la página Web de efectos y funcionalidades. Las páginas de este tipo dependen de la plataforma y más concretamente del navegador sobre el cual se ejecuten. La ventaja que pueden ofrecer es la rápida respuesta ante acciones del usuario permitiendo la descarga del

servidor. No obstante, se debe contemplar el enorme número de navegadores Web existentes y la compatibilidad con los diferentes lenguaje de tipo *Scripts*.

2.1.1. El lenguaje JavaScript

JavaScript es un lenguaje interpretado desarrollado por Netscape bajo el nombre de *LiveScript* y que tras formar la alianza SUN y Netscape pasó a tomar el nombre con el que hoy en día es conocido. El objetivo era crear un lenguaje con características similares, en lo que a sintaxis se refiere, a los lenguajes de alto nivel pero de uso sencillo, sin necesidad de utilizar compiladores. Simplemente, se introducía código JavaScript dentro de un documento HTML y el navegador se encargaba de interpretar este código.

Como se ha comentado, el código de lenguaje JavaScript se introduce dentro del código HTML. Existen cuatro formas diferentes de incrustar código Javascript.

- Entre las etiquetas `<script>` y `</script>`
- Especificando un archivo JavaScript en el código HTML (*.*js*).
- Especificando una expresión Javascript como valor de un atributo HTML.
- Especificando código Javascript en un evento de los controles HTML que los contemplen (ej. botón).

2.1.1.1. Etiqueta de marcado y archivos JavaScript

Para diferenciar el código HTML del código JavaScript se utiliza la etiqueta de marcado `<SCRIPT>` para indicar que comienza un fragmento de código JavaScript y `</SCRIPT>` para indicar el final.

A lo largo de un documento HTML se pueden utilizar varias veces dichas etiquetas, pero siempre que se introduzca una etiqueta de apertura se deberá introducir una de cierre.

```
<SCRIPT>
  window.alert(“Hola Mundo JavaScript!!!”)
</SCRIPT>
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
      Hola Mundo HTML!!!
  </BODY>
</HTML>
```

Listado 2.1. Ejemplo sencillo de introducción de javascript en un documento HTML.

En principio no existe ninguna norma que indique dónde se debe introducir las etiquetas JavaScript, aunque es importante conocer cómo el navegador interpreta el código para no obtener resultados no deseados.

Cuando un cliente realiza una petición a un servidor Web el servidor responde con un documento que puede contener HTML y código JavaScript. El navegador Web comienza a analizar el código del documento HTML y va interpretando el código, bien sea JavaScript o HTML, y procediendo a su ejecución. En el ejemplo anterior primero se mostraría en pantalla una ventana con el mensaje `Hola Mundo JavaScript!!!` Posteriormente se procedería a mostrar el documento HTML.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
      Hola Mundo HTML!!!
  </BODY>
</HTML>
<SCRIPT>
  window.alert(“Hola Mundo JavaScript!!!”)
</SCRIPT>
```

Listado 2.2. Ejemplo sencillo de introducción de JavaScript en un documento HTML.

En este último caso primero, se mostraría en pantalla el mensaje `Hola Mundo HTML!!!` Posteriormente se mostraría una ventana con el mensaje `Hola Mundo JavaScript!!!`

Para separar el código HTML del de JavaScript y poder gestionarlo y mantenerlo de una forma óptima podemos utilizar los archivos de extensión *`*.js`*. De esta forma tendremos archivos con el contenido JavaScript que

anteriormente se encontraba entre las etiquetas `<script>` y `</script>`. Esta forma de incrustar JavaScript nos permite, además, reutilizar funcionalidades comunes a varias páginas. Es posible combinarlo con la forma de introducir JavaScript anteriormente mencionada.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
      Hola Mundo HTML!!!
  </BODY>
</HTML>
<SCRIPT SRC=”saludo.js”>
  ...
</SCRIPT>
```

Listado 2.3. Ejemplo sencillo de introducción de JavaScript en un documento HTML mediante archivos **.js*.

```
window.alert(“Hola Mundo JavaScript!!!”)
```

Listado 2.4. Contenido del archivo `saludo.js`.

El atributo `SRC` puede contener una *URL* relativa o absoluta.

En estas dos formas de incrustar código JavaScript, junto con la tercera manera, el navegador en cuanto recibe el documento desde el servidor analiza el código y directamente va interpretándolo, mostrando el resultado al usuario.

2.1.1.2. Valores de atributos y eventos

Otra de las posibles maneras de incrustar código JavaScript es introduciéndola dentro de las etiquetas del código HTML, bien como valores de atributos, bien como valores de eventos de los controles.

En el primer caso el valor del atributo HTML es creado de manera dinámica de tal forma que cuando el navegador vaya a interpretar el código HTML interpretará la expresión JavaScript asociada.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    <HR WIDTH=”&{barWidth};%” ALIGN=”LEFT”/>
    Hola Mundo HTML!!!
  </BODY>
</HTML>
```

Listado 2.5. Ejemplo de código HTML dinámico mediante JavaScript.

En el segundo caso se produce como respuesta a una acción realizada por el usuario, un evento. Mediante JavaScript es posible controlar las acciones que realiza un usuario al interactuar con la página Web y de esta forma realizar una funcionalidad ante el comportamiento del usuario. Para conseguir esta segunda forma, el código JavaScript debe ser introducido en una serie de atributos del código HTML.

```
<SCRIPT>
  function saludar()
  {
      Document.alert(“Hola Mundo Javascript”);
  }
</SCRIPT>
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    <input type=button value=enviar
    onclick="saludar()"/>
    Hola Mundo HTML!!!
  </BODY>
</HTML>
```

Listado 2.6. Ejemplo de evento en JavaScript.

Cuando se pulsa el control de tipo botón con la leyenda `enviar` se llama al evento `onclick` del control que realiza una llamada a la función JavaScript `saludar()`.

2.2. HTML DINÁMICO: DHTML

DHTML (acrónimo de *dynamic HTML*) no es un lenguaje de *Script* como Javascript. Se trata de la unión de varias tecnologías para crear sitios Web dinámicos e interactivos. DHTML utiliza HTML estático, un lenguaje de tipo *Script* de lado del cliente (como JavaScript), un lenguaje de definición de estilos para la presentación (por ejemplo *CSS*) y el modelo de objetos de documentos (*DOM*). DHTML no es una tecnología por sí misma si no que es la unión de un conjunto de éstas.

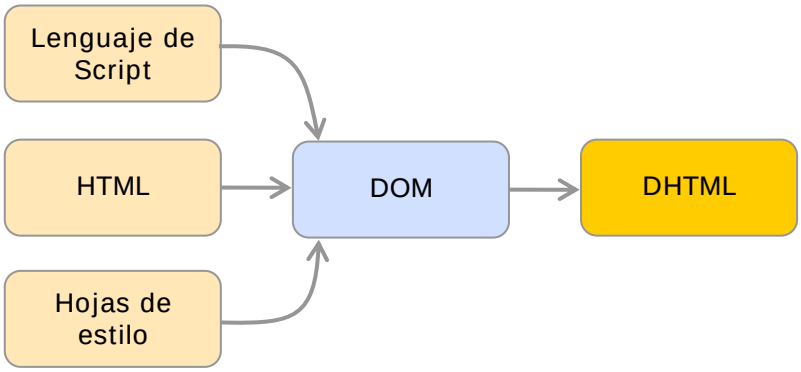


Figura 1.2. Ejemplo de solicitud HTTP

La problemática que puede surgir del uso de DHTML es que al ser una agrupación de tecnologías dependientes del navegador, se pueden encontrar problemas de interpretación en función del navegador utilizado.

2.2.1. Hojas de estilo en cascada: CSS

CSS es el acrónimo de *Cascading Style Sheets* u hojas de estilo en cascada. Se trata de un lenguaje de definición de estilo para la presentación de documentos escritos con lenguajes de marcado (tipo XML). La aparición de esta tecnología ha permitido separar los estilos de presentación de un documento Web permitiendo dar un mayor dinamismo a los sitios Web. Con este lenguaje podemos definir los colores, fuentes y otros aspectos de la presentación.

2.2.1.1. Introducción de hojas de estilo en documentos HTML

Existen tres formas de dar estilo mediante el lenguaje CSS a un documento HTML.

- Mediante un archivo externo (**.css*) que se enlace al documento HTML

```
h1 {color: blue; text-align: center}
```

Listado 2.7. Ejemplo de archivo CSS.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
    <LINK rel=”stylesheet” type=”text/css”
    href=”http://www.dtic.ua.es/grupoM/presentacion.css”/>
  </HEAD>
  <BODY>
    <h1>Hola mundo azul</h1>
  </BODY>
</HTML>
```

Listado 2.8. Ejemplo de referencia de archivo CSS en HTML.

- Mediante el uso del elemento `<style>` dentro del documento HTML

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
    <STYLE type=”text/css”>
      h1 {color: blue; text-align: center}
    </STYLE>
  </HEAD>
  <BODY>
    <h1>Hola mundo azul</h1>
  </BODY>
</HTML>
```

Listado 2.9. Ejemplo de estilo mediante el uso de <style>.

- Mediante el uso del atributo `style` dentro de los elementos HTML

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    <h1 style="color: blue">Hola mundo azul</h1>
  </BODY>
</HTML>
```

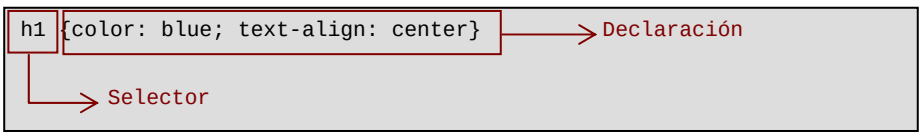
Listado 2.10. Ejemplo de estilo mediante el atributo style.

Este último caso es poco recomendable puesto que no permite una separación entre contenido y presentación.

2.2.1.2. Funcionamiento de las hojas de estilo CSS

CSS funciona a través de la declaración de reglas que establecen los estilos sobre determinados elementos. Un documento de estilos puede contener una o más reglas que se aplicarán a determinados documentos HTML o XML. Las reglas se componen de dos partes:

- Selector que identifica la regla. Normalmente se refiere a la etiqueta o elemento HTML al que queremos dar el estilo. En otras ocasiones, es un nombre lógico único el cual es especificado en el atributo class de un elemento o etiqueta HTML al que queremos darle un estilo determinado. Esta última forma de uso es utilizada cuando diferentes elementos HTML tienen el mismo estilo. Además, en el primer caso podemos establecer varios estilos para un mismo elemento a través del atributo `class`.
- Declaración que especifica los estilos determinados para un elemento HTML.



Listado 2.11. Declaración de estilos.

En el primer ejemplo todos los elementos del documento HTML, `h1`, establecerían el texto entre dichas etiquetas a color azul y centrado.

Si yo quiero establecer varios estilos para el elemento `h1` se debería utilizar el nombre del elemento `h1` seguido por un `'.'` y un nombre lógico

para cada regla. En el documento HTML, en cada elemento `h1` se debería utilizar el atributo class indicando el identificador del estilo a aplicar.

```
h1.primero {color: blue; text-align: center}
h1.segundo {color: red; text-align: center}
```

