Entorno tecnológico (Alternativas)

Florentino Fernández [1] and Roberto Casado-Vara [2]

[1] University of Vigo, Circunvalación ao Campus Universitario, 36310 Vigo, Pontevedra, Spain
verola@uvigo.es
[2] University of Salamanca, Plaza de los Caídos s/n – 37002 – Salamanca, Spain
rober@usal.es

Resumen. Con el nacimiento de la World Wide Web (WWW) se creó un sistema de documentos de hipertexto enlazados mediante hipervínculos para poder acceder por medio de internet. La gran diversidad de navegadores y servidores Web que han surgido desde la creación de la WWW hace que sea importante mantener unos estándares con el fin de maximizar la compatibilidad e interoperabilidad entre todos ellos. Como ya se ha comentado, el organismo responsable de mantener la mayor parte de estos estándares es el World Wide Web Consortium (W3C), cuyo director desde el año 2007 es Tim Berners-Lee. En este capitulo vamos a repasar los principales protocolos de la WWW y, las tecnologías web asociados a ellos.

Palabras clave: World Wide Web.

Abstract. With the growth of the World Wide Web (WWW) a system of hyperlinked hypertext documents was created for access via the Internet. The great diversity of browsers and Web servers that have emerged since the creation of the WWW makes it important to maintain standards in order to maximize compatibility and interoperability among them all. As already mentioned, the body responsible for maintaining most of these standards is the World Wide Web Consortium (W3C), whose director since 2007 is Tim Berners-Lee. In this chapter we will review the main protocols of the WWW and the web technologies associated with them.

Keywords: World Wide Web

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

1 Introducción

1.1 ¿Qué es la World Wide Web?

La World Wide Web (WWW) es un sistema de documentos de hipertexto y de hipermedia enlazados por medio de hipervínculos, a los cuales se puede acceder por medio de Internet. Está compuesto por un conjunto de sitios Web que almacenan páginas de hipertexto que pueden ser visitados mediante un navegador Web.

El organismo responsable de mantener y gestionar los estándares de la WWW es el World Wide Web Consortium (W3C). Este consorcio está formado, actualmente, por más de trescientas organizaciones (listadas en http://www.w3.org/Consortium/Member/List) entre las que se encuentran empresas, organizaciones sin ánimo de lucro, universidades y entidades gubernamentales [1-5].

1.2 Historia de la World Wide Web

El nacimiento de la WWW se produce alrededor de 1989, aunque las ideas en las se basa datan de los años 40 y 50.

En los años 40, Vannevar Bush propuso el sistema Memex, un sistema documental de almacenamiento y consulta de todo tipo de documentos basado en microfilms. El sistema, que nunca llegó a ser materializado, permitía almacenar los documentos de manera que resultaba sencillo recuperar un documento concreto junto con aquellos que estuviesen relacionados con él. El sistema también le proporcionaba al lector mecanismos para añadir anotaciones a los documentos, que pasaban a formar parte de la información almacenada.

Posteriormente, en la década de los 50, Theodore Holm "Ted" Nelson describió lo que se podía considerar como un sistema de hipertexto, en el cual la información estaba enlazada. Fue el primero en acuñar el término hipertexto para definir aquellos documentos de texto relacionados entre sí mediante hiperenlaces y que pueden ser consultados mediante medios no lineales. También acuñó el término hipermedia como extensión del término hipertexto para hacer referencia a imágenes, vídeos, sonido y otros tipos de medios.

En la segunda mitad de 1980, Tim Berners-Lee diseña e implementa el sistema ENQUIRE para el CERN (*Organización Europea para la Investigación Nuclear*). Éste era un sistema servidor que almacenaba los documentos en una base de datos y permitía gestionarlos. Dos de las características más destacables de este sistema eran que permitía editar los documentos directamente en el servidor y establecer enlaces (hipervínculos) entre los documentos.

Tomando ideas de este sistema, en 1989, Tim Berners-Lee redactó para el CERN una propuesta de sistema de gestión de información que acabaría convirtiéndose en lo que hoy conocemos como la World Wide Web. Ayudado por Robert Cailliau, en 1990 publican un documento más elaborado y formal para describir la propuesta inicial de Tim Berners-Lee. Ese mismo año Tim Berners-Lee lleva a cabo su idea implementando el primer servidor Web, al que llamó HTTPd (*HyperText Transfer Prococol daemon*), y el primer navegador Web, al que llamó WorldWideWeb. Para probar el sistema desarrollado también creó las primeras páginas Web, cuyo contenido fue la descripción del proyecto.

El siguiente paso de Tim Berners-Lee fue convertir su idea en un nuevo servicio público de Internet. Esto se produjo en agosto de 1991 y desde entonces la WWW no ha parado de crecer.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

1.3 Los Estándares en la World Wide Web

La gran diversidad de navegadores y servidores Web que han surgido desde la creación de la WWW hace que sea importante mantener unos estándares con el fin de maximizar la compatibilidad e interoperabilidad entre todos ellos.
Como ya se ha comentado, el organismo responsable de mantener la mayor parte de estos estándares es el World Wide Web Consortium (W3C), cuyo director desde el año 2007 es Tim Berners-Lee.
Actualmente existen una gran cantidad de estándares en la WWW (algunos de ellos empleados también en otros ámbitos) que unifican criterios sobre diversos elementos, tales como la estructura de los documentos, el protocolo de comunicaciones, la representación de los documentos, etc.
Algunos de los principales estándares son:

- **HTML (HyperText Markup Language) y XHTML (eXtensible HyperText Markup Language):** Son lenguajes de marcado empleados en la creación de documentos para la Web.
- **XML (eXtensible Markup Language):** Es un lenguaje de marcado que permite definir nuevos lenguajes.
- **HTTP (HyperText Transfer Protocol):** Es el protocolo de comunicaciones empleado para el envío de mensajes entre cliente y servidor.
- **URI (Uniform Resource Identifier):** Es un sistema universal de identificación de recursos en Internet.

1.4 El Protocolo HTTP

Uno de los protocolos más importantes para comprender el funcionamiento de la WWW es el protocolo HTTP (HyperText Transfer Protocol), pues define cómo se deben realizar las comunicaciones entre clientes y servidores.
El HTTP es el protocolo de nivel de aplicación de la WWW, estandarizado por el W3C. Como todo protocolo, establece un conjunto de reglas de comunicación para el intercambio de datos o información multimedia (gráficos, audio, etc.). Su versión actual es la 1.1, la cual considera la existencia de ordenadores intermedios (como proxys, gateways, etc.), cachés de ficheros, mantenimiento de conexiones activas y otros aspectos [6-10].
Las conexiones HTTP se realizan utilizando el protocolo de transporte TCP/IP. En este protocolo la identificación de los interlocutores (cliente y servidor) se hace mediante direcciones IP. Cuando un cliente desea hacer una petición a un servidor, el cliente abre una conexión con un puerto determinado del servidor (por defecto el puerto 80), identificado y localizado por su dirección IP. Entonces transmite su solicitud, que incluye:

- El tipo de operación que solicita el cliente (obtención de un recurso, envío de parámetros o información acerca de un recurso).
- Identificación del documento, fichero o recurso que se quiere recuperar mediante una URL (*Uniform Resource Locator*) o un URI (*Uniform Resource Identifier*).
- La versión del protocolo que implementa el navegador del cliente (generalmente HTTP/1.0 o HTTP/1.1).

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
<method> <resource identifier> <HTTP versión><crlf>
[<Header> : <value>]<crlf>
…
[<Header> : <value>]<crlf>
<blank line><crlf>
[Entity body]
```

- Una serie de modificadores aplicables a la misma petición. Como por ejemplo, la versión y nombre del programa que hace la petición (*User-Agent*), tipos de datos MIME (*Multi-purpose Internet Mail Extensions*) del fichero de respuesta que se espera (*Accept*), etc.
- Adicionalmente, se pueden incluir datos en la petición (parámetros de un formulario) dentro del cuerpo de esta.

El servidor, por su parte, incluye en su respuesta:

- Una línea de estado con el código y mensaje de éxito o error según corresponda, además de la versión del protocolo HTTP que implementa el servidor.
- Información propia del servidor (nombre y versión del programa).
- Información sobre el documento o recurso solicitado (fecha de última modificación, tamaño, etc.).
- El documento o recurso solicitado.

Finalmente, el servidor cierra la conexión, aunque este último aspecto depende de la versión HTTP que implemente el servidor. La especificación HTTP/1.1 implementa el mantenimiento de conexiones abiertas, por lo que es necesario incluir el modificador [*Connection: close*] para que el servidor cierre la conexión después de dar la respuesta. Las conexiones abiertas deben usarse cuando un cliente vaya a solicitar un gran número de recursos del servidor en un corto espacio de tiempo. Los servidores HTTP/1.0 cierran automáticamente la conexión después de enviar la respuesta.

1.4.1 Solicitud HTTP

Una solicitud HTTP se compone de una línea de solicitud, uno o más campos de encabezado opcionales y un cuerpo de entidad también opcional. Las líneas se separan por medio de un retorno de carro y avance de línea (*CR/LF: carriage-return/line-feed*). El cuerpo de entidad va precedido por una línea en blanco. El formato de una solicitud es el siguiente.

La línea de solicitud se compone de tres campos de texto, separados por espacios en blanco. El primer campo (*method*) especifica el método (o comando) que se aplicará al recurso del servidor. El método más común es GET, por medio del cual se solicita al servidor el envío de una copia del recurso al cliente. El segundo campo (*resource identifier*) especifica el nombre del recurso al que se hace referencia en la solicitud, es decir, la URL sin el protocolo ni el nombre de dominio del servidor. El tercer campo (*http version*) identifica la versión del protocolo usada por el cliente, por ejemplo: HTTP/1.0.
Los campos de encabezado de solicitud (*Header*), ofrecen información adicional sobre la solicitud y sobre el cliente al servidor. Cada campo de encabezado consiste en un nombre, seguido por dos

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

puntos (:) y el valor del campo. El orden en que se transmiten los campos de encabezado no es significativo.
El cuerpo de entidad se emplea cuando el cliente debe enviar al servidor datos en un formato especificado. Debe ir precedido por una línea en blanco.
La Tabla 9 muestra información sobre los distintos métodos disponibles en cada una de las versiones del protocolo HTTP.

Método	HTTP/1.0	HTTP/1.1	Descripción del método
GET	S	S	Recupera la URL especificada.
GET condicional	N	S	Recupera el recurso si se cumplen las condiciones incluidas en la petición. Si se añade el parámetro *If-Modified-Since*, con una fecha en formato HTTP, el servidor comprueba que el recurso ha sido actualizado con posterioridad a la fecha recibida en la petición e incluirá el recurso en su respuesta. En otro caso, indicará que el recurso no ha sido modificado y por lo tanto no será enviado. También puede incluir parámetros en la URL. Por ejemplo, una petición HTTP con GET condicional sería: *GET /default.htm http/1.1* *If-Modified-Since: Sat, 14 Sep 2002 01:15:48 GMT* *Host: www.microsoft.com* *Connection: close* *Accept: */**
HEAD	S	S	Idéntico a GET, salvo que el servidor no envía el documento completo en la respuesta, sólo envía los encabezados. Los clientes lo usan para obtener metadatos de recursos o para probar la validez de vínculos de hipertexto.
POST	S	S	Envía los datos a la URL especificada.
PUT	N	S	Almacena los datos en la URL especificada. Sobrescribe el contenido anterior.
PATCH	N	S	Similar a PUT, salvo que contiene una lista de diferencias entre la versión original de la URL y el contenido deseado tras la aplicación del método.
COPY	N	S	Copia el recurso identificado por la URL en la(s) ubicación(es) especificada(s).
MOVE	N	S	Traslada el recurso indicado por la URL a la(s) ubicación(es) especificada(s). Este método es equivalente a COPY + DELETE.
DELETE	N	S	Borrar el recurso identificado por la URL.
LINK	N	S	Establece una o más relaciones de vinculación entre el recurso identificado por la URL y otros recursos.
UNLINK	N	S	Eliminar una o más relaciones de vinculación en la URL especificada.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

TRACE	N	S	Notifica todo lo que se recibe del cliente en el cuerpo de entidad de la respuesta.
OPTIONS	N	S	Solicita información sobre las opciones de comunicación disponibles en la cadena de solicitud/respuesta para la URL especificada. Permite a los clientes determinar las capacidades de un servidor sin recuperar un recurso.
WRAPPED	N	S	Permite que las solicitudes se encapsulen y codifiquen en un sólo conjunto para reforzar la seguridad y/o privacidad de la solicitud. El servidor de destino debe desencapsular el mensaje y cederlo al manipulador apropiado.

Tabla 9: Métodos disponibles en el protocolo HTTP.

En la Tabla 10 se muestran los campos de encabezado disponibles para las diferentes versiones del protocolo HTTP. HTTP/1.1 que soporta más de 41 cabeceras, en contraposición con las 17 que incorpora HTTP/1.0. La Tabla 10 está dividida en cuatro categorías: (*i*) *encabezados generales*, que pueden aparecer en mensajes tanto de solicitud como de respuesta, (*ii*) *encabezados de solicitudes*, que sólo pueden aparecer en mensajes de solicitud, (*iii*) *encabezados de respuestas*, que sólo pueden aparecer en mensajes de respuesta y (*iv*) *encabezados de entidad*, que pueden aparecer en solicitudes o respuestas. Los encabezados de entidad describen el contenido de los datos del mensaje, por ejemplo, un documento devuelto por un servidor o datos con formato enviados por un cliente.

Encabezado	HTTP/1.0	HTTP/1.1
Encabezados generales de HTTP		
Cache-Control	N	S
Connection	N	S
Date	S	S
Forwarded	N	S
Keep-Alive	N	S
MIME-Version	S	S
Pragma	S	S
Upgrade	N	S
Encabezados de solicitudes de HTTP		
Accept	N	S
Accept-Chars	N	S
Accept-Encoding	N	S
Accept-Language	N	S
Authorization	S	S
From	S	S
Host	N	S
If-Modified-Since	S	S
Proxy-Authorization	N	S
Refer	S	S
Unless	N	S
User-Agent	S	S
Encabezados de respuestas de HTTP		
Location	S	S
Proxy-Authenticate	N	S
Public	N	S
Retry-After	N	S
Server	S	S
WWW-Authenticate	S	S
Encabezados de entidad de HTTP		
Allow	S	S
Content-Encoding	S	S
Content-Language	N	S
Content-Length	S	S
Content-Type	S	S
Content-Version	N	S
Derived-From	N	S
Expires	S	S

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

Last-Modified	S	S
Link	N	S
Title	N	S
Transfer-Encoding	N	S
URL-Header	N	S

Tabla 10: Cabeceras utilizadas en el protocolo HTTP.

1.4.2 Respuesta HTTP

La sintaxis de una respuesta HTTP consiste en una línea de encabezado de respuesta, uno o más campos de encabezado de respuesta opcionales y un cuerpo de entidad, también opcional. Las líneas se separan por medio de un retorno de carro y avance de línea (*CR/LF: carriage-return/line-feed*). El cuerpo de entidad debe ir precedido por una línea en blanco. El formato de una respuesta es el siguiente.