Listado 2.12. Ejemplo de archivo CSS.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    <h1 class="primero">Hola mundo azul</h1>
    <h1 class="segundo">Hola mundo rojo</h1>
  </BODY>
</HTML>
```

Listado 2.13. Uso de estilos en HTML.

2.2.2. El modelo de objetos de documento: DOM

El Modelo de Objetos de Documento (*Document Object Model*) define una manera de representar los elementos de un documento estructurado, tal como HTML o XML, como objetos con sus métodos y propiedades y como se accede y manipula el documento a través de sus objetos. Esta tecnología presenta una manera de acceder a los elementos de un documento de este tipo, tanto a sus elementos como a sus atributos, permitiendo añadir, eliminar o modificar estos elementos. Por lo tanto se puede definir como una interfaz de programación de aplicaciones (API) para documentos HTML o XML.

Con el Modelo de Objetos de Documento los programadores pueden construir documentos, navegar por su estructura, y añadir, modificar o eliminar elementos y contenido. Se puede acceder a cualquier cosa que se encuentre en un documento HTML o XML, y se puede modificar, eliminar o añadir usando el Modelo de Objetos de Documento, salvo algunas excepciones. En particular, aún no se ha especificado las interfaces DOM para los subconjuntos internos y externos de XML.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo</TITLE>
  </HEAD>
  <BODY>
    <TABLE>
      <TR>
        <TD>Nombre</TD>
        <TD>Apellidos</TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

Listado 2.14. Documento XML.

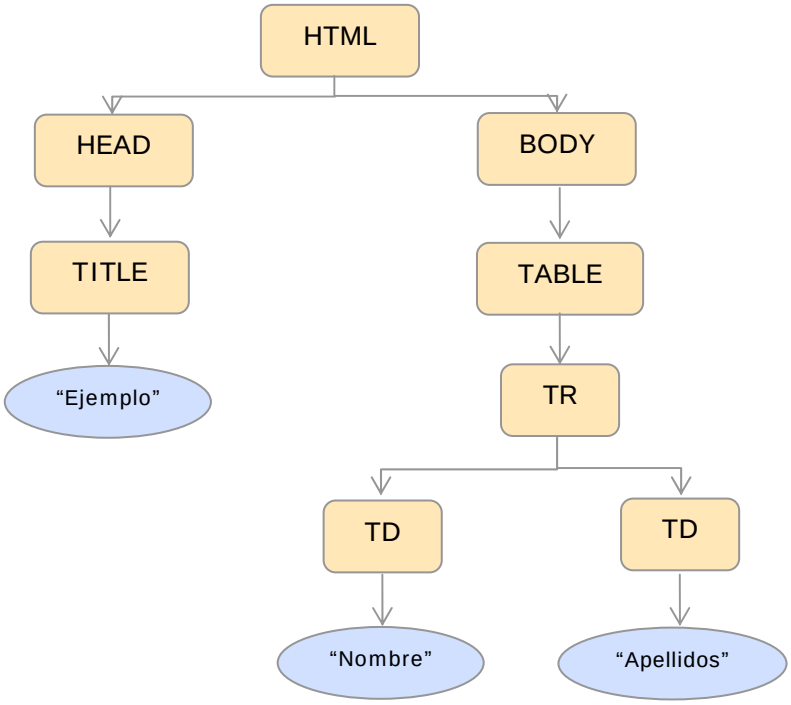


Figura 2.3. Representación estructurada del documento XML.

Los documentos DOM tienen una estructura jerárquica similar a una estructura de árbol pero no tiene porque mantener necesariamente esta organización. Puede además estar compuesto de varios documentos con este tipo de estructura.

2.3. JAVASCRIPT Y XML ASÍNCRONO: AJAX

AJAX es el acrónimo de *Asynchronous JavaScript And XML*. Al igual que DHTML no se trata de ningún lenguaje de programación. AJAX es una técnica de desarrollo de aplicaciones Web que combina un conjunto de tecnologías con el objetivo de proveer al cliente una navegación ágil y rápida y convirtiendo el navegador en un entorno muy dinámico. Entonces, ¿Cuál es la diferencia con DHTML? Básicamente, se puede decir que AJAX es una combinación de DHTML junto con otra tecnología que es la que marca la principal diferencia, el *objeto XMLHttprequest* (este elemento tampoco es nuevo y fue propuesto por Microsoft en 1998). La verdadera diferencia de esta técnica radica en su filosofía de funcionamiento, dónde de forma transparente al usuario, el cliente mantiene una comunicación asíncrona con el servidor en un segundo plano. Realiza peticiones GET y POST obteniendo un documento XML como resultado y utilizando DOM le permite leer los datos y modificar la página. A efectos del usuario se verá que la página cambia de aspecto sin la necesidad de recargarla.

Las tecnologías básicas usadas para el desarrollo de aplicaciones Web con AJAX son:

- *XHTML y CSS* como estándares de presentación.
- *DOM* para mostrar e interactuar con la información.
- *XML y XSLT* manipulación e intercambio de datos (en la parte del servidor).
- *XMLHttpRequest* para el envío y la recepción de información de forma asíncrona.
- *JavaScript* como enlace para gestionar todas las tecnologías anteriores.

2.3.1. Modelo de aplicaciones Web con AJAX

Tradicionalmente, en las aplicaciones Web el usuario interactúa con el servidor a través de un formulario, que una vez completado, se le envía por medio del navegador como una petición Web. Este tipo de aplicaciones tienen ciertos límites y no es posible crearlas con las mismas características que las aplicaciones de escritorio (versatilidad, interactivas, etc.). Cada vez

que se requieran datos del servidor se debe recargar elementos comunes (información de presentación). Esto hace, por un lado, que la transferencia de información sea mayor y por otro, que el usuario tenga que estar a la espera de cada nueva recarga.

Fundamentalmente, AJAX establece una capa entre la interfaz del cliente y la aplicación de servidor, que hace que las operaciones de comunicación y trasiego de información junto con el refresco de datos de cara al usuario se hagan de manera transparente para éste. De esta forma se consigue parte de la potencia de las aplicaciones de escritorio y además se mejora la comunicación, puesto que la nueva capa introducida (*motor AJAX*) solamente obtiene los datos necesarios en cada momento sin necesidad de descargar la estructura del documento en cada operación.

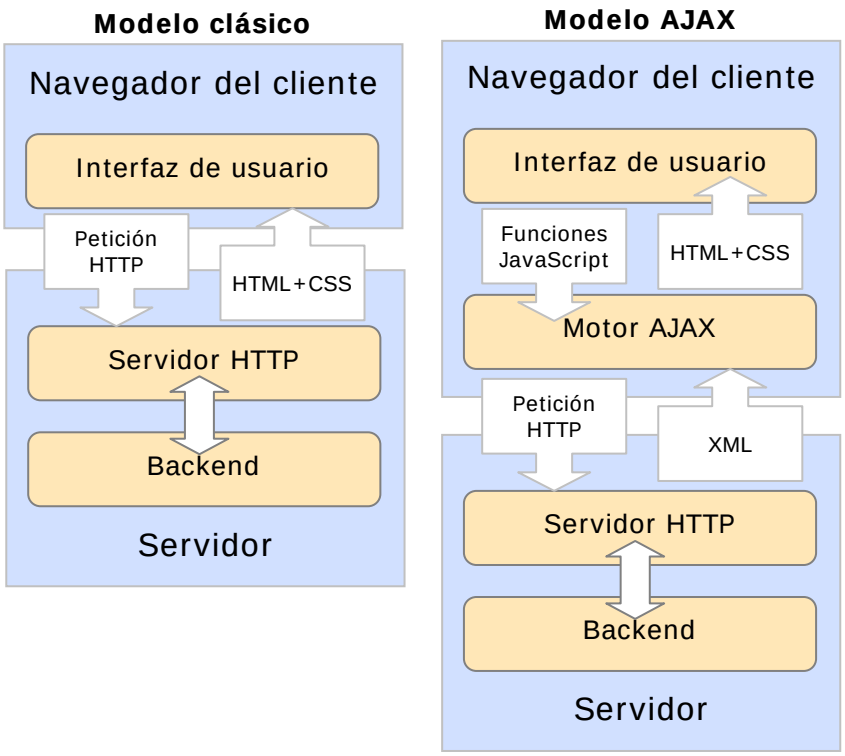


Figura 2.4. Modelo tradicional Web y modelo AJAX.

El *motor AJAX* o, conceptualmente, la nueva capa que se introduce, no es más que un conjunto de archivos *js* utilizados como librerías, que se encargan de *renderizar* la interfaz del usuario y realizar la comunicación con el servidor de manera transparente al usuario. Este motor permite establecer una interacción entre el usuario y la aplicación de forma asíncrona (figura 2.5), es decir, independientemente de la comunicación con el servidor.

Cualquier acción realizada por el usuario, en lugar de generar una petición HTTP, genera una llamada a una función JavaScript al *motor AJAX*. Si para llevarse a cabo la petición del usuario no es necesaria información del servidor o no requiere realizar una acción en el servidor, se resolverá en el propio *motor AJAX* ubicado en el cliente (validación de datos, visualización de información en memoria, etc.). Si por lo contrario, la petición requiere de la ejecución de un proceso en el servidor para generar la respuesta, el *motor AJAX* realiza las peticiones necesarias de manera asíncrona, mediante algún objeto como el *XMLHttpRequest*, sin parar la interacción con el usuario.

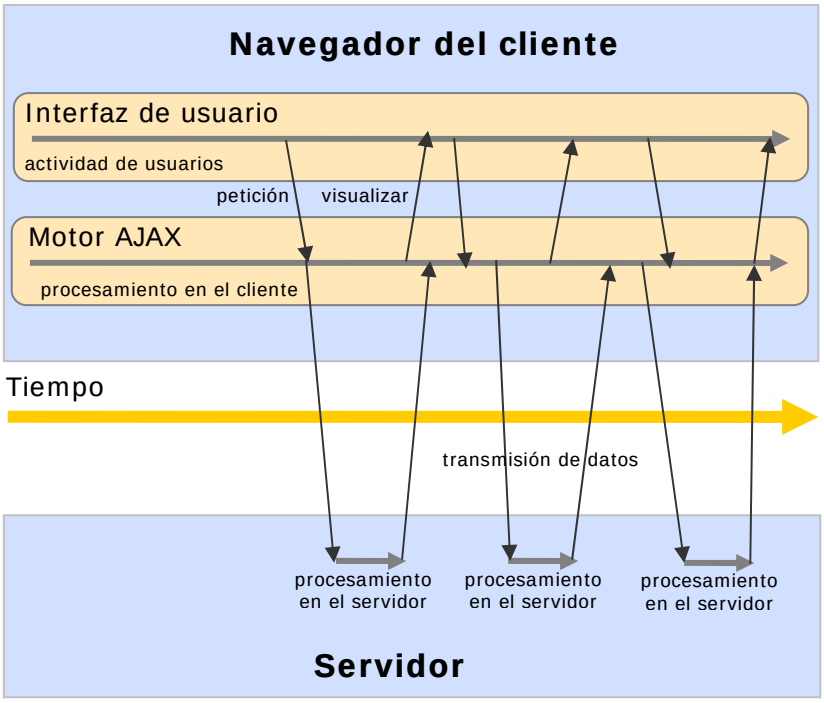


Figura 2.5. Modelo asíncrono de AJAX.

De una manera más concreta podríamos describir el funcionamiento de AJAX con los siguientes pasos:

- El usuario interactúa con la interfaz Web provocando un evento.
- El motor AJAX crea e inicializa un objeto *XMLHttpRequest*.
- El objeto *XMLHttpRequest* realiza una petición HTTP al servidor.
- El servidor procesa la petición y devuelve como resultado la información estructurada en XML.
- El objeto *XMLHttpRequest* obtiene la información como XML y se llama a la función encargada de procesarla.
- Se actualiza el DOM de la página asociada a la petición a partir del resultado devuelto por el servidor.

2.3.2. El objeto XMLHttpRequest

El objeto *XMLHttpRequest* fue desarrollado originalmente por Microsoft como un objeto ActiveX y está presente desde la versión 5 del Internet

Explorer. Este objeto ha sido implementado en otros navegadores como en Mozilla desde la versión 1.0 y en Safari desde la 1.2. Por este motivo su uso se ha extendido a navegadores ampliamente utilizados. Este objeto ofrece un API que permite realzar conexiones HTTP con el servidor el cual puede devolver la respuesta usando XML, texto plano, JavaScript o JSON. El objeto ha sido utilizado por los diferentes *lenguajes de Script* y utiliza para la comunicación un canal de conexión independiente. Este objeto permite a lenguajes como JavaScript realizar peticiones HTTP a un servidor remoto sin la necesidad de volver a cargar la página que se este visualizando. Como se comentó anteriormente con este objeto se pueden estar haciendo peticiones y recibiendo respuestas del servidor en segundo plano sin que el usuario final sea consciente de este proceso.

Un pequeño problema es, que, mientras que para utilizar el objeto desde JavaScript en un navegador IE se instancia un objeto ActiveX como se muestra en la listado 2.15, en Mozilla o Safari se trata de un objeto nativo (listado 2.16), y su forma de instanciarlo es completamente diferente. Se debe tener en cuenta este aspecto para hacer la aplicación compatible con los navegadores.

```
Var req = new ActiveXObject(“Microsoft.XMLHTTP”);
```

Listado 2.15. Instancia del objeto como ActiveX.

```
Var req = new XMLHttpRequest();
```

Listado 2.16. Instancia del objeto nativo de Mozilla o Safari.

Con motivo de esta inconsistencia en el objeto es posibles establecer la funcionalidad en JavaScript que permita indicar el objeto que se debe usar, como se muestra en el listado 2.17.

```
var req;

function loadXMLDoc(url)
{
    // Opción para objeto XMLHttpRequest nativo
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
        req.onreadystatechange = processReqChange;
        req.open("GET", url, true);
        req.send(null);

    // Opción para objeto ActiveX de IE/Windows
    } else if (window.ActiveXObject) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
        if (req) {
            req.onreadystatechange = processReqChange;
            req.open("GET", url, true);
            req.send();
        }
    }
}
```

Listado 2.17. Instancia del objeto nativo de Mozilla o Safari.

Los métodos que ofrece el API *XMLHttpRequest* son:

- *abort()*: Detiene la petición actual.
- *getAllResponseHeaders()*: Devuelve todas las cabeceras de la respuesta como pares de etiqueta y valores en una cadena.
- *getResponseHeader("headerLabel")*: Devuelve el valor de una cabecera determinada.
- *open("method", "URL"[, asyncFlag[, "userName"[, "password"]]]):* Asigna la *URL* de destino, el método y otros parámetros opcionales de una petición pendiente.
- *send(content)*: Envía la petición, opcionalmente se puede enviar una cadena de texto o un objeto DOM.
- *setRequestHeader("label", "value")*: Asigna un valor al par *label/value* para la cabecera enviada.

De la misma forma las propiedades que ofrece son:

- *onreadystatechange*: El manejador del evento llamado en cada cambio de estado del objeto. Permite preparar la recepción de la respuesta del servidor asociando el manejador a una función que tratará la respuesta.

- *readyState*: Entero que indica el estado del objeto (0 = sin inicializar, 1 = cargando, 2 = fin de la carga, 3 = actualizando la información recibida, 4 = Operación completada).
- *responseText*: Cadena de texto con los datos devueltos por el servidor.
- *responseXML*: Objeto DOM devuelto por el servidor.
- *status*: Código numérico devuelto por el servidor, ejemplos: *404* si la página no se encuentra, *200* si todo ha ido bien, etc.
- *statusText*: Mensaje que acompaña al código de estado.

A continuación se muestra un pequeño ejemplo básico de uso del AJAX con la aplicación `"Hola Mundo"`.

Lo primero que se debe hacer es crear una variable (*httpRequest*) que será la que contenga la instancia al *objeto XMLHttpRequest*. Después, como se comentó anteriormente se debe crear la parte de código que nos permita instanciar el *objeto XMLHttpRequest* en función del navegador desde el cual se solicite la página.

Una vez instanciado se debe asociar al manejador la función que realizará la lectura de los datos enviados por el servidor. Este proceso como se muestra en el listado 2.19 se realiza con el atributo o propiedad `onreadystatechange` y la función se ejecutará cada vez que la propiedad `readyState` del objeto `XMLHttpRequest` cambie de estado. Para leer estos datos se deberá comprobar que la petición se encuentre en estado 4, como se realiza en la función asociada `alertContent`. En el ejemplo simplemente se mostraría un mensaje de alerta con el contenido de la respuesta del servidor, pero en función del objetivo de la aplicación se podría manejar el resultado mediante DOM para presentar al usuario los datos finales.

```
<script type="text/javascript" language="javascript">
  function makeRequest(url) {
    var httpRequest;

    if (window.XMLHttpRequest) { // Mozilla, Safari, ...
       httpRequest = new XMLHttpRequest();
       if (httpRequest.overrideMimeType) {
          httpRequest.overrideMimeType('text/xml');
       }
    }
    else if (window.ActiveXObject) { // IE
         try {
            httpRequest = new ActiveXObject("Msxml2.XMLHTTP");
         }
         catch (e) {
            try {
               httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch (e) {}
         }
    }
    if (!httpRequest) {
         alert('Giving up :( Cannot create an XMLHTTP instance');
         return false;
    }
```

Listado 2.18. “Hola Mundo” con AJAX, parte I.

Una vez asociada la función al manejador se debe abrir la conexión con el servidor mediante los métodos GET, POST o HEAD. Para ello se debe indicar la *URL* de la que se desea obtener los datos, el tipo de modo a usar (en AJAX siempre se usará el modo asíncrono indicando como valor del parámetro `true`). Para realizar la conexión se debe llamar al método `send` del *objeto XMLHttpRequest* después de `open`.

El método `send` envía la petición con los datos pasados como parámetros como cuerpo de la petición.

En la última parte del código de ejemplo del listado 2.19 se mostraría al usuario un enlace que al pulsarlo invocaría a la función que inicia todo el proceso (`makeRequest`).

```
    httpRequest.onreadystatechange = function() {
                                      alertContents(httpRequest);
                                     };
    httpRequest.open('GET', url, true);
    httpRequest.send(null);
  }
  function alertContents(httpRequest) {

    if (httpRequest.readyState == 4) {
        if (httpRequest.status == 200) {
                alert(httpRequest.responseText);
        }
        else {
                alert('There was a problem with the request.');
        }
    }
  }
</script>
<span
    style="cursor: pointer; text-decoration: underline"
    onclick="makeRequest('holamundo.xml')">
        Make a request
</span>
```

Listado 2.19. “Hola Mundo” con AJAX, parte II.

En el listado 2.20 se muestra el documento XML que devuelve el servidor como respuesta a la petición realizada por el cliente.

```
<?xml version="1.0" ?>
<root>
      Hola mundo!!!
</root>
```

Listado 2.20. Archivo XML devuelto por el servidor (`holamundo.xml`).

2.4. APLICACIONES LIGERAS

Existe la posibilidad de extender el comportamiento y las funcionalidades de los navegadores Web. Por un lado, se pueden introducir componentes software para que se ejecuten o corran en el contexto de terceras aplicaciones, en este caso en los navegadores Web. En este primer caso el navegador debe soportar este tipo de aplicaciones de forma nativa. Estas aplicaciones ligeras no tienen sentido de manera independiente. La

programación de estos componentes es más compleja y requieren más conocimientos que el uso de los *lenguajes de Script*. Otra gran diferencia, es que estos componentes han sido compilados previamente a su utilización y lo que recibe el navegador Web es código binario. En el caso de los *Script*, es el propio navegador el que cada vez que se carga la página o se inicia una acción interpreta el código. Una de las ventajas de estos componentes es que suelen ser más potentes que los *lenguajes de Script* interpretados en el cliente puesto que se basan en lenguajes de programación de alto nivel con las características de dichos lenguajes. Pero no todos los navegadores soportan este tipo de aplicaciones de manera nativa, debido a la cantidad de aplicaciones ligeras que aparecen cada día. Por este motivo, la mayoría de los navegadores implementan una arquitectura de plug-ins o conectores, que permite añadir módulos para incorporar aplicaciones ligeras.

2.4.1. Applets

Un *Applet* es un programa escrito en lenguaje Java que se puede incluir en una página HTML de la misma manera que incluimos una imagen. Cuando se solicita desde un navegador una página Web, esta puede contener una referencia a un Applet. Cuando el navegador esté interpretando la página HTML llegará hasta la etiqueta que incluye el Applet y posteriormente le solicita al servidor la transmisión del código binario del Applet. Una vez el código es transferido al cliente, es posible ejecutarlo en el contexto de la máquina virtual de java asociada al navegador, pero en el lado del cliente (generalmente con la ayuda de un plug-in). Generalmente, los principales inconvenientes del uso de los Applets en las aplicaciones Web son: por un lado, el tiempo que se requiere para la transmisión del Applet y que dependerá de la línea que se posea y por otro lado, que el navegador se haya habilitado para soportar aplicaciones java, es decir que se incluya la máquina virtual de java para poder ejecutar el Applet.

Como desventajas, en relación con JavaScript, cabe señalar que los Applets son más lentos de procesar y que tienen espacio muy delimitado en la página donde se ejecutan, es decir, no se mezclan con todos los componentes de la página ni tienen acceso a ellos. Por este motivo, con los Applets de Java no se puede, directamente, hacer cosas como abrir ventanas secundarias, controlar Frames, formularios, capas, etc, pero si ampliar la funcionalidad de nuestro navegador e incluso acceder a componentes del lado del servidor.

La principal ventaja de utilizar Applets consiste en que son mucho menos dependientes del navegador que los S*cripts* en JavaScript, incluso independientes del sistema operativo del ordenador donde se ejecutan. Además, Java es más potente que JavaScript, por lo que las aplicaciones de los Applets podrán ser mayores.

En la figura 2.21 se puede ver el código para crear un sencillo Applet. Este tipo de Applet que extiende de la clase `java.applet.Applet` se utiliza cuando no vamos hacer uso de componentes *GUI* de tipo *Swing* y utiliza únicamente los componentes *AWT*. Pero esta opción comienza a estar en desuso y se utiliza la clase `javax.swing.JApplet` la cual permite introducir en el Applet componentes *GUI de tipo Swing*. El único problema con este último tipo de applets es que el plug-in instalado debe soportar *Swing*.

```
import java.applet.*;
import java.awt.*;
public class HolaMundo extends Applet {
      public void paint (Graphics g) {
             g.drawString ("¡¡Hola Mundo!!",10,80);
      }
}
```

Listado 2.21. Ejemplo de un Applet sencillo.