```
<HTTP version> <result code> [<explanation>]<crlf>
[<Header> : <value>]<crlf>
…
[<Header> : <value>]<crlf>
<blank line><crlf>
[Entity body]
```

La línea de encabezado de respuesta envía la versión de http (*HTTP version*), el estado de la respuesta (*result code*) y explicación opcional del estado de devolución (*explanation*). Los campos de encabezado de respuesta envían información en la que se describen los atributos del servidor y del documento HTML enviado al cliente. Cada campo de encabezado se compone de un nombre, seguido por dos puntos (:) y el valor del campo. El orden en que el servidor remite los campos de encabezado no es relevante. El cuerpo de entidad contiene, por lo general, el documento HTML que el cliente ha solicitado [11-18].

Los códigos de respuesta que pueden ser devueltos por un servidor HTTP se pueden ver en la Tabla 11.

Información 1xx
100 Continue
101 Switching Protocols
Éxito 2xx
200 OK
201 Created
202 Accepted
203 Non-Authoritative Information
204 No Content
205 Reset Content
206 Partial Content
Redireccionamiento 3xx
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
306 (Unused)
307 Temporary Redirect
Error del cliente 4xx
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
Error del servidor 5xx
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported

Tabla 11: Códigos de respuesta enviados por el servidor HTTP.

2 Tecnologías Web

2.1 Estado del Arte

Inicialmente la WWW se pensó para que los servidores proporcionasen documentos estáticos a los clientes. Los documentos se encontraban almacenados en el servidor y no existía más procesamiento que el de enviar el documento al cliente cuando éste lo solicitase.
Con la evolución de la Web, los servidores empezaron a incluir contenidos dinámicos, de forma que los documentos o parte de ellos se generan en el momento en el que se crea la respuesta a una petición del cliente.
La generación dinámica de contenidos para la Web, requiere que el servidor realice algún tipo de procesamiento adicional sobre la petición HTTP iniciada por el cliente, con el fin de generar una respuesta personalizada y adaptada al usuario.

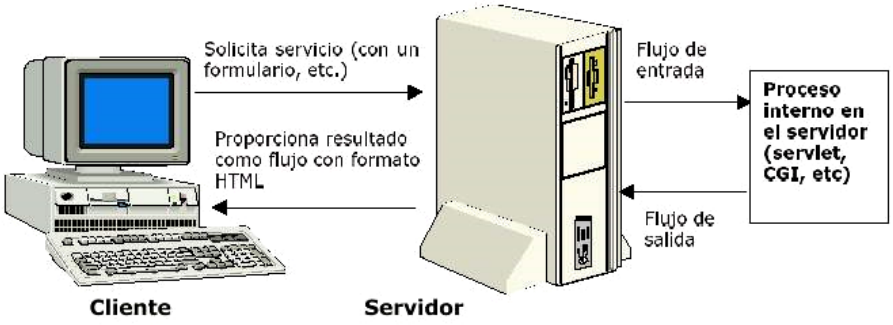


Figura 2: Esquema de conexión para la generación dinámica de contenidos

La mayoría de los sistemas de generación dinámica de contenido (ColdFusion, ASP, PHP, etc.) emplean lenguajes de script. La utilización de estos lenguajes no requiere el ciclo clásico de edición-compilación-enlace, posibilitando así una rápida codificación y visión de resultados. Debido a que las etiquetas HTML proporcionan un marco donde el contenido dinámico generado se inserta, estas herramientas se llaman comúnmente sistemas de plantillas (*template systems*).
A continuación veremos algunas de las tecnologías y sistemas de generación de contenido dinámico que podemos encontrar hoy en día en la Web.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

2.2 CGI: Common Gateway Interface

En los primeros años de existencia de la Web, tan sólo se podía acceder a información estática. De esta forma, la interactividad con el usuario no existía. Los primeros servidores HTTP no incluían ningún mecanismo para generar respuestas dinámicamente, por lo tanto se desarrollaron interfaces para comunicar el servidor con programas externos que implementaran dicha funcionalidad.

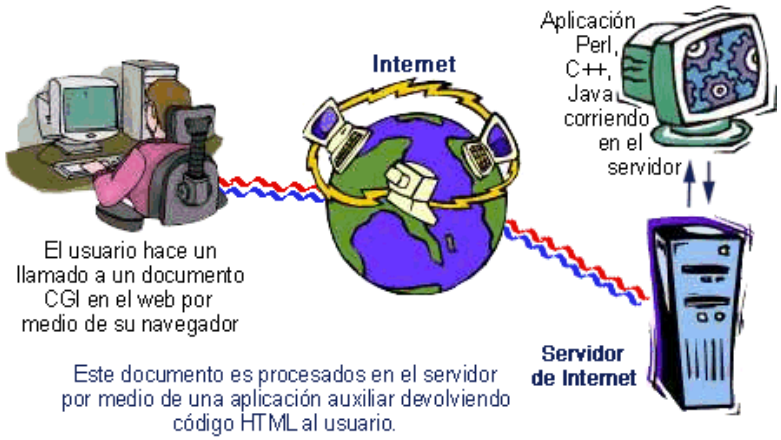


Figura 3: Esquema de conexión y respuesta para soluciones CGI

Este mecanismo de comunicación se denominó interfaz CGI (*Interfaz de Pasarela Común*). La interfaz permite que el servidor Web se comunique con otros programas que realizan tareas diversas. Estos programas se ejecutan como procesos independientes del servidor HTTP. La Figura 3 muestra de forma esquemática el proceso de una petición CGI.

La especificación CGI describe una interfaz estándar que sirve para que un servidor Web envíe solicitudes del navegador al programa CGI, y éste devuelva los datos de respuesta al navegador a través del servidor Web.

La mayor parte de programas CGI están escritos en lenguajes de automatización como Perl (*Practical Extraction and Report Language*) y scripts del shell de UNIX, aunque también se pueden escribir en lenguajes compilados como C y C++. Algunos programas CGI basados en Windows están escritos en Visual Basic, Delphi o incluso en ficheros de proceso por lotes (ficheros .bat).

Este tipo de procesamiento posee ineficiencias inherentes a su estructura. Puesto que el programa CGI se ejecuta fuera del servidor Web, se debe ejecutar un nuevo proceso cada vez que se dé servicio a una petición CGI. Además, los programas CGI están diseñados para manejar solamente una petición y terminar, lo que los hace inadecuados para sitios Web que soportan miles de peticiones simultáneas [19-25].

Actualmente, los sistemas de generación de contenido dinámico implementan módulos en forma de plug-ins (subprocesos del servidor Web). Estos módulos amplían la API del servidor con la finalidad de interaccionar directamente con él sin incurrir en la sobrecarga ocasionada por los programas CGI y permitiendo al mismo tiempo una mayor escalabilidad que la aproximación tradicional.

Los servidores Web se pueden comunicar con los programas CGI por medio de tres métodos:

- **Variables de entorno**: Las variables de entorno se definen fuera del programa pero en el contexto de éste. De forma que la posibilidad de leer variables de entorno desde un lenguaje es crucial para la implementación de programas CGI.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

- **Argumentos de línea de órdenes**: Ciertos tipos de solicitudes del navegador hacen que el servidor Web pase información a programas CGI por medio de los argumentos de línea de órdenes de dicho programa.
- **Entrada estándar**: Las solicitudes del navegador que se envían por medio del método POST, van al programa CGI a través de su flujo de entrada estándar.

Aunque existen tres maneras para que la información llegue desde el servidor Web hasta el programa CGI, éste sólo puede utilizar el flujo de salida estándar para devolver información al servidor Web. La Figura 4 muestra el esquema de servicio de una petición CGI.

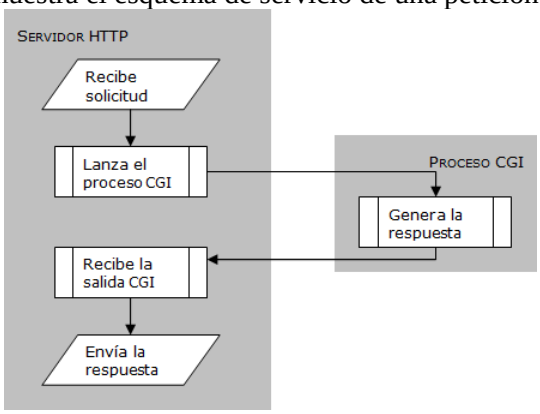


Figura 4: Esquema de servicio de una solicitud CGI.

2.2.1 Fast CGI

FastCGI es la evolución natural de CGI en la que el programa CGI se conserva en memoria de forma persistente. Mediante FastCGI existe la posibilidad de que el programa CGI resida en máquinas diferentes a la del servidor de Web, algo que conlleva ventajas a la hora de repartir la carga y fijar políticas de seguridad.

FastCGI es una propuesta abierta y libre en servidores Web comerciales que lo soporten. El primer servidor Web en emplear FastCGI fue *Open Market WebServer*, aunque actualmente son muchos los servidores que lo soportan.

Las aplicaciones CGI y FastCGI son caminos efectivos para permitir a una aplicación actuar como una extensión del servidor Web. CGI no suministra explícitamente soporte para diferentes clases de aplicaciones: bajo CGI, todas las aplicaciones reciben una petición HTTP, la procesan de alguna forma y generan una respuesta que será enviada al cliente. FastCGI suministra soporte para varios "roles" comunes que las aplicaciones pueden desempeñar. Por ejemplo, los tres papeles soportados por *Open MarketWebServer 2.0* son: *responder*, *filter* y *authorizer*.

- **Aplicaciones Contestadoras (*Responder*)**: Una aplicación responder es la clase más básica de aplicación FastCGI. Recibe la información asociada con la petición HTTP y genera una respuesta HTTP. El Responder es el rol más similar a la programación tradicional de CGI.
- **Aplicaciones Filtro (*Filter*)**: Un aplicación filtro FastCGI recibe la información asociada con la petición HTTP más un flujo extra de datos de un fichero almacenado en el servidor Web, generando una versión filtrada del flujo de datos como respuesta HTTP.

  Con las aplicaciones filtro, el administrador del sistema asocia a cada tipo MIME una aplicación FastCGI particular para su filtrado. Cuando un cliente pide una URL con ese

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

tipo MIME, el servidor Web invoca la aplicación filtro, la cual procesa el fichero especificado en la URL y envía la respuesta (usualmente texto HTML) de vuelta al cliente.

- **Aplicaciones Autorizadora (*Authorizer*)**: Una aplicación FastCGI *authorizer* recibe la información en una cabecera HTTP y genera una decisión en función de la petición. Para etiquetar una aplicación FastCGI con el papel de *authorizer*, el administrador del sistema nombra la aplicación dentro del archivo de configuración del servidor, usando una directiva llamada *AuthorizeRegion*.

  Cuando el servidor recibe una petición de cliente a una URL con el criterio *AuthorizeRegion*, el servidor Web llama a la aplicación FastCGI *authorizer*. Si la aplicación concede la autorización (por retorno de una respuesta con código 200 OK), el servidor Web reactiva la ejecución de comandos en la sección *AuthorizeRegion*. Si la aplicación deniega la autorización (por respuesta con otro código), el servidor Web para el proceso de comandos siguientes en la sección *AuthorizeRegion*, y devuelve la respuesta de la aplicación FastCGI al cliente.

Como ventajas más significativas se pueden citar las siguientes:

- Facilita la realización de aplicaciones persistentes entre peticiones de clientes y posibilita mantener el estado entre distintas llamadas de un mismo cliente.
- FastCGI permite a las aplicaciones residir en sistemas remotos.
- FastCGI proporciona una flexibilidad adicional a las aplicaciones, mediante el soporte explícito para autentificación de clientes y filtro de entradas.
- Soporta el inicio de múltiples copias de una aplicación para manejo concurrente de peticiones.

2.3 Los Lenguajes de Script

Dentro de la Web, los lenguajes de script ocupan un lugar muy importante, tanto en los servidores Web como en los navegadores. Un lenguaje de script es un lenguaje interpretado y que, por lo tanto, no necesita ser compilado por el programador antes de ser ejecutado. Los scripts se ejecutan sobre un intérprete, que es el responsable de compilarlos para su ejecución.
Generalmente los lenguajes de script forman parte de las páginas Web. Esto hace que sea muy sencillo crear páginas Web con este tipo de lenguajes, pues la creación de páginas dinámicas no varía en exceso con respecto a la creación de páginas estáticas [26-31].
Se pueden distinguir dos tipos de lenguajes de script: los lenguajes del lado del servidor (*Server-Side Scripts*) y los lenguajes del lado del cliente (*Client-Side Scripts*). Se suelen diferenciar, no sólo por el lado en el que se ejecutan, sino también por la forma en que actúan.

2.3.1 Scripts del Lado del Servidor

El uso de comandos o programas en el servidor para generar las páginas Web, es decir, el modelo CGI, tiene el problema de que el proceso de creación de documentos suele resultar poco intuitivo y engorroso. Los lenguajes de script son una buena alternativa que facilita el proceso de creación y mantenimiento de páginas Web.
Los lenguajes de script permiten crear documentos de un modo natural, incrustando en ellos el código para la generación de las partes dinámicas del documento, que no se crean hasta el momento de servir el documento al cliente. El documento que llega finalmente al cliente es un documento de texto libre del código de los scripts.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
<html>
<head>
<title>Hoy</title>
</head>
<body>
    Hoy es el día: <?php echo date('d/m/Y'); ?>
</body>
</html>
```