```
<html>
  <head>
    <title> Ejemplo simple de applet </title>
  </head>
  <body>
    <p>A continuación está la salida del programa</p>
    <applet code="Clock.class" width=170 height=150>
      <param name=bgcolor value="000000">
      <param name=fgcolor1 value="ff0000">
      <param name=fgcolor2 value="ff00ff">
    </applet>
    No hay disponible un intérprete de Java
</body>
</html>
```

Listado 2.22. Inserción de un Applet en un documento HTML.

2.4.1.1. Ciclo de vida de un Applet

Mediante la invocación de ciertos métodos, un navegador gestiona el ciclo de vida de un Applet, si el applet es cargado en una página Web.

El ciclo de vida de un Applet básicamente se compone de cuatro pasos relacionados con un método cada uno.

- El método `init` es utilizado para realizar los procesos de inicialización del Applet, si es necesario. Este método es invocado después de que el navegador lea el atributo `param` de la etiqueta `<applet>`.
- El método `start` es automáticamente invocado por el navegador después del método `init`. También se invoca a este método siempre que el usuario regrese a la página que lo contiene después de haber navegado por otras páginas.
- El método stop es invocado de forma automática siempre y cuando el usuario abandone la página que contiene el Applet.
- El método destroy es invocado cuando el navegador es cerrado siguiendo el cauce habitual.

De esta forma, el Applet puede ser inicializado una única vez, arrancado y parado una o más veces en su ciclo de vida y destruido, también, una única vez.

La etiqueta `<APPLET>` de arriba especifica que el navegador debería cargar la clase cuyo código compilado está en el fichero llamado `HelloWorld.class`. El navegador busca este fichero en el mismo directorio que contiene el documento HTML que contiene la etiqueta.

Cuando el navegador encuentra el fichero de la clase, la carga a través de la red, si es necesario, en el ordenador donde se está ejecutando el navegador. Entonces el navegador crea un ejemplar de la clase. Si incluimos un applet dos veces en la misma página, el navegador sólo cargará el Applet una vez y creará dos ejemplares de esa clase.

Cuando un navegador que soporte Java encuentra una etiqueta `<APPLET>`, reserva un área de pantalla de la anchura y altura especificadas, carga los *bytecodes* de la subclase Applet especificada, crea un ejemplar de la subclase y luego llama a los métodos `init` y `start` del ejemplar.

Incluimos los Applets en páginas HTML usando la etiqueta `<APPLET>`. Cuando un navegador visita una página que contiene un Applet suceden los siguientes pasos:

- El navegador busca el archivo *.class* de la subclase del Applet.

- La localización del archivo *.class* (que contiene los *bytecodes* Java) se especifica con los atributos `CODE` y `CODEBASE` de la etiqueta `<APPLET>`.
- El navegador descarga los *bytecodes* a través de la red hasta el ordenador del usuario.
- El navegador crea un ejemplar de la subclase Applet (cuando nos referimos a un Applet, generalmente nos referimos a este ejemplar).
- El navegador llama al método `init` del Applet (este método realiza cualquier inicialización que sea necesaria).
- El navegador llama al método `start` del Applet (este método normalmente arranca un *thread* que realiza las tareas del Applet).

Una subclase Applet es la clase principal, la clase controladora, pero los Applets también pueden usar otras clases. Estas otras clases pueden ser locales del navegador, proporcionadas como parte del entorno Java, o ser clases personalizadas que el usuario suministra. Cuando el Applet intenta utilizar una clase por primera vez, el navegador intenta encontrarla en el host en el que se está ejecutando. Si no puede encontrarla allí, busca la clase en el mismo lugar de donde vino la subclase de Applet. Cuando el navegador encuentra la clase, carga sus *bytecodes* (a través de la red, si es necesario) y continúa ejecutando el Applet.

Cargar código ejecutable a través de la red es un riesgo de seguridad clásico. Para los Applets Java, este riesgo se reduce porque el lenguaje Java está diseñado para ser seguro (por ejemplo, no permite punteros a memoria). Además, los navegadores compatibles Java mejoran la seguridad imponiendo sus propias restricciones. Estas restricciones incluyen no permitir a los Applets que carguen código escrito en otros lenguajes distintos de Java, y no permitiendo que los Applets lean o escriban ficheros en el *host* del navegador.

2.4.1.2. Instalación de Java plug-in en navegadores

Existen pequeños navegadores, como el *appletviewer* proporcionado por SUN en la máquina virtual de java JDK 1.2 que ya incorpora la compatibilidad con los Applets.

En el resto de navegadores independientes de Java, para poder visualizar un Applet en un navegador tipo IE o Mozilla Firefox es necesario instalar el plug-in de Java. El *Java plug-in* es incluido como parte del entorno de ejecución de Java, JRE, y permite establecer una conexión entre los navegadores más populares y la plataforma Java. Este permite a los Applets ejecutarse en el contexto del navegador. Cargar el *Java plug-in* es un proceso muy sencillo que únicamente requiere instalar la máquina

virtual de java JRE. Automáticamente se podrán visualizar los Applets en nuestro navegador.

Una vez instalado tenemos que asegurarnos que está habilitada en el navegador la opción para permitir usar el JRE y visualizar Applets.

En el navegador IE debemos ir al menú de *herramientas* → *opciones de Internet* → pestaña de *opciones avanzadas* y asegurarnos que en el *check box Java (Sun)* este marcado, de lo contrario no podremos visualizar el Applet.

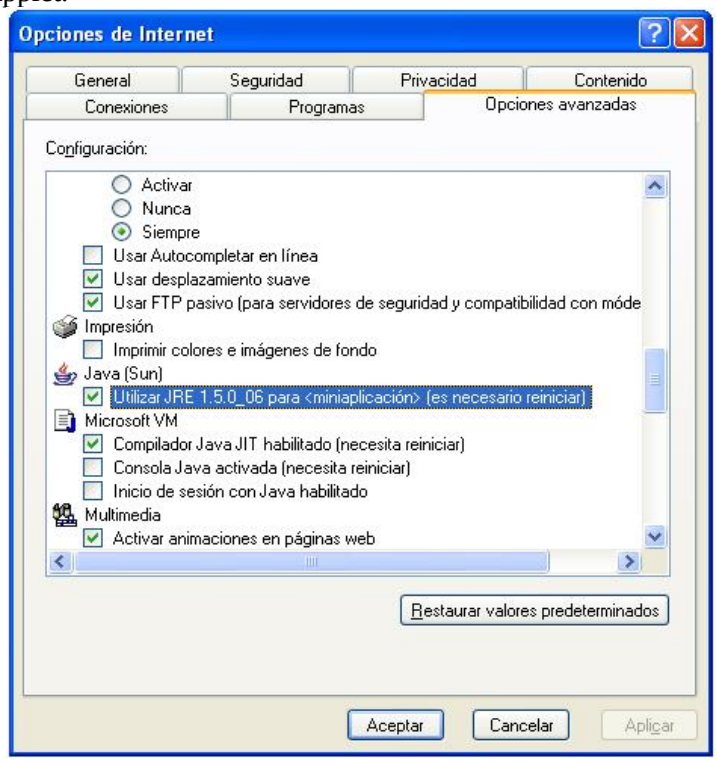


Figura 2.6. Habilitar en IIS el uso de Applets.

En el navegador Mozilla Firefox se requiere un proceso similar. En el menú *Herramientas* → *opciones* → *pestaña contenido* activamos la casilla cuya inscripción dice *Activar Java*. Con esto se obtiene el mismo resultado que en el proceso del IE.

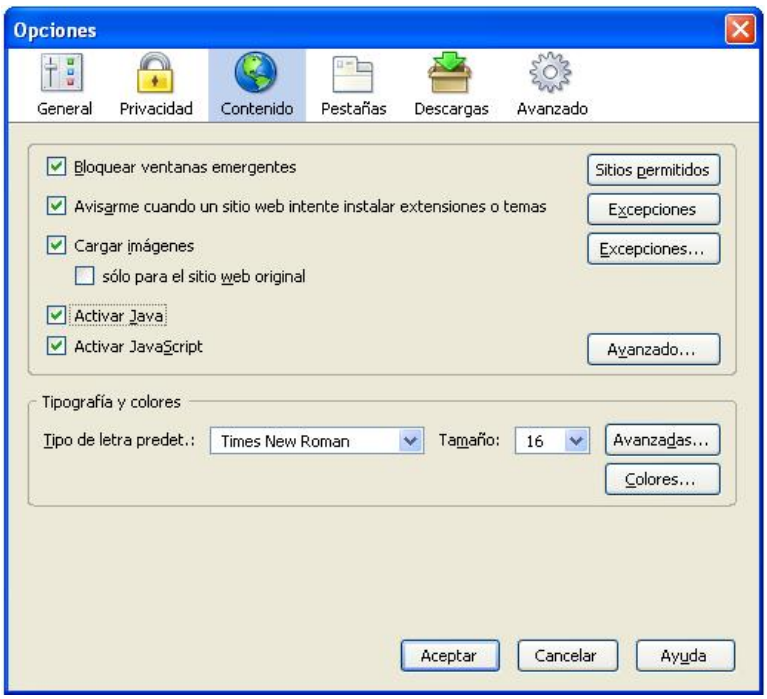


Figura 2.7. Habilitar en Firefox el uso de Applets.

El panel de control de java, el cual se puede encontrar en el panel de control del sistema operativo permite configurar y ver un amplio rango de parámetros de configuración para controlar las características de la máquina virtual de java en el propio entorno. Por tanto controlar también la configuración del plug-in. Para más información se puede acceder a la *URL* `http://java.sun.com/j2se/1.5.0/docs/guide/deployment/deployment-guide/jcp.html`.

2.4.2. ActiveX

Tecnología utilizada, entre otras cosas, para dotar a las páginas Web de mayores funcionalidades, como animaciones, vídeo, navegación tridimensional, etc. Los controles ActiveX son pequeños programas que se incluyen dentro de estas páginas. Lamentablemente, por ser programas, pueden ser el objetivo de algún virus.

ActiveX es una tecnología de Microsoft para el desarrollo de páginas dinámicas. Tiene presencia en la programación del lado del servidor y del lado del cliente, aunque existan diferencias en el uso en cada uno de esos dos casos.

Son pequeños programas que se pueden incluir dentro de páginas Web y sirven para realizar acciones de diversa índole. Por ejemplo hay controles ActiveX para mostrar un calendario, para implementar un sistema de FTP, etc.

Son un poco parecidos a los Applets de Java en su funcionamiento, aunque una diferencia fundamental es la seguridad, pues un Applet de Java no podrá tomar privilegios para realizar acciones malignas (como borrarnos el disco duro) y los controles ActiveX sí que pueden otorgarse permisos para hacer cualquier cosa.

Los controles ActiveX son particulares de Internet Explorer aunque existen plug-ins específicos para poder ejecutar controles ActiveX dentro del contexto de otros navegadores.

Los controles ActiveX son componentes de software que se pueden descargar y que se utilizan para ampliar las funciones de Internet Explorer a través de elementos de la interfaz de usuario, como menús emergentes, botones.

Mientras se explora la Web se tiene protección ante la descarga de controles ActiveX no deseados, ya que si una página intenta descargar uno de ellos, Internet Explorer le mostrará un certificado firmado con el nombre de la compañía que creó el control. Este certificado aparece como un cuadro de diálogo de advertencia de seguridad. Si se encuentra en un sitio Web de su confianza (en este caso, Windows Update) y el control ActiveX lo proporciona una compañía en la que confía (en este caso, Microsoft), puede aceptar el certificado y permitir la instalación del control.

Aunque en principio los controles ActiveX fueron creados para utilizarse con Internet Explorer, posteriormente se han creado plug-ins para otros navegadores de uso común como Netscape y Mozilla Firefox.

Uno de los problemas a diferencia de los Applets es que los controles ActiveX tienen acceso completo al sistema operativo con el peligro que ello puede conllevar. Para controlar esta falla Microsoft creó un sistema de registro para que los navegadores puedan identificar y autenticar un control ActiveX antes de descargarlo y ejecutarlo.

```
<html>
  <head>
    <title> Ejemplo simple de activeX </title>
  </head>
  <body>
    <object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
width="160" height="144"
codebase="http://www.apple.com/qtactivex/qtplugin.cab">
      <param name="SRC" value="sample.mov">
      <param name="AUTOPLAY" value="true">
      <param name="CONTROLLER" value="false">
    </object>
  </body>
</html>
```

Listado 2.23. Inserción de ActiveX en un documento HTML.

En el listado 2.23. se muestra un ejemplo para incluir un control ActiveX dentro de un documento HTML (el control ActiveX para Quicktime). El atributo `classid` identifica de forma única que control ActiveX se va a usar. Un atributo `classid` con el valor `clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B` indica al navegador Internet Explorer que use el control ActiveX para Quicktime. El desarrollador debe conocer este valor. El atributo `codebase` indica la localización del control ActiveX. El usuario hará uso de este valor únicamente si no tienen instalado el control de forma que el navegador accederá a la ubicación y descargará el control. El navegador Internet Explorer automáticamente se ofrecerá para descargar e instalar el control.

2.4.2.1. Configuración de Internet Explorer con ActiveX

Ya se ha comentado anteriormente los peligros que puede conllevar permitir la ejecución de controles ActiveX sin control. En el navegador Internet Explorer por defecto las opciones de descarga y ejecución de ActiveX se encuentran deshabilitadas dejando el control al usuario si desea descargar y ejecutar los controles. En función del nivel de seguridad que tengamos asignado o definido se actuará de diversas formas.

La opción `personalizar`, da un mayor control a los usuarios avanzados y a los administradores sobre todas las opciones de seguridad. Por ejemplo, la opción `descargar` los controles ActiveX sin firmar se deshabilita de forma predeterminada en la *zona Intranet local* (la seguridad media es la configuración predeterminada de la *zona Intranet local*). En este caso, Internet Explorer no puede ejecutar ningún control de ActiveX en la intranet de su organización porque la mayoría de las organizaciones no

firman los controles ActiveX que sólo se utilizan internamente. Para que Internet Explorer ejecute los controles ActiveX sin firmar en la intranet de una organización, cambie en el nivel de seguridad de la *opción Descargar*, los controles ActiveX sin firmar a *Preguntar* o *Habilitar* para la *zona Intranet local*. En las opciones de seguridad en la opción *Nivel personalizado* se puede establecer el acceso a los archivos, controles ActiveX y secuencias de comandos.

Figura 2.8. Opciones de seguridad para controles ActiveX en IE.

2.4.2.2. Plug-in de ActiveX para Mozilla.

El navegador Mozilla no soporta controles ActiveX de forma nativa. Se ha desarrollado un plug-in, el cual también trabaja con Netscape 4.x, que permite utilizar controles ActiveX en estos navegadores. Este plug-in se ha creado con ciertas limitaciones puesto que no permite descargar controles e instalarlos de forma automática. Solamente permite ejecutar controles que ya se encuentran instalados e identificados como seguro para ser incrustado.

Se puede obtener más información en el *Mozilla ActiveX Project* en la *URL* `http://www.iol.ie/~locka/mozilla/mozilla.htm`.

Se debe tener en cuenta que los controles ActiveX pueden suponer un riesgo para nuestro equipo.

A continuación se describe el proceso de instalación del plug-in para *Mozilla Firefox 1.5*. Se debe tener en cuenta que el plug-in se realiza para versiones específicas del navegador y que podría acarrear daños si se utiliza con versiones no adecuadas (se debería realizar una desinstalación del plug-in).

Se puede descargar el plug-in para la versión nombrada anteriormente en:

`http://www.iol.ie/~locka/mozilla/plugin.htm#download`

Para ello se debe descargar el archivo con extensión *xpi* y ejecutamos el navegador Firefox. En el menú *archivo* elegimos *abrir archivo* y seleccionamos el archivo *xpi* descargado (en este caso *mozactivex-ff-15.xpi*). Una vez seleccionado y abierto se no ofrece la opción de *instalar el plug-in*.

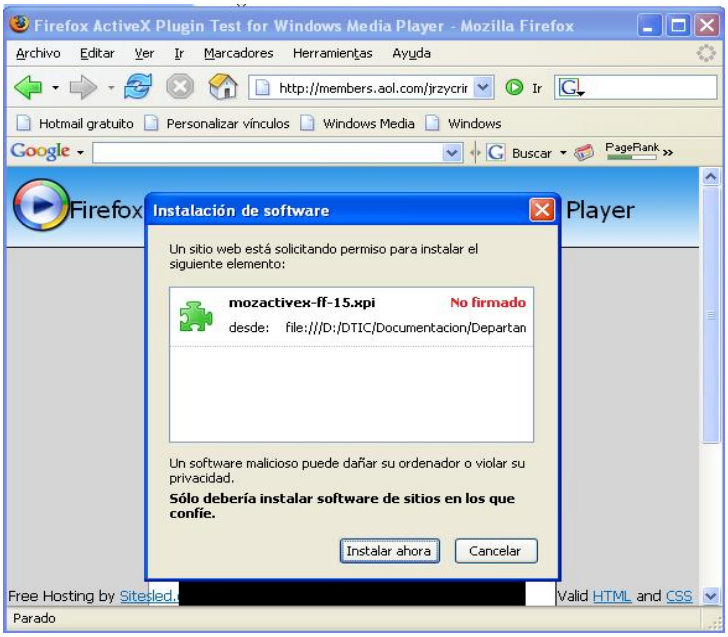


Figura 2.9. Opciones de seguridad para controles ActiveX en Mozilla.

Una vez instalado debemos cerrar el navegador y volverlo a ejecutar. Si no ha habido ningún problema se pueden visualizar controles ActiveX en este navegador.

Se puede realizar una prueba accediendo a la *URL* `http://members.aol.com/jrzycrim01/mozilla/wmp/wmpaxtest.html` y confirmar si se pueden ver los controles ActiveX que instancia el documento HTML descargado.

Para desinstalar el plug-in se deben seguir los siguientes pasos:

- Cerrar todas las instancias del navegador.
- Ir al directorio del *Mozilla Firefox*.
- Ir al directorio *Mozilla Firefox\plugins* y borrar el archivo `npmozax.dll`.
- Volver al directorio del *Mozilla Firefox* e ir al directorio *\components* para borrar los archivos `nsIMozAxPlugin.xpt` y `nsAxSecurityPolicy.js`.
- Acceder a la ruta *Mozilla Firefox\defaults\pref* y borrar el archivo `activex.js`.

2.4.3. Extensión del navegador mediante plug-ins

Los plug-ins son aplicaciones o módulos externos al navegador que permiten extender su funcionalidad para poder ejecutar mini-aplicaciones que doten de mayor potencia al navegador sin que el navegador tenga que soportarlas de una forma nativa.

Los plug-ins van a permitir que el navegador actúe como contenedor de otras aplicaciones gestionando su ciclo de vida. Esta técnica surge ante la imposibilidad de que un navegador Web sea capaz de procesar todos los tipos de datos que puede enviar como respuesta un servidor Web.

Un plug-in es cargado por el navegador si los datos enviados por el servidor son del tipo asociado al plug-in (la asociación se realiza mediante el tipo MIME).

2.4.3.1. Funcionamiento de los plug-ins

Se ha dividido la forma de actuar del navegador en referencia a los plug-ins en dos categorías. La primera en la cual se recibe un flujo de datos el cual requiere de la actuación de un plug-in directamente y no está insertado en código HTML. La segunda consiste en la inserción de objetos que requieren el uso de un plug-in por parte del navegador en código HTML.

En el primer caso, el cliente hace una petición mediante una *URL* a un servidor Web (`http://localhost/prueba.pdf`).

El servidor responde al cliente con el contenido mostrado en el listado 2.24. Cuando el navegador recibe el flujo de datos analiza la

cabecera (Content-Type) y dependiendo del tipo MIME que se indica el navegador invocará al plug-in correspondiente para mostrar el contenido. En este caso se ejecutaría el plug-in de *Adobe Acrobat Reader*.

```
HTTP/1.1 200 OK
Date: Mon, 30 Oct 2006 20:17:17 GMT
Server: Apache/2.2.3 (Win32)
Last-Modified: Mon, 30 Oct 2006 19:52:12 GMT
ETag: "132f6-16e3-9b217560"
Accept-Ranges: bytes
Content-Length: 5859
Connection: close
Content-Type: application/pdf

%ÔÒ¤Ë1.4 /L 5859/O 8/E 1750/N 1/T 5693/H [ 476 149]>>
xref
6 9
0000000016 00000 n
0000000625 00000 n
0000000701 00000 n
0000000833 00000 n
0000000916 00000 n
trailer
----8CFDA75A284D5A8033E016C87CBCE897—
......
......
```

Listado 2.24. Contenido de un archivo *pdf*.

El segundo caso se produce cuando se insertan objetos dentro de un documento HTML. En la especificación de HTML 4.01 se ha definido el elemento `Object` como mecanismo para invocar a los plug-ins. Este elemento es usado de forma diferente en el navegador Internet Explorer que en los navegadores basados en Mozilla. Para conocer más sobre este elemento se puede consultar la especificación de HTML 4.01 en `http://www.w3.org/TR/1999/PR-html40-19990824/`.

```
<!ELEMENT OBJECT - - (PARAM | %flow;)*

<!ATTLIST OBJECT
%attrs;
declare     (declare)      #IMPLIED
classid     %URI;          #IMPLIED
codebase    %URI;          #IMPLIED
data        %URI;          #IMPLIED
type        %ContentType;  #IMPLIED
codetype    %ContentType;  #IMPLIED
archive     %URI;          #IMPLIED
standby     %Text;         #IMPLIED
height      %Length;       #IMPLIED
width       %Length;       #IMPLIED
usemap      %URI;          #IMPLIED
name        CDATA          #IMPLIED
tabindex    NUMBER         #IMPLIED>
```

Listado 2.25. DTD del elemento object.

En primer lugar, el navegador Internet Explorer invoca a un plug-in creado como ActiveX. Esto implica que se debe indicar el identificador del ActiveX dentro del elemento `object` (atributo `classid`). Esto implica que el programador debe conocer el identificador único de cada plug-in. El atributo codebase usado apunta a la localización donde está el archivo *CAB* que contiene el control del ActiveX que actúa como plug-in. En este contexto, el atributo codebase se usa como *mecanismo de obtención*, lo cual quiere decir, que se trata de una forma de obtener el controlador si no esta presente. Por ejemplo, si el control de ActiveX de Flash no está instalado, IE irá entonces a la *URL* indicada en el atributo `codebase` y obtendrá el control de ActiveX que permita visualizar la película.

Los atributos `param` especifican los parámetros de configuración para el plug-in.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/s
wflash.cab#version=5,0,0,0"
width="366" height="142" id="myFlash">
    <param name="movie" value="javascript-to-flash.swf" />
    <param name="quality" value="high" />
    <param name="swliveconnect" value="true" />
</object>
```

Listado 2.26. Plugins en Internet Explorer.

Los navegadores basados en Mozilla soportan la arquitectura de plug-in de Netscape, los cuales no están basados en COM como el ActiveX (y por ello, no son llamados vía identificador único) si no, basados en el tipo MIME. Cuando el navegador interpreta un documento HTML y se encuentra con un elemento `object` este hace uso del atributo `type` el cual contiene el tipo MIME que identifica el tipo de objeto y por tanto le indica al navegador el plug-in al cual debe invocar. Puede suceder que el plug-in no esté instalado y se nos ofrezca un mensaje indicando la ubicación para descargarlo o que el propio navegador la conozca e indique si la queremos instalar.

```
<object type="application/x-shockwave-flash" data="javascript-to-
flash.swf"
width="366" height="142" id="myFlash">
    <param name="movie" value="javascript-to-flash.swf" />
    <param name="quality" value="high" />
    <param name="swliveconnect" value="true" />
    <p>You need Flash -- get the latest version from
    <a href= "http://www.macromedia.com/downloads/">here.</a></p>
</object>
```

Listado 2.27. Plug-ins en Firefox.

Para usar el plug-in de java en los navegadores e introducir Applets en los documentos HTML se sigue utilizando en la mayoría de las aplicaciones la etiqueta `<APPLET>` en lugar de `<OBJECT>` aunque en la especificación de HTML 4.1 se ha desechado esta opción y se recomienda usar el elemento `Object`.

2.5. MANTENIMIENTO DE LA SESIÓN: COOKIES

Puesto que HTTP es un *protocolo petición/respuesta* la peticiones son tratadas de manera individual. Las aplicaciones Web necesitan un mecanismo que les permita identificar un determinado cliente y el estado de cualquier conversación que se mantiene con los clientes. Una sesión es una corta secuencia de peticiones de servicio realizadas por un único usuario por medio de un único cliente para acceder al servidor. El estado de la sesión es la información mantenida en la sesión a través de las peticiones.

Una de las técnicas para el almacenamiento de esta información ha sido las denominadas *Cookies*. Se trata de una potente herramienta empleada por los servidores Web para almacenar y recuperar información acerca de sus visitantes.

Las *Cookies* son pequeñas porciones de información que se almacena en el disco duro del visitante de una página web a través de su navegador, a petición del servidor de la página. Esta información puede ser luego recuperada por el servidor en posteriores visitas. Al ser el protocolo HTTP incapaz de mantener información por sí mismo, para que se pueda conservar información entre una página vista y otra (como *login* de usuario, preferencias de colores, etc.), ésta debe ser almacenada, ya sea en la *URL* de la página, en el propio servidor, o en una *Cookie* en el ordenador del visitante.

Una *Cookie* no es más que un archivo de texto que algunos servidores piden a nuestro navegador que escriba en nuestro disco duro, con información acerca de lo que hemos estado haciendo por sus páginas.

Entre las mayores ventajas de las C*ookies* se cuenta el hecho de ser almacenadas en el disco duro del usuario, liberando así al servidor de una importante sobrecarga. Es el propio cliente el que almacena la información y quien se la devolverá posteriormente al servidor cuando éste la solicite. Además, las cookies poseen una fecha de caducidad, que puede oscilar desde el tiempo que dure la sesión hasta una fecha futura especificada, a partir de la cual dejan de ser operativas.

Entre las tareas que realiza una *Cookie* podemos destacar:

- Llevar el control de usuarios: cuando un usuario introduce su nombre de usuario y contraseña, se almacena una *Cookie* para que no tenga que estar introduciéndolas para cada página del servidor. Sin embargo, una *Cookie* no identifica a una persona, sino a una combinación de computador y navegador.
- Ofrecer opciones de diseño (colores, fondos, etc.) o de contenidos al visitante.

- Conseguir información sobre los hábitos de navegación del usuario, e intentos de *spyware*, por parte de agencias de publicidad y otros. Esto puede causar problemas de privacidad y es una de las razones por la que las *Cookies* tienen sus detractores.

2.5.1. Funcionamiento de las Cookies

El protocolo HTTP ha creado una serie de cabeceras que permite realizar operaciones con *Cookies* (crear, obtener *Cookies*, cambiar el tiempo de expiración, borrar, etc.).

Cuando un cliente realiza una petición a un servidor Web, este como respuesta puede enviar información en la cabecera del documento HTTP para la creación de *Cookies* en el cliente. Esta operación se realiza mediante la directiva de cabecera `Set-Cookie`.

Es el servidor el que inicia la sesión al responder al cliente con un mensaje que tiene una cabecera para establecer una *Cookie* (`Set-Cookie`). Cuando el cliente recibe esta respuesta, en la siguiente petición incorporará una cabecera para informar de las *Cookies* que tiene (`Cookie`). Ante esta petición puede tener en cuenta la *Cookie* o no para dar la respuesta, y para ello puede establecer el mismo valor u otro valor para la *Cookie* o no enviar ninguna `cookie`. Para acabar la sesión, el servidor debe establecer una *Cookie* con duración "0".

El servidor puede tener varias cabeceras para establecer varias *Cookies* en una misma respuesta, aunque si el mensaje pasa por un proxy o una pasarela, pueden convertirlo en una sola cabecera para establecer *Cookies*. Para establecer una *Cookie* se usa la cabecera `Set-Cookie`, cuya sintaxis es:

```
Set-Cookie: nombre=valor *[;modificador]
```

Con esto se establece que por lo menos ha de haber un par nombre/valor y cada par puede tener o no modificadores.

Los modificadores de un par nombre, valor pueden ser:

- `Comment = valor`, el servidor informa del uso de la *Cookie*.
- `Domain = valor`, indica el dominio en el que la *Cookie* es válida.
- `Max-Age = valor`, validez de la *Cookie* en segundos, después se descarta.
- `Path = valor`, indica el subconjunto de *URL* a las que afecta la *Cookie*.
- `Secure`, indica que el cliente debe usar en entornos seguros para contactar con el servidor.

- `Version = valor`, la versión de la especificación de mantenimiento de estado que es.

Se pueden enviar varias *Cookies* en una misma cabecera separando los pares nombre, valor con “,”.

Además de enviar la *Cookie*, el servidor puede establecer las condiciones para que la *Cookie* se ponga en una *caché* o no.

Un ejemplo de la cabecera de una respuesta:

```
Set-Cookie: NOMBRE=VALOR; expires=FECHA; path=PATH;
domain=NOMBRE_DOMINIO; secure
Set-Cookie: ID=789; path=/
Set-Cookie: NOMBRE=Juan; path=/
……
<html>
……
</html>
```

Listado 2.28. Ejemplo de respuesta del servidor.

El cliente trata por separado las distintas informaciones de estado que le llegan por medio de respuestas que establecen *Cookies*. Además toma valores por defecto para los atributos que no están presentes de los pares nombre/valor.

Además el cliente puede rechazar una *Cookie* por razones de seguridad o violaciones de la privacidad. Para rechazar una *Cookie* tiene una serie de reglas que aplica en cada caso.