En el siguiente código vemos un ejemplo de una página que emplea scripts. Concretamente se emplea un script de PHP para mostrar la fecha actual en el documento. Como se puede observar el documento se compone como un documento HTML estático, con la diferencia de que incluye un pequeño código de script incrustado.

El uso de lenguajes de script en el servidor resulta transparente para el cliente. Aunque las páginas Web se almacenan en el servidor con el código de los scripts incrustado, antes de ser entregadas al cliente son preprocesadas y sus scripts son interpretados para generar el contenido final. Por lo tanto, el código fuente de los scripts nunca llegará al cliente.

La Figura 5 muestra el flujo de ejecución típico en un servidor Web con scripts:

1. El cliente solicita una página Web.
2. El servidor recupera la página asociada a la consulta del cliente. Esta página está formada por código HTML y código de scripts.
3. Un analizador sintáctico se encarga de detectar el código de los scripts incrustado. Este código es pasado por un intérprete que lo ejecutará y añadirá cualquier texto producido a la página Web.
4. La página generada será una página en HTML normal.
5. La página generada se envía al cliente, para el cual el procesado de la página con código de scripts ha sido transparente.

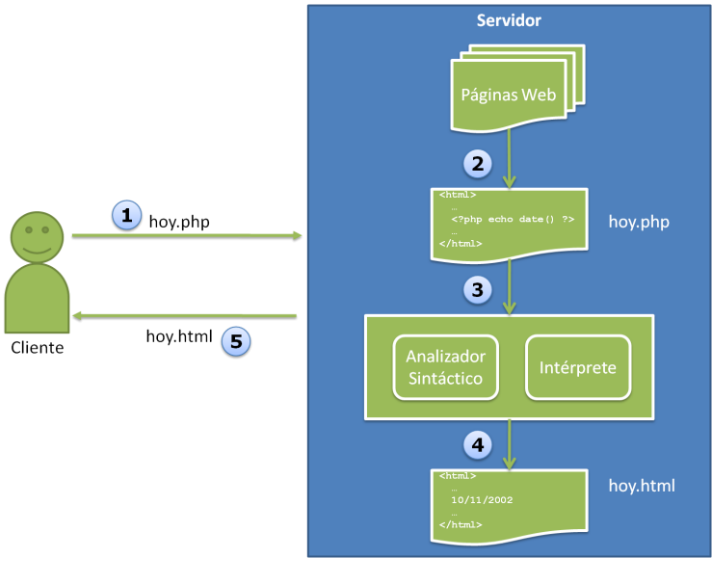


Figura 5: Flujo de ejecución típico en un servidor Web con scripts.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

2.3.2 Scripts del Lado del Cliente

La función de los scripts del lado del cliente es distinta de la de los scripts del lado del servidor. Mientras que los scripts del lado del servidor se emplean para generar dinámicamente los documentos servidos, los scripts del lado del cliente permiten modificar la estructura del documento una vez es mostrado en el navegador. También permiten realizar otras acciones adicionales tales como gestionar eventos, mostrar diálogos, etc.
Este tipo de scripts está muy relacionado con el DHTML (*Dynamic HTML*), que es término empleado para describir la combinación de un conjunto de tecnologías que permiten aumentar la interactividad entre el usuario y las páginas Web, así como modificar dinámicamente la estructura del documento cuando éste se encuentra en el cliente. Son cuatro las tecnologías que se combinan en el DHTML:

- **Un lenguaje de marcado**: Es decir HTML o XHTML, que define la estructura del documento.
- **Un lenguaje de script del lado del cliente**: Que controla la interacción con el usuario y permite modificar dinámicamente la estructura del documento. Aunque existen otros, el lenguaje actualmente más extendido y soportado por los navegadores es Javascript. Javascript es un dialecto del estándar ECMAScript.
- **Un lenguaje de definición de presentación**: Que define cómo se debe mostrar el documento en el navegador. El lenguaje de este tipo más extendido es el CSS (*Cascading Style Sheets*). Separando la estructura de un documento de su presentación, se consigue que un mismo documento se pueda ver de distintas formas y que las modificaciones visuales (cambio de color, ocultar una sección, etc.) no afecten a la estructura del documento.
- **Un modelo de representación en forma de objetos del documento**: El DOM (*Document Object Model*) es un estándar que define una interfaz independiente de la plataforma y del lenguaje que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, estructura y estilo de los documentos. En el DOM los documentos son representados en forma de árbol de objetos. La ventaja de representar un documento en forma de objetos es que resulta mucho más fácil de manipular por parte de los lenguajes de script.

La práctica totalidad de los navegadores actuales soporta estas tecnologías, por lo que son muy explotadas por los programadores. Cuando un documento llega a un cliente, el navegador lo interpreta y genera su representación en forma de árbol DOM. El árbol DOM se combina con los ficheros CSS que pudiera haber para *renderizar* finalmente la página Web. Cuando la página se ha mostrado al usuario, los scripts incluidos en ella permiten gestionar cierto tipo de interacciones entre el usuario y la página. Los scripts permiten incluso modificar la estructura

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

del árbol DOM o las propiedades de sus nodos, provocando que la página o parte de ella se *renderice* de nuevo.
La Figura de la izquierda muestra el flujo de *renderización* de una página Web dinámica en el navegador. Como se puede observar, una vez recibida la página Web se genera el árbol DOM y el navegador sólo trabajará sobre éste [32-28].

2.4 Asynchronous JavaScript and XML (AJAX)

A medida que la Web fue evolucionando, las páginas Web se volvieron cada vez más complejas. La aparición de los lenguajes de scripts, tanto del lado del servidor como del lado del cliente, hizo que las páginas pasaran de ser simples documentos relacionados a convertirse en aplicaciones cliente/servidor (aplicaciones Web) con funcionalidades similares a las aplicaciones de escritorio. Sin embargo las aplicaciones Web tenían un grave problema respecto a las aplicaciones de escritorio en la interacción del usuario con la aplicación. Cuando un cliente de una aplicación Web necesitaba información del servidor se debía recargar toda la página Web para poder mostrar esta nueva información. Esto hacía que la interacción entre el usuario y la aplicación se viese continuamente interrumpida y también hacía complicado para los desarrolladores mantener el estado de todos los elementos de las aplicaciones.
La solución a estos problemas vino de la mano de AJAX (*Asynchronous JavaScript and XML*). AJAX es un conjunto de tecnologías que permite que las peticiones entre los clientes y el servidor se realicen de modo asíncrono, evitando las interrupciones en la interacción entre el cliente y el navegador.

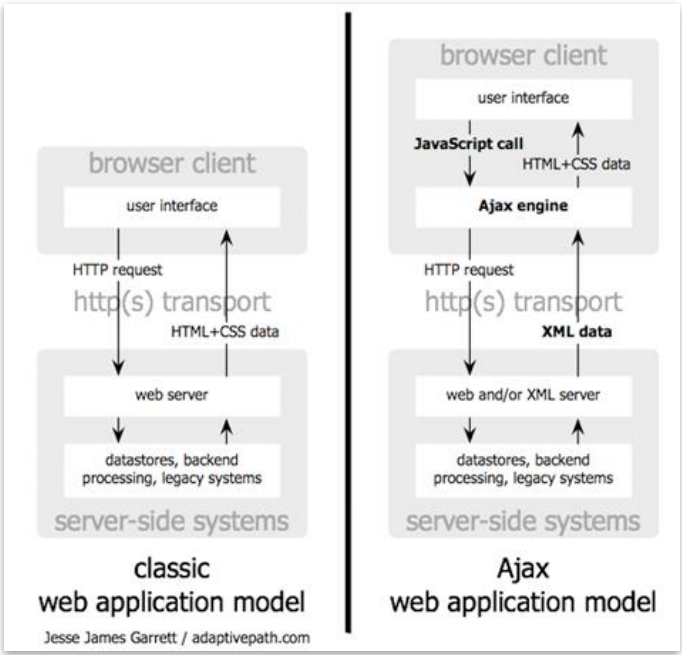


Figura 6: Modelo tradicional de aplicaciones Web (izqda) comparado con el modelo AJAX (dcha).

AJAX introduce un nuevo modelo de aplicación Web, al modificar el modo en el que los navegadores Web interactúan con el servidor. En el modelo clásico el cliente interactúa con la interfaz presentada por el navegador y, cada vez que el navegador necesita información del servidor, el cliente debe esperar a que el servidor responda y una nueva página sea cargada. Mientras el navegador espera la respuesta del servidor, el cliente no podrá interactuar con la aplicación.

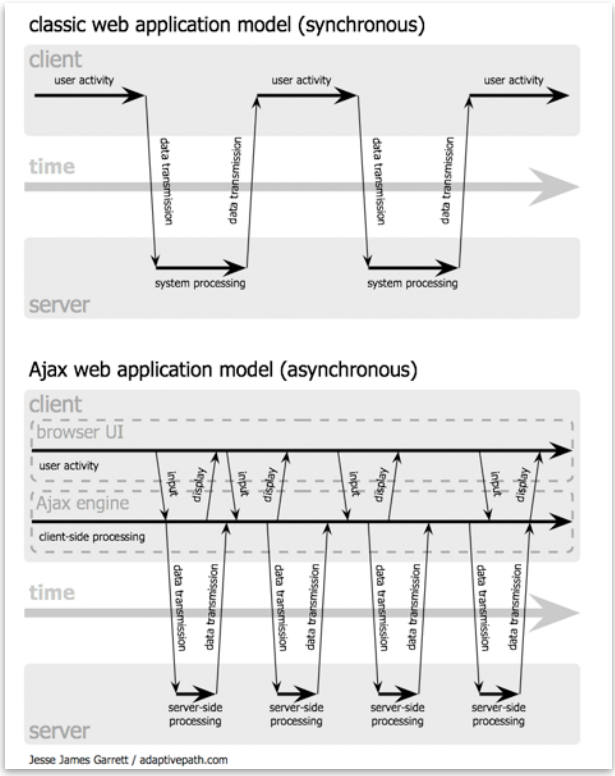
ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239