Si el cliente recibe una *Cookie* con un nombre que ya tiene asociado a otra *Cookie* y cuyos atributos `Domain` y `Path` son exactamente iguales, entonces la nueva *cookie* reemplaza a la anterior. Además si la nueva *Cookie* tiene el atributo `Max-Age` con un valor de 0, entonces elimina la *Cookie* en vez de reemplazarla.

Como el cliente tiene un espacio limitado para las *Cookies*, ha de ir eliminando *Cookies* antiguas para hacer sitio a las nuevas. Esta operación la puede realizar siguiendo algún algoritmo como el *LRU* (*Least Recently Used*), o *FIFO* (*First In First Out*).

Si una *Cookie* tiene el atributo `Comment`, entonces el cliente ha de almacenar el comentario para que el usuario (humano) pueda leerlo.

Cuando el cliente realiza una petición al servidor, lo hace por medio de una *URI*. Para cada petición, además del mensaje envía una cabecera `Cookie` para informar al servidor de las *cookies* que tiene y que afectan a esa petición. La sintaxis para esta cabecera es la siguiente:

```
Cookie: versión *((;|,)nombre = valor
                                  [;path] [;dominio])
```

Donde versión es:

```
Version = valor
```

path es:

```
Path = valor
```

Y dominio es:

```
Domain = valor
```

El valor de la versión es el del atributo Versión si alguna de las *Cookies* lo impone, si no es 0. El valor de `path` ha de ser el impuesto por las *cookies* si alguna lo establece, si no se omite. El caso del dominio se trata igual que el del `path`.

En la lista de *Cookies* aparecen las que satisfacen los criterios:

- El nombre del servidor ha de tener el mismo dominio que la *Cookie*.
- El `path` de la *Cookie* ha de ser un prefijo de la *URI* que representa la petición.
- El límite de tiempo de la *cookie* no ha sido sobrepasado.

Cuando el servidor recibe una petición con una cabecera *Cookie,* ha de interpretar los pares nombre/valor teniendo en cuenta que los nombres que comienzan por el símbolo "$" son especiales y hacen referencia a la *Cookie* anterior, y si no hay una *Cookie* anterior, hacen referencia a todas las *Cookies* de esta cabecera.

En la parte del cliente se ha establecido la cabecera *Cookie* que permite informar de las *Cookies* existentes.

En la cabecera de la siguiente solicitud:

```
NOMBRE1=VALOR1; NOMBRE2=VALOR2; …
Cookie: ID=789; NOMBRE=JuanBla, bla, ….
……
<html>
……
</html>
```

Listado 2.29. Ejemplo de petición del cliente.

2.5.2. Habilitación de las Cookies

La mayor parte de los navegadores modernos soportan *Cookies*. Sin embargo, un usuario puede normalmente elegir si las *Cookies* deberían ser utilizadas o no. A continuación, las opciones más comunes:

- Las *Cookies* no se aceptan nunca.
- El navegador pregunta al usuario si se debe aceptar cada *Cookies*.
- Las *Cookies* se aceptan siempre.

El navegador también puede incluir la posibilidad de especificar mejor qué *Cookies* tienen que ser aceptadas y cuáles no. En concreto, el usuario puede normalmente aceptar alguna de las siguientes opciones: rechazar las *Cookies* de determinados dominios; rechazar las *Cookies* de terceros; aceptar *Cookies* como no persistentes (se eliminan cuando el navegador se cierra); permitir al servidor crear *Cookies* para un dominio diferente. Además, los navegadores pueden también permitir a los usuarios ver y borrar *Cookies* individualmente.

La mayoría de los navegadores que soportan JavaScript permiten a los usuarios ver las *Cookies* que están activas en una determinada página escribiendo `javascript:alert("Cookies: "+document.cookie)` en el campo de dirección.

En el navegador Mozilla Firefox podemos configurar los parámetros para permitir el almacenamiento de *Cookies*. Accediendo al menú de *herramientas* → *opciones* → *pestaña de privacidad* y en la *sub-pestaña de cookies* podemos configurar todas las posibilidades.

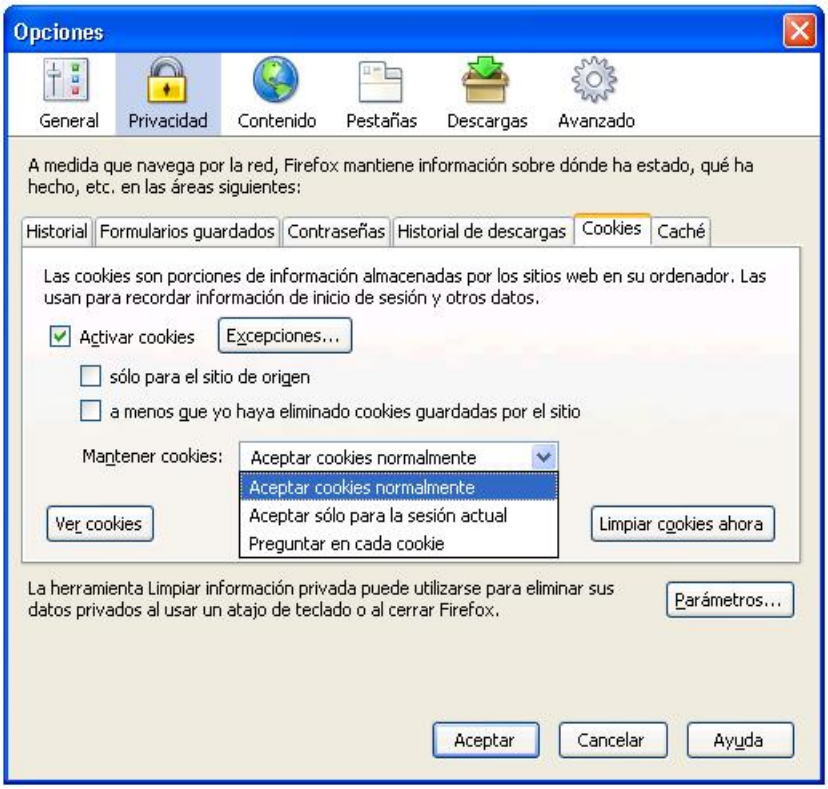


Figura 2.10. Habilitar en Firefox para el uso de *Cookies*.

Para configurar el almacenamiento de *Cookies* en el navegador Internet Explorer debemos realizar los siguientes pasos. Ir al menú de *herramientas* → *opciones de Internet* → *pestaña de privacidad* → *opciones avanzadas*. Ahora se pueden configurar las opciones de las *Cookies* en el cliente.

Si utilizamos en navegador Mozilla Firefox las *Cookies* son almacenadas en un único archivo llamado `cookies.txt` en `%HOME-PATH%\Datos de programa\Mozilla\Firefox\Profiles\hjpv0b0b.default.`

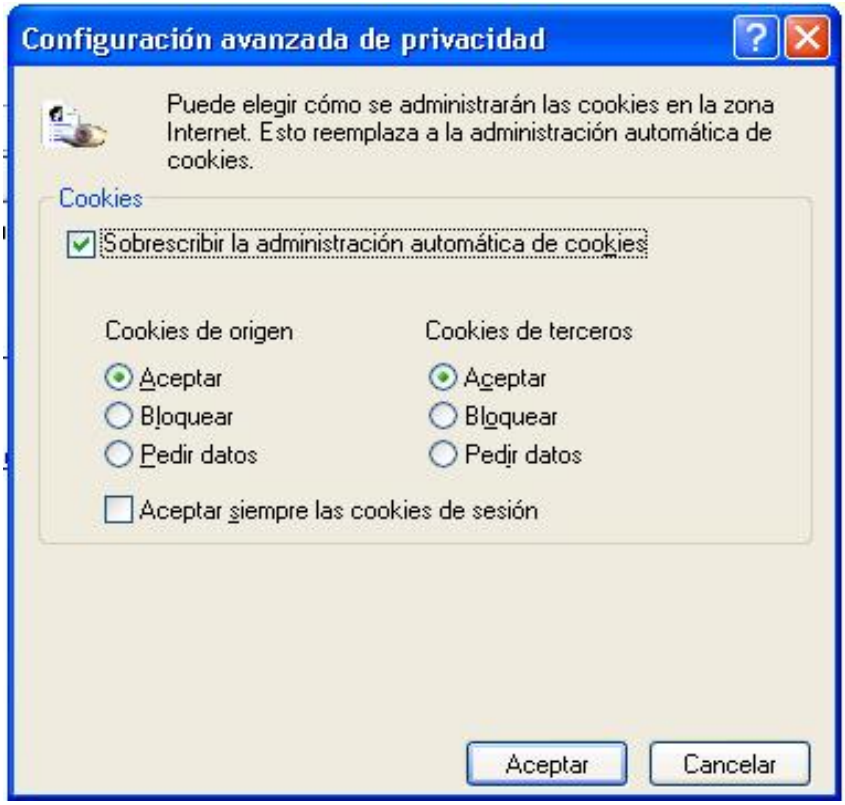


Figura 2.11. Habilitar en IE para el uso de *Cookies*.

Las cookies en Windows, utilizando el navegador Internet Explorer, normalmente se almacenan en `%HOMEPATH%/cookies`. En este directorio se pueden encontrar un elevado número de archivos con la información mantenida con cada sesión que se ha establecido con el servidor.

IE ofrece también una interfaz menos completa que solo permite eliminar todas las *Cookies*.

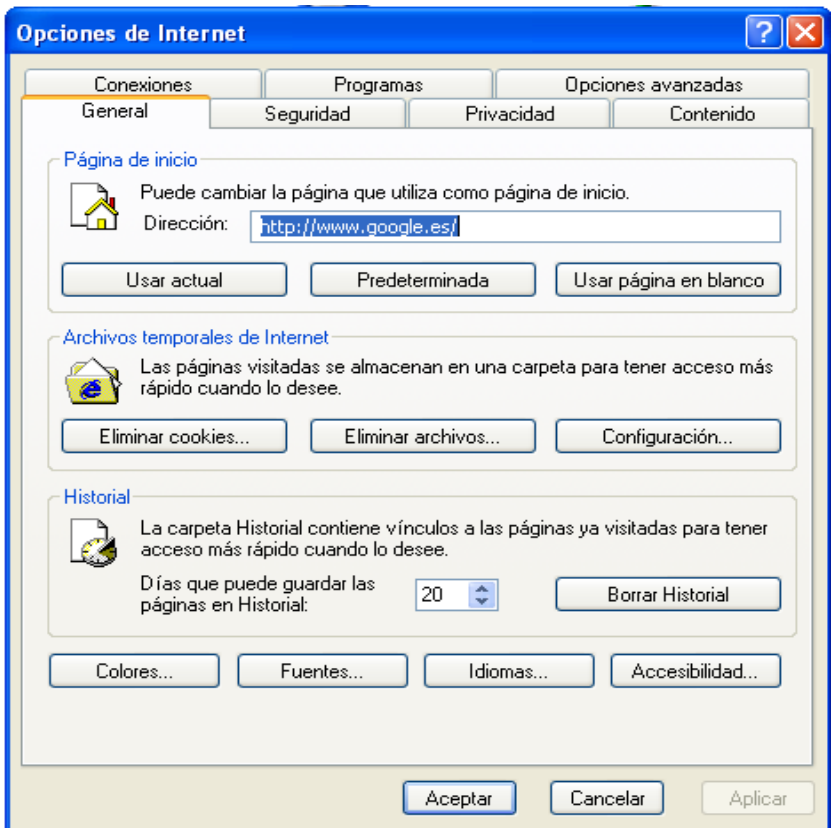


Figura 2.12. Interfaz que permite la eliminación de *Cookies*.

En ciertas ocasiones, por motivos de seguridad, no se permiten las *Cookies*, y por tanto, se debe acudir a otro tipo de técnicas como puede ser la reescritura de la *URL*. Esta técnica consiste en la modificación de la *URL* por parte del servidor para identificar cada *URL* con una sesión de un cliente.

3. AMPLIACIONES EN EL SERVIDOR

Si en el apartado anterior nos centrábamos en la posibilidad de personalizar nuestro cliente ligero engordándolo mediante diversas técnicas para dotar al servicio http de mayor funcionalidad, en este capítulo haremos lo propio con el servidor Web.

La mayor parte de las técnicas analizadas hasta el momento se basan en trasladar más o menos cantidad de conocimiento (*know-how*) desde el servidor hacia el cliente. Aunque esta técnica es muy interesante en numerosas ocasiones —sobre todo cuando se está buscando un control más fino de la interfaz y una respuesta más rápida al usuario—, en general resultan muy insuficientes para descargar al servidor de las sucesivas oleadas de demanda a la que debe enfrentarse a medida que se adentra en las turbulentas aguas del mundo de los negocios.