Con el modelo AJAX el navegador puede solicitar información del servidor de forma indirecta, mediante el uso de bibliotecas Javascript. El usuario interactúa igualmente con la interfaz presentada por el navegador, pero se introduce una capa intermedia entre la interfaz y el servidor en la que se sitúa el motor AJAX (*AJAX engine*). El motor AJAX es una pieza de código escrita con Javascript que permite realizar peticiones HTTP al servidor de modo asíncrono. Cada vez que se hace una petición a través del motor AJAX, el mismo motor se encargará de esperar y recibir la respuesta del servidor. La respuesta suele incluir código (generalmente en XML) que puede ser incorporado directamente a la aplicación Web o ser tratado de algún modo para mostrar nueva información.
La Figura inferior permite apreciar la diferencia entre el modelo de ejecución clásico (síncrono) y el modelo de ejecución con AJAX (asíncrono).
En el modelo clásico la actividad del usuario se ve interrumpida cada vez que existe una comunicación entre el navegador y el servidor. En el modelo AJAX las peticiones al servidor son delegadas al motor de AJAX que se hace responsable de las comunicaciones con el servidor, evitando que se interrumpa la interacción entre el usuario y la aplicación.
De este modo las páginas Web pasan de ser un conjunto de documentos relacionados a actuar como un único documento dinámico e interactivo.

2.4.1 La Tecnología de AJAX

Como ya se ha comentado, AJAX es un conjunto de tecnologías. Básicamente está formado por las tecnologías presentes en el DHTML más dos elementos: un método de intercambio asíncrono de información entre el cliente y el servidor y un formato para la información enviada desde el servidor.
El método más común de intercambio de información es usar el objeto XMLHttpRequest (XHR), perteneciente al API DOM. El objeto XHR implementa una interfaz que contiene todas las funcionalidades de un cliente HTTP (envío de formularios, cargar datos desde el servidor, etc.). Pese a su nombre, el objeto XHR soporta cualquier formato de texto (incluyendo XML) y, además, permite realizar peticiones en HTTP y en HTTPS. Este objeto es el que actúa como motor de AJAX.



El formato empleado por el servidor para enviar información varía dependiendo de las necesidades de la aplicación. Aunque se puede emplear cualquier formato basado en texto, los más comunes son XML, texto con formato HTML, texto plano y texto en notación JSON (*JavaScript Object Notation*). El formato JSON permite representar estructuras y arrays asociativos en texto plano, de forma que se pueden ser leídos fácilmente por una persona o ser convertidos en tipos de dato JavaScript.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

La ventaja de emplear XML o HTML como formato de intercambio es que la información recibida puede ser incrustada directamente en la página Web, añadiéndola al árbol DOM o sustituyendo un nodo del mismo. La información recibida en formato JSON tiene la ventaja de que puede ser añadida directamente como código Javascript para ser manipulada.

2.5 Servicios Web

Los servicios Web son otro ejemplo de evolución y transformación de la Web. En este caso la Web sirvió para solventar un problema de comunicación entre aplicaciones.
Desde el nacimiento en 1975 del *Electronic Data Interchange* (EDI) han sido varios los intentos de crear un estándar de comunicaciones entre aplicaciones sobre una red. Algunos de los más destacados son *Common Object Request Broker Architecture* (CORBA), *Distributed Component Object Model* (DCOM), *Remote Procedure Call* (RCP) o *Java Remote Method Invocation* (RMI). Aunque todas estas tecnologías tuvieron (y siguen teniendo) éxito en determinados ámbitos, ninguna de ellas consiguió estandarizar la comunicación entre aplicaciones a través de la red. Problemas tales como la complejidad, el coste, la flexibilidad o el soporte por parte de la industria evitaron una mayor aceptación de estas tecnologías.
La llegada de la Web al mundo empresarial consiguió que las empresas adoptasen el protocolo HTTP sobre TCP/IP como un estándar de comunicaciones. Por lo tanto, ya existía un medio de comunicaciones aceptado por las empresas y tan sólo faltaba un sistema de mensajería y encapsulación de la información que aceptasen todas las empresas.
XML se convirtió en la pieza que faltaba para completar los servicios Web. Fueron varios los intentos por conseguir un protocolo estándar de comunicaciones entre procesos pero, finalmente, sería SOAP (*Simple Object Access Protocol*), desarrollado por Microsoft, quien se alzaría como el protocolo estándar más aceptado por las empresas. En 1999 Microsoft presentó SOAP 1.0 a IBM y otras compañías. El año siguiente, con la colaboración de IBM, se presentó SOAP 1.1. Pronto fue ganando aceptación entre las empresas, principalmente por ser independiente de las plataformas, flexible y de propósito general.
IBM y Microsoft continuaron trabajando juntos para desarrollar esta tecnología y, a finales del año 2000, presentaron dos estándares más que completaban los servicios Web: el *Web Services Description Language* (WSDL) y el *Universal Description, Discovery, and Integration* (UDDI).
WSDL es un lenguaje que proporciona un modelo para la descripción de servicios Web. En WSDL, los servicios Web se describen como un conjunto de puntos finales (*endpoints*) o puertos. Un puerto se define asociando una dirección de red con un enlace reusable, que generalmente es una función invocable. Los mensajes son la descripción abstracta de los datos que se intercambian. Para cada puerto se define el formato de los mensajes que puede recibir, es decir, el tipo de datos que puede recibir.
UDDI es un registro de servicios Web independiente de la plataforma basado en XML. Permite que las empresas publiquen y descubran servicios y que definan cómo los servicios o aplicaciones interactúan sobre Internet. Un registro UDDI está compuesto por tres elementos:

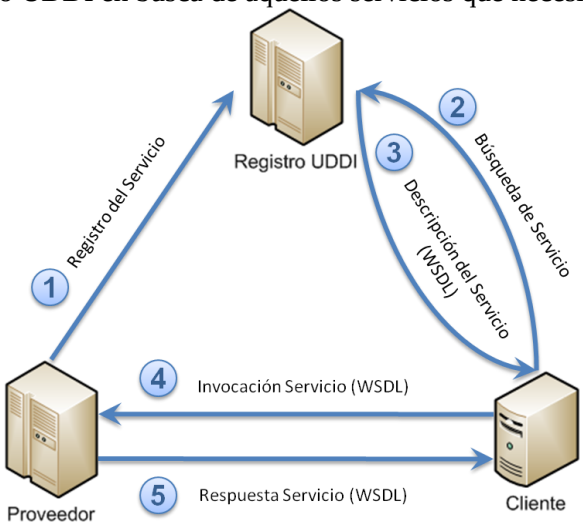
- **Páginas Blancas**: Contienen información sobre la dirección, contacto y otros identificadores conocidos de los servicios.
- **Páginas Amarillas**: Contienen información categorizada de los servicios basada en taxonomías estandarizadas.
- **Páginas Verdes**: Contienen información sobre los servicios proporcionados por las empresas.

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

UDDI está pensado para ser accedido mediante mensajes SOAP y para proporcionar acceso a documentos WSDL que describen el acceso a los servicios Web.

La Figura inferior muestra el funcionamiento típico de los servicios Web. Cuando un proveedor crea un nuevo servicio Web debe publicarlo para darlo a conocer a los clientes potenciales. Para ello emplea un directorio UDDI en el que describe y categoriza su servicio. Los clientes acceden al directorio UDDI en busca de aquellos servicios que necesiten.



Una vez localizado el servicio deseado se obtiene la descripción del mismo en WSDL almacenada directorio. Con esta descripción el cliente sabe cuál es la ubicación del servicio y cuál debe ser el formato del mensaje que debe emplear para comunicarse con él. El cliente invoca el servicio en el proveedor y este le responde con el resultado de la ejecución.

En el caso de que el cliente conozca el servicio Web, la invocación es directa, sin que tenga que consultar el registro UDDI.

Además de las ventajas ya comentadas, una de las grandes ventajas de los servicios Web es que hacen uso del puerto 80 (el puerto HTTP). Esto evita problemas de configuración de firewalls, pues normalmente este puerto no se bloquea al ser el usado por los servidores Web, el tipo de servidor más común en las empresas [39-46].

2.6 Frameworks Web

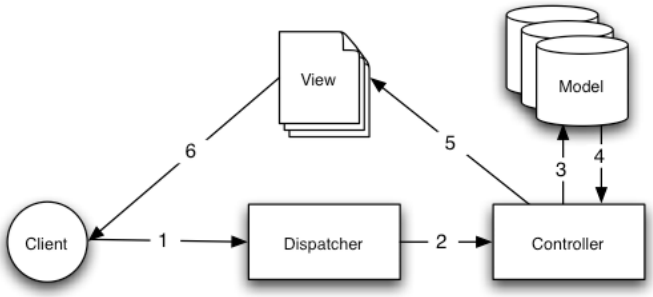
La gran cantidad de desarrollos Web producidos en los últimos años ha hecho que surjan muchos frameworks Web con el objetivo de reducir los tiempos de desarrollo reutilizando la mayor cantidad de código posible. Aunque es un término muy genérico, un framework suele proporcionar tres elementos:

- **Una arquitectura**: Definen la arquitectura de la aplicación. Generalmente la aplicación desarrollada es sólo una parte de esta arquitectura.
- **Un mecanismo de control**: El framework controla el flujo de ejecución de la aplicación.
- **Un conjunto de bibliotecas**: Con APIs y herramientas para realizar las tareas más comunes.

Los frameworks pueden verse como una evolución de las bibliotecas, en los que no sólo se reutilizan funciones, sino que también se reutiliza una arquitectura y un flujo de ejecución.
Cuando se desarrolla una aplicación empleado un framework, la aplicación se desarrolla sobre el framework, de forma que al final la aplicación y el framework son uno solo. Se puede decir que

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

los framework actúan como una base de desarrollo de aplicaciones, proporcionando un punto de inicio avanzado.
Existen muchos tipos de software, algunos de ellos de propósito general y otros con un propósito específico. Por ello resulta muy difícil definir unas características comunes a todos ellos. Entre las características más típicas se encuentra el uso de una arquitectura Modelo-Vista-Controlador (MVC). Esta arquitectura se caracteriza por dividir la aplicación en tres partes: los datos (Modelo), la lógica de la aplicación (Controlador) y la presentación de los datos (Vista).
La Figura siguiente muestra la ejecución típica de un framework Web con una arquitectura MVC. El cliente hace una solicitud al servidor que llega a un despachador que decide a que controlador redirigir la solicitud. El controlador recibe la solicitud y recupera la información necesaria a través del modelo. Una vez ha recuperado la información necesaria se pasa a la vista que da formato a la información. La información recuperada y formateada según la vista es devuelta al usuario en la respuesta final.
Otras características típicas son la gestión de la seguridad (autenticación y autorización), acceso a bases de datos, mapeo de datos a objetos, mapeo de URLs, gestión de cachés, uso de sistemas de plantillas, integración de AJAX, publicación de servicios Web, etc.



2.6.1 Content Management Systems (CMS)

Los CMS son un tipo de frameworks de propósito específico que han cobrado mucha popularidad últimamente. Este tipo de frameworks suelen consistir en una interfaz que permite controlar una o varias bases de datos donde se aloja el contenido del sitio.

El contenido y la presentación del mismo se manejan de forma independiente, de modo que resulta muy sencillo cambiar la presentación de la página Web en cualquier momento sin afectar al contenido.

Suelen tener un mecanismo de control de publicación de los contenidos, pudiéndose mantener ocultos al público ciertos contenidos hasta que, por ejemplo, sean revisados y aceptados. Es típico encontrar páginas basadas en CMS en las que existen varios redactores, que generan los contenidos, y un cuerpo de revisores, que revisan y publican los documentos.

2.7 Servidores de Aplicaciones

Una aplicación empresarial es un tipo de aplicación que, por su naturaleza, posee serie de requisitos especiales. Son aplicaciones que hacen un alto uso de la red y que, al tener que dar servicio a muchos usuarios y resultar críticas para muchos de ellos, se necesita que sean aplicaciones con una alta disponibilidad evitar caídas de los servicios, que sean fácilmente escalables para poder hacer frente a un aumento de la demanda, que sean seguras para evitar accesos incontrolados, que puedan balancear la carga para repartir el trabajo y evitar sobrecargas, que sean tolerantes a fallos para no romper el servicio por un error y que tengan otras características asociadas a mantener el servicio a los usuarios activo.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

El problema que existe en el desarrollo de este tipo de aplicaciones es que muchos de los recursos se tienen que dedicar a conseguir que las aplicaciones cubran las necesidades descritas. Esto puede hacer que los recursos dedicados a estos requisitos no funcionales sean iguales o incluso superiores a los dedicados a los requisitos funcionales de la aplicación. En lugar de centrarse los recursos en desarrollar la lógica de negocio asociada al objetivo central de un proyecto, muchos de los recursos se desvían a objetivos secundarios.

Para poder centrarse en el objetivo principal de un proyecto, muchas empresas subcontrataban la parte asociada a los requisitos secundarios. Esto suponía un gran coste presupuestario, pues las empresas subcontratadas solían estar muy especializadas y tener personal muy cualificado, y también suponía un gran coste de tiempo, pues la integración y pruebas de todas las partes del proyecto resultaba muy compleja.

Con el objetivo de solventar este problema nacieron los servidores de aplicaciones. Un servidor de aplicaciones es un tipo especial de framework con un gran número de funcionalidades, muchas de ellas destinadas a cubrir los requisitos de las aplicaciones empresariales. A diferencia de los frameworks descritos anteriormente, en los servidores de aplicaciones, el framework no se integra como parte de las aplicaciones, sino que las aplicaciones se *despliegan* sobre el framework.

Las aplicaciones son un conjunto de recursos (clases, bibliotecas, imágenes, ficheros de configuración, etc.) que se distribuyen de forma conjunta y, generalmente, empaquetadas. Cuando una aplicación se despliega sobre un servidor, lo que se hace es que sus distintas partes se añaden a los *contenedores* del servidor de aplicaciones. Los contenedores le proporcionan a cada parte de la aplicación un conjunto de funcionalidades, necesarias para que la aplicación pueda funcionar. Normalmente las aplicaciones contienen un descriptor de despliegue que le indica al servidor como se debe desplegar.

En este tipo de servidores las aplicaciones no tienen sentido sin el servidor de aplicaciones, pues existen cierto tipo de funcionalidades que las aplicaciones delegan en los servidores que las contienen y sin las cuales no podrían ejecutarse.

El origen del nombre de servidor de aplicaciones se encuentra en que, así como los servidores Web sirven páginas Web, cada una de ellas formada por un conjunto de documentos, los servidores de aplicaciones sirven aplicaciones, cada una de ellas formada por un conjunto de recursos.

2.7.1 Java Platform, Enterprise Edition (JavaEE)

JavaEE es el principal representante de los servidores de aplicaciones. JavaEE no es un servidor en sí mismo, sino que son un conjunto de especificaciones que Oracle propone para la creación de servidores de aplicaciones en Java. Cabe destacar que la especificación de JavaEE nació de la mano de Sun Microsystems (compañía que creó el lenguaje de programación Java), empresa que fue adquirida recientemente por Oracle.

Aunque la especificación inicial fue desarrollada completamente por Sun Microsystems, a partir de la versión 1.3 (actualmente se encuentra en la versión 1.5, rebautizada como 5) la especificación de JavaEE se desarrolla bajo Java Community Process, que son procesos comunitarios en los que participan distintas entidades.