Las aplicaciones CGI nos permiten salir del paso, pero no están preparadas para soportar un gran número de transacciones, sobre todo porque el servidor no tiene el control de la ejecución de estas aplicaciones: no sabe realmente cómo están funcionando y no puede realizar optimizaciones de carga y ejecución.

Nuevamente, al no existir un estándar ampliamente reconocido, cada fabricante se lanzó a una carrera para proponer soluciones que pudieran lograr hacerse con la aceptación del mercado y, por lo tanto, convertirse con el tiempo en un estándar *de facto*. La realidad es que, aunque muchas ya han sido desechadas por este mercado, muchas otras se han afianzado en uno u otro contexto y, en la actualidad, conviven de una forma más o menos digna formando parte de las aplicaciones que operan sobre la Red.

3.1. EXTENSIONES DEL SERVIDOR

Una de las primeras ideas que se les ocurrió a los fabricantes de servidores Web consistió en permitir que sus clientes pudieran «construirse» su

servidor «a medida». Mediante determinados lenguajes nativos de la plataforma y siguiendo determinadas especificaciones podremos realizar filtros o extensiones al servidor Web, a fin de dotarle de funcionalidades de las que en principio carece. Obsérvese que la intención es la misma que en el caso de la interfaz CGI, pero aquí lo que se va a construir es una librería de acceso dinámico (`.dll` en Windows, `.so` en sistemas Unix). Esto significa que, a diferencia de los programas CGI que se instancian cada vez que son llamados, estas librerías, una vez cargadas en memoria, van a permanecer allí para atender cualquier petición posterior que se requiera de ellas (figura 3.1). Y, lo que es más interesante, se ejecutan como un módulo más, perfectamente entrelazado, del propio servidor.

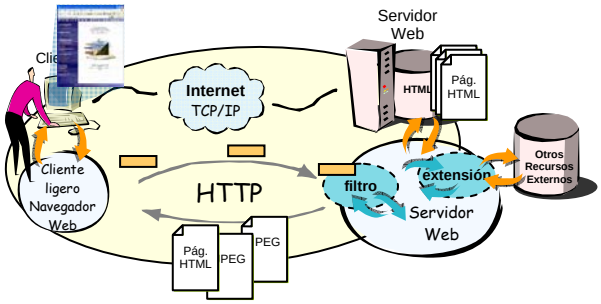


Figura 3.1. Escenario genérico para un modelo de filtros y extensiones del Servidor Web.

Aunque a nivel de desarrollo son equivalentes, funcionalmente un filtro se emplea para realizar un trabajo previo sobre la solicitud HTTP que llega al servidor Web mientras que las extensiones son invocadas (generalmente a través de estas mismas solicitudes HTTP) a posteriori para acceder de una forma eficiente a recursos o funcionalidades externas.

Para facilitar la labor, los diferentes fabricantes que han optado por esta técnica proporcionan una interfaz de programación de aplicaciones (API –*Application Programming Interface*) adecuada para su servidor Web. De esta forma, *Information Server API* (ISAPI) está orientado a *Internet Information Server* (IIS) de Microsoft y *Netscape API* (NSAPI), a los servidores Web de Netscape (Enterprise Netscape Server).

Por tanto, mediante esta tecnología ganamos sustancialmente en rendimiento y aprovechamiento de los recursos de la máquina. Sin embargo, nos encontramos con el gran inconveniente de estar cerrada, no sólo a una determinada plataforma, sino también a un determinado servidor Web e, incluso, a una determinada versión del mismo.

Otro inconveniente que ha contribuido a que esta técnica no sea muy empleada en la práctica es que la programación de estas API es bastante compleja o, como mínimo, con una curva de aprendizaje mayor que la que se requiere para realizar un programa CGI mediante un lenguaje y técnicas más o menos familiares para los desarrolladores.

3.2. PÁGINAS ACTIVAS

Esta tecnología está basada en incrustar código escrito en un lenguaje tipo *script* dentro de las propias páginas HTML con el fin de generar páginas dinámicas. Para ello, el servidor Web interpretará este código, accederá, si es necesario, a los recursos externos que precise y generará dinámicamente código en lenguaje HTML antes de servir la página al cliente (figura 3.2).

Esta forma de desarrollar aplicaciones basadas en Web es de las más aceptadas actualmente ya que existe una amplia alternativa de lenguajes más o menos dependientes de la plataforma y a que, debido a su concepción —código script incrustado en HTML—, se puede emplear cualquier editor HTML como herramienta de desarrollo.

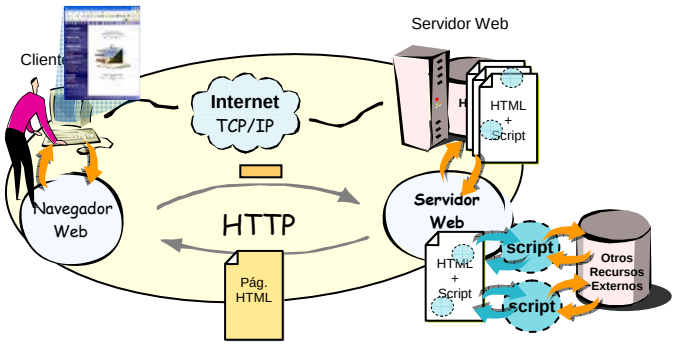


Figura 3.2. Escenario de desarrollo para una página activa.

A estas características que lo hacen popular, debemos unir otras de carácter más operativo, como el control que se obtiene entre la aplicación (el código script interpretado) y el servidor Web. Además, esta tecnología habilita el concepto de sesión de usuario, es decir, podemos conocer el estado de la conexión de un usuario en un momento determinado. Al mismo tiempo, gestiona bien los recursos externos, como es el caso de accesos a bases de datos, permite una gran flexibilidad para compartir los

mismos y para la gestión de la petición y respuesta Web. Finalmente, cómo el código se interpreta y ejecuta en el servidor, se mantiene la independencia del cliente Web.

A cambio, el desarrollador deberá aprender el lenguaje script correspondiente, el cual es propietario del servidor Web que lo ha de interpretar. Según esto, podemos encontrar diferentes alternativas como son: PHP, ASP, *LiveWire* o JSP.

A continuación analizaremos muy brevemente cada una de estas propuestas y mostraremos un sencillo ejemplo de cómo se puede presentar una página de saludo equivalente en cada una de ellas y cuya representación gráfica puede observarse en la figura 3.3.

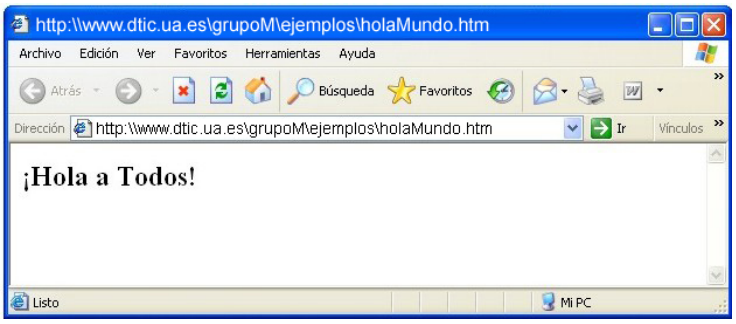


Figura 3.3. Representación por un navegador de la página HTML `holaMundo.htm`.

3.2.1. PHP

PHP es el acrónimo recursivo de «*PHP: Hypertext Preprocessor*» y puede considerarse como la propuesta pionera para el desarrollo de páginas dinámicas de una forma sistemática mediante el concepto de «páginas activas».

Puesto que esta característica no estaba incluida en los primeros servidores Web, se desarrolló mediante lenguaje PERL, como una serie de aplicaciones CGI que realizaban el preprocesamiento de páginas HTML con un código incrustado en etiquetas especiales al estilo de Perl. En la actualidad, diversos servidores Web, como Apache, ya incluyen esta característica de forma nativa.

La familiaridad de los desarrolladores en este tipo de lenguajes, la facilidad de uso y desarrollo, incluso, mediante un sencillo editor HTML, la potencia y el control que proporciona junto con su buena integración con el servidor Web, han contribuido notablemente a que esta tecnología sea

plenamente aceptada por la comunidad Web para el desarrollo de sitios dinámicos y aplicaciones Web en general.

En el listado 3.1 puede observarse una sencilla página HTML denominada `holaMundo.php` con código PHP incrustado dentro de la etiqueta especial `<?php …?>`, cuya interpretación por un navegador Web convencional puede observarse en la figura 3.3.

```
<html>
<head>
</head>
<body>
<h2>
<?php echo "¡Hola a Todos!"; ?>
</h2>
</body>
</html>
```

Listado 3.1. Contenido de la *página activa* **holaMundo.htm** diseñada para mostrar en el navegador un mensaje de bienvenida generado dinámicamente mediante *php*.

3.2.2. LiveWire

Este lenguaje fue desarrollado originalmente por Netscape en 1995, con la finalidad de proporcionar un mayor dinamismo a las páginas Web.

Los servidores Web de *Netscape* e *Iplanet* incluyen LiveWire que representa un entorno de desarrollo que utiliza JavaScript ejecutado desde el servidor para crear aplicaciones (de manera similar a CGI, JSP y ASP). A diferencia del código JavaScript que se ejecuta en el cliente (en algún *Browser* como Navigator o Explorer) las aplicaciones LiveWire JavaScript se ejecutan en el propio servidor. Para ello, primero son compiladas y almacenadas en un archivo ejecutable binario. Durante el proceso de ejecución, Livewire genera código HTML de manera dinámica. Por supuesto este código también puede incluir código Javascript para el navegador. Finalmente, el resultado es enviado al cliente a través del servidor Web para que éste interprete y muestre el resultado.

En el listado 3.2 puede observarse una sencilla página HTML denominada `holaMundo.htm` con código Livewire Javascript incrustado entre las etiqueta `<server>` y `</server>`, cuya interpretación por un navegador Web convencional puede apreciarse en la figura 3.3. Es muy importante no confundir Livewire javascript con el código javascript que se ejecuta en el cliente y que, aunque es el mismo, iría entre las etiquetas `<script>` y `</script>`.

```
<html>
<head>
</head>
<body>
<h2>
<SERVER>
    write("¡Hola a Todos!")
</SERVER>
</h2>
</body>
</html>
```

Listado 3.2. Contenido de la *página activa* **holaMundo.htm** que muestra en el navegador un mensaje de bienvenida generado dinámicamente mediante *Liveware Javascript*.

3.2.3. ASP

Los servidores Microsoft implementan la tecnología de páginas activas bajo el nombre de ASP (Active Server Pages). ASP se apoya en *Visual Basic Script* (VBScript) como lenguaje de *scripting* y está orientada al servidor *Internet Information Server* (IIS).

Las páginas pueden ser generadas incrustando código en las páginas HTML. Este código es interpretado por el servidor y desde el mismo se puede, como en el resto de casos, acceder a recursos y funcionalidades del equipo servidor o, incluso, de equipos remotos, incluyendo bases de datos.

En el listado 3.3 se muestra una sencilla página HTML denominada `holaMundo.asp` con código ASP incrustado dentro de la etiqueta especial `<% … %>`, cuya interpretación por un navegador Web convencional puede apreciarse en la figura 3.3.

```
<html>
<head>
</head>
<body>
<h2>
<% Response.Write "¡Hola a Todos!" %>
</h2>
</body>
</html>
```

Listado 3.3. Contenido de la *página activa* **holaMundo.asp** que muestra en el navegador un mensaje de bienvenida generado dinámicamente mediante *ASP*.

3.2.4. JSP

JavaServer Pages (JSP) es la propuesta desarrollada por Sun Microsystems para generar páginas Web de forma dinámica en el servidor. Esta tecnología está basada en Java y permite a los programadores generar dinámicamente HTML, XML o algún otro tipo de página Web. Para ello, como en el resto de casos, se inserta código, en este caso Java, dentro del contenido estático.

En versiones más recientes de la especificación se añadió la posibilidad de ejecutar *clases java* referenciadas desde la página HTML, mediante etiquetas denominadas «taglib». La asociación de las etiquetas con las clases java se declara en archivos de configuración escritos en XML.

La principal ventaja que presenta JSP frente a otras tecnologías es la posibilidad de integrarse con clases Java (`.class`). Esto permite organizar las aplicaciones Web en diferentes niveles, almacenando en clases java las partes que consumen más recursos o las que requieren más seguridad, manteniendo únicamente la parte encargada de proporcionar formato al documento HTML dentro del archivo JSP. Además, Java se caracteriza por ser un lenguaje que puede ejecutarse en cualquier sistema que incorpore una máquina virtual de java (JVM). Teniendo en cuenta que en la actualidad se pueden encontrar implementaciones de JVM para casi cualquier plataforma, la portabilidad de las aplicaciones es realmente considerable.

Como ocurría en el caso de LiveWire, antes de que el servidor Web pueda ejecutar el código java, debe compilarlo, generando un objeto «servlet» (ver apartado 3.3.1). Aunque este proceso ralentiza inicialmente la ejecución, en posteriores invocaciones no sólo no se producirá retardo, sino que se ejecutará más rápido y podrá disponer del API de Java en su totalidad.

En el listado 3.4 se presenta el código fuente de una sencilla página HTML denominada `holaMundo.jsp` con código JSP incrustado dentro de la etiqueta especial `<% … %>`, cuyo resultado, interpretado por un navegador Web convencional, puede verse en la figura 3.3.

```
<html>
<head>
</head>
<body>
<h2>
<%= "¡Hola a Todos!" %>
</h2>
</body>
</html>
```

Listado 3.4. Contenido de la página activa **holaMundo.jsp** que muestra en el navegador un mensaje de bienvenida generado dinámicamente mediante JSP.

3.2.5. ASP .NET

ASP.NET es la plataforma unificada de Microsoft para el desarrollo Web que proporciona a los desarrolladores los servicios necesarios para crear aplicaciones Web empresariales.

Microsoft ha visto en la estrategia de separar la lógica de la aplicación del contenido estático y de los aspectos de presentación desarrollada por SUN, su más directa competidora, una seria amenaza. Esta es la principal motivación que le ha llevado a incorporar en su plataforma *.NET* un lenguaje de *scripts ASP.NET* que permite ser integrado con clases *.NET* (ya estén desarrolladas en *C++*, *VisualBasic* o *C#*) del mismo modo que JSP se integra con *clases Java*.

Esta nueva tecnología se ha desarrollado como parte de la estrategia *.NET* para el desarrollo Web, con el objetivo de resolver las limitaciones de ASP y posibilitar la creación de *software como servicio*.

Para distinguir unas versiones ASP de otras, las versiones «pre-.NET» se denominan actualmente «ASP clásico».

3.3. COMPONENTES EN LA PARTE DEL SERVIDOR

Un componente en la parte del servidor o un «servicio ligero» conceptualmente no difiere mucho de los componentes en la parte del cliente o aplicaciones ligeras tratadas en el apartado 2.3. En cualquier caso, la diferencia fundamental radica en que este software ahora no se transfiere al cliente, sino que se ejecuta directamente en el propio servidor Web.

Según esta definición, los componentes en la parte del servidor también podrían confundirse con las aplicaciones CGI analizadas en el

punto 1.3, sin embargo, también existe una diferencia notable con respecto a éstas y es que su ejecución sí se desarrolla dentro del contexto del propio servidor Web —recordemos que en el caso de las aplicaciones CGI, dicha ejecución era gestionada por el sistema operativo en un contexto diferente al del servidor Web—. Esto dota al servidor y a la propia aplicación de un control mutuo que permite superar las limitaciones del modelo CGI. Un servidor Web capaz de ejecutar este tipo de componentes se denomina también, «contenedor Web».

Funcionalmente, los componentes software en la parte del servidor se pueden considerar *piezas de software* creadas para encapsular una determinada *lógica de negocio*, un determinado servicio o, incluso, ser empleados como almacén de datos. Estos componentes se utilizan en la construcción de aplicaciones como si fueran piezas de un mecano y están diseñados para ser reutilizables.

Si bien con las tecnologías revisadas hasta el momento se pueden desarrollar básicamente los mismos servicios, a medida que estos servicios o aplicaciones Web adquieren relevancia, no están preparadas para hacerlo con un rendimiento adecuado o para poder aplicar técnicas de ingeniería del software que faciliten el desarrollo inicial y su posterior mantenimiento. Con la apuesta firme del mundo de los negocios por las tecnologías Web para desarrollar sus aplicaciones, han ido surgiendo y evolucionando diversas propuestas encaminadas a suplir estas necesidades operativas, como *servlets*, *java beans* o *COM*. Durante los siguientes apartados iremos analizando las opciones disponibles más extendidas.

3.3.1. Servlets

Los *servlets* son programas escritos en Java, se pueden invocar desde un cliente, de forma similar a como se invocan los programas CGI; se ejecutan en el seno del servidor Web, que actúa como un «contenedor de componentes»; realizan una determinada tarea; y generan una salida (página Web dinámica) que es recogida por el servidor Web y posteriormente reenviada al navegador que realizó la invocación de este componente.

Teniendo encuentra que un *servlet* es un objeto software que se invoca externamente, representa un modelo mucho más cercano al *modelo CGI* que al *modelo de páginas activas*, donde este objeto se encuentra incrustado dentro de código HTML estático. Sin embargo, la diferencia fundamental con respecto al modelo CGI clásico se encuentra en que un

servlet se ejecuta dentro del contexto, y por lo tanto del control, del servidor o, ahora, contenedor Web (como por ejemplo: *Apache Tomcat*).

En el listado 3.5 se presenta el código fuente de un servlet, cuya interpretación por un navegador Web convencional puede verse en la figura

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HolaMundoServlet extends HttpServlet {

    public void init(ServletConfig conf)
            throws ServletException {
        super.init(conf);
    }  // de initServlet

    public void service(HttpServletRequest req, \
            HttpServletResponse res)
            throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter salida = res.getWriter();

        salida.println("<html>");
        salida.println("<head>");
        salida.println("</head>");
        salida.println("<body>");
        salida.println("<h2>¡Hola a Todos!</h2;>");
        salida.println("</body>");
        salida.println("</html>");
    }  // de service
}  // de HolaMundoServlet
```

3.3.

Listado 3.5. Contenido de la página activa `holaMundo.js` que muestra en el navegador un mensaje de bienvenida.

Sus características y método de trabajo son los siguientes:

- **Portabilidad**. Al estar escritos en Java, los servlets garantizan la portabilidad entre diferentes plataformas que dispongan de la *Máquina Virtual Java*. También existe portabilidad entre los diferentes servidores Web, ya que el API Servlet define una interfaz estándar entre el servlet y el servidor Web que es independiente del fabricante. Por lo tanto, no hay especificaciones dependientes del

servidor, como ocurre con la interfaz ISAPI/NSAPI o con las *páginas activas*.

- **Rendimiento**. Al igual que las extensiones del servidor ISAPI/NSAPI y a diferencia de CGI, los servlets son instanciados una única vez, pudiendo, a partir de entonces, atender diferentes peticiones. La instanciación del servlet se puede producir cuando tiene lugar la primera petición o estar ya instanciado desde la propia inicialización del servidor Web.
- **Concepto de sesión**. El servlet puede mantener información acerca de la sesión de usuario, variables o recursos —como por ejemplo: una conexión a la base de datos— entre diferentes conexiones dirigidas al mismo servlet e, incluso, entre varios de ellos.
- **Software distribuido**. Se pueden cargar indiferentemente y de forma transparente tanto desde un disco local como desde una dirección remota, respondiendo a la nueva filosofía de software distribuido. Los servlets pueden comunicarse entre sí y, por tanto, es posible una reasignación dinámica de la carga de proceso entre diversas máquinas, es decir, un servlet podría pasarle trabajo a otro servlet residente en otra máquina.
- **Ventajas inherentes a Java**. Además de la portabilidad, se hereda otras ventajas de este lenguaje idóneo para desarrollos en Internet, como es la orientación a objetos, la modularidad y la reutilización. Por otra parte, la responsabilidad de la gestión de memoria, tarea a cargo del programador en otros lenguajes, recae en el propio Java, facilitando la recolección automática de memoria no utilizada e impidiendo el acceso directo a la misma.
- **Multithread**. Dado que los servlets pueden manejar múltiples peticiones concurrentemente, es posible implementar aplicaciones cooperativas, como por ejemplo una aplicación de videoconferencia.

3.3.2. JavaBeans

Un *JavaBean*, o sencillamente un «bean», representa un modelo de componentes software especificado por Sun Microsystems dentro de su plataforma *Java 2 Platform, Standard Edition* (J2SE) para la construcción de aplicaciones Java. Estos componentes están pensados para poder ser reutilizados fácilmente en la creación de aplicaciones.

A diferencia de los servlets, los beans están diseñados para actuar de forma autónoma, ejecutándose dentro del contexto de un contenedor más genérico que el servidor Web. Se podría decir que esta propuesta marca el

inicio de la imparable estrategia por desacoplar la lógica de negocio, no sólo del código HTML, sino también del propio servidor Web. Este nuevo contexto se denomina «Servidor de Aplicaciones». Sin embargo, dejaremos su desarrollo para el próximo apartado, en el que analizaremos propuestas mucho más específicas y maduras, inspiradas desde el comienzo para proporcionar soporte a este nuevo concepto.

Los JavaBeans, además de exponer sus propiedades para poder ser personalizados y utilizar los eventos para comunicarse con el sistema o con otros beans, también utilizan la serialización de objetos Java para soportar la persistencia —lo que les permite guardar su estado y restaurarlo posteriormente—. Además, sus métodos, que no son diferentes de otros métodos Java, pueden ser invocados desde otros beans o desde páginas JSP.

Los beans han sido diseñados para que todas las claves de su API puedan ser entendidas y gestionadas por herramientas de desarrollo visual, incluyendo el soporte para eventos, las propiedades y la persistencia. Son los llamados *beans visuales*, y están íntimamente relacionados con la interfaz gráfica de usuario.

En cualquier caso, en el contexto del servidor de aplicaciones hablamos de los *beans no-visuales*, que pueden ser utilizados de forma programática, conociendo su funcionalidad e interfaz. En este grupo se distinguen dos tipos por su funcionalidad: los que contienen una determinada lógica de negocio y los que sirven como almacén de datos a los que se accederá posteriormente, cuando se necesite la información que albergan, en otro punto de la aplicación Web.

Aunque los beans individuales pueden variar ampliamente en funcionalidad, desde los más simples a los más complejos comparten las siguientes características:

- **Introspección (introspection)**: permite que la herramienta de programación o IDE pueda analizar cómo trabaja el bean.
- **Personalización (customization)**: el programador puede alterar la apariencia y la conducta del bean y se permite cambiar los valores de las propiedades del bean para personalizarlo.
- **Eventos (events)**: informa al IDE de los sucesos que puede generar en respuesta a las acciones del usuario o del sistema, y también los sucesos que puede manejar.
- **Persistencia (persistente)**: un bean tiene que tener persistencia, es decir, implementar el interfaz *Serializable*.

Cuando el IDE carga un bean, usa el mecanismo denominado *reflection* para examinar todos los métodos, fijándose en aquellos que

empiezan por **set** y **get**. El IDE añade las propiedades que encuentra a la hoja de propiedades para que el programador personalice el bean.

3.3.3. Objetos OLE, ActiveX y COM

Microsoft fue uno de los pioneros en poner en práctica las ventajas del concepto de objeto implementando el intercambio dinámico de datos (*Dynamic Data Exchange* —DDE) desde las primeras versiones de Windows. Como evolución de este concepto desarrolló un sistema de objeto distribuido junto con un protocolo que denominó (*Object Linking and Embedding* —OLE 1.0). Mientras que DDE se limitaba a transferir una cantidad concreta de información entre dos aplicaciones, OLE fue capaz de mantener enlaces activos entre dos documentos o incluso incrustar un tipo de documento dentro de otro.

El principal cometido de OLE era la gestión de documentos compuestos, pero también se podía emplear para transferir datos entre aplicaciones mediante la técnica de «arrastrar y soltar» (*drag and drop*) y operaciones del portapapeles (*clipboard*). Sin embargo, el concepto que popularizó esta tecnología en el entorno Web era el de incrustar (*embedding*) objetos multimedia (animaciones flash, archivos de vídeo, etc.) en páginas Web, dentro del código HTML.

El problema es que OLE 1.0 es una tecnología del lado del cliente, pues el objeto incrustado debía estar situado en el equipo en el que se iba a trabajar.

Posteriormente OLE 1.0 evolucionó a OLE 2.0 que, por motivos probablemente comerciales, pasó a denominarse ActiveX. Además, y para confundir más aun las cosas, OLE 2.0 se volvió a implementar sobre COM (*Component Object Model*).

COM representa la primera arquitectura software de componentes de Microsoft, diseñada para facilitar la reutilización de software y la interoperatividad de los componentes mediante la comunicación entre procesos y la creación dinámica de objetos, en cualquier lenguaje de programación que soporte dicha tecnología.

El término COM es a menudo usado en el mundo del desarrollo de software como un término que abarca tecnologías anteriores a las que había absorbido o de las que había evolucionado: OLE, *OLE Automation* o ActiveX; pero también podemos encontrar referencias a otras tecnologías que han evolucionado a partir de ella, como DCOM y COM+, que estudiaremos con más detalle en el capítulo 4.