No existe el servidor de aplicaciones JavaEE, sino que existen servidores de aplicaciones que siguen la especificación JavaEE. Oracle posee un proceso de certificación que comprueba que un servidor de aplicaciones cumpla con la especificación. Con la certificación de los servidores de aplicaciones se garantiza que si una aplicación puede ser desplegada y funciona en un servidor de aplicaciones certificado, entonces también puede ser desplegada y funcionar en cualquier otro servidor de aplicaciones certificado.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

La especificación de Java EE está formada por varias otras especificaciones de distintos tipos, como pueden ser APIs (JDBC, RMI, JMS, JNDI, etc.) o componentes (Servlets,JSP, EJB, etc.). Todas ellas destinadas a proporcionar funcionalidades a las aplicaciones empresariales para manejar transacciones, seguridad, escalabilidad, gestión de componentes, balanceo de carga, etc.

Arquitectura

Las aplicaciones Java EE generalmente tienen una arquitectura de tres capas (cliente, lógica de negocio y datos), aunque también es habitual encontrar arquitecturas de n-capas. El aumento del número de capas puede responder a diversos motivos tales como decisiones de diseño, distribución de funciones, reducción del acoplamiento, etc. Muchas veces este aumento de capas consiste en una fragmentación de alguna de las tres capas básicas.

La Figura 7 muestra las tres capas básicas de las aplicaciones en JavaEE y la relación existente entre ellas.

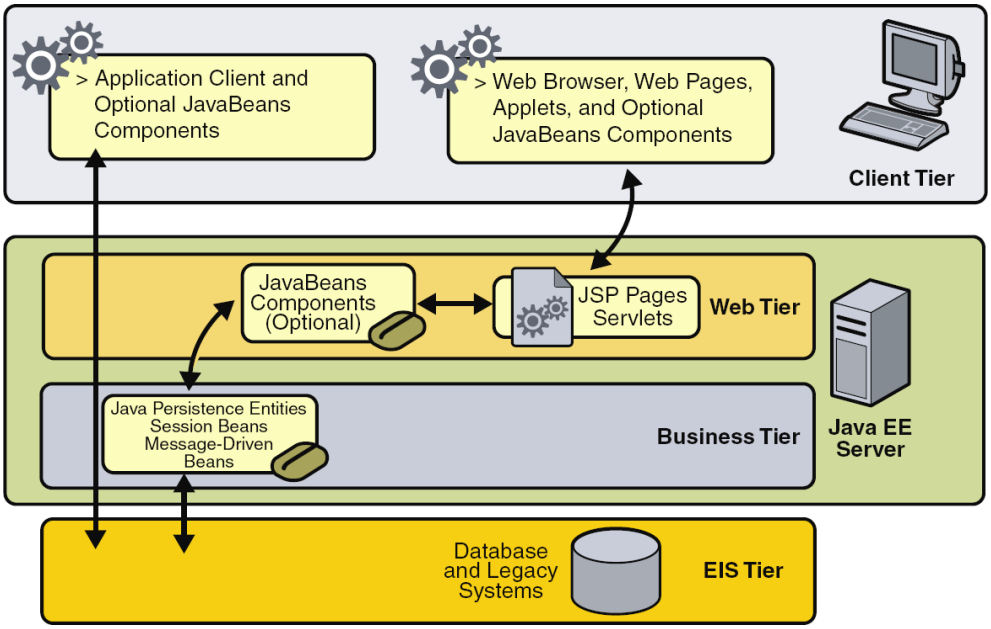


Figura 7: Arquitectura en tres capas de Java EE.

La capa superior es la capa cliente, en la que se ubican aplicaciones, habitualmente ligeras, con las interfaces gráficas a través de las que el cliente puede interactuar con la aplicación. Existe una gran libertad en este tipo de aplicaciones que pueden ser aplicaciones Web, aplicaciones de escritorio, aplicaciones móviles, aplicaciones incrustadas, etc.

La capa intermedia contiene la lógica de negocio y la parte de presentación Web que corresponde al servidor Web. En esta capa es donde se ubican los servidores JavaEE, dentro de los cuales existen dos contenedores: un contenedor Web y un contenedor de *Enterprise Java Beans* (EJB). El contenedor Web no es más que un servidor Web capaz de interpretar lenguaje de scripts JSP (*Java Server Pages*) y de manejar Servlets (objetos de Java que generan respuestas a peticiones HTTP). El contenedor de EJBs es quien contiene la lógica de negocio en forma de pequeños componentes modulares y reutilizables denomidados “Beans”. Existen distintos tipos de EJBs, cada uno de ellos con un objetivo distinto como puede ser la gestión de la persistencia, la ejecución de acciones o la gestión de mensajes. Por lo tanto, una aplicación empresarial en un servidor Java EE tiene forma de EJBs y de páginas JSP o Servlets.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

La última capa es la capa de los Sistemas Empresariales de Información. Es la capa que contiene la información con la que trabaja la aplicación. Esta información puede presentarse en forma de bases de datos, ERPs, sistemas de infraestructura empresariales, etc.

2.8 Rich Internet Applications (RIA)

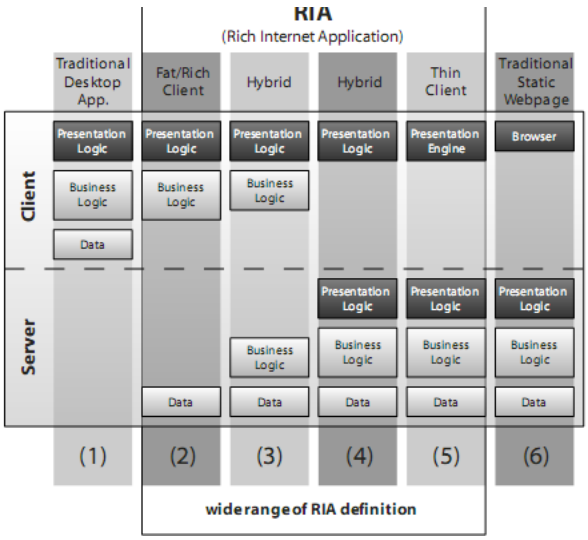
El término RIA hace referencia a un tipo de aplicación que hace un alto uso de Internet. Es un término muy genérico, quizás en parte por su juventud, bajo el que se pueden englobar muchos tipos de aplicaciones. La Figura siguiente nos da una idea de los distintos tipos de aplicaciones que se pueden considerar RIA.
Las características que mantienen en común las aplicaciones RIA son la existencia de una arquitectura Cliente/Servidor, el almacenamiento de la información en el servidor y una capa de presentación que actúa como en las aplicaciones de escritorio.
La idea detrás de las aplicaciones RIA es conseguir una convergencia entre las aplicaciones Web y las aplicaciones de escritorio. Para ello, las tecnologías de desarrollo de aplicaciones RIA suelen proporcionar métodos sencillos (aunque muy potentes) de acceso a información remota en aplicaciones de escritorio. También es común encontrar en estas tecnologías mecanismos para que una aplicación pueda actuar tanto como aplicación de escritorio, como aplicación Web.
Pese a que la aparición de este tipo de aplicaciones es muy reciente, existen una gran cantidad de plataformas y herramientas para su desarrollo y parece que el mercado se ha volcado en proporcionar apoyo a este tipo de aplicaciones [47-54].

2.9 Cloud Computing

El Cloud Computing es una tecnología muy reciente en la que se ofrecen recursos como servicio a través de Internet. Su nombre viene de la metáfora de que toda la información y lógica de las aplicaciones pasan a estar en Internet, denominada "nube" por su abstracción y complejidad. Los recursos que se proporcionan suelen ser tres, cada uno de los cuales da nombre a un tipo de servicio:

- **Infrastructure as a Service (IaaS)**: Consiste en proporcionar infraestructura como un servicio. En vez de comprar servidores, software, espacio para centros de datos o equipo de red, los clientes pueden comprar estos recursos como un servicio externalizado. Generalmente el cliente paga por la cantidad de recursos empleados, por lo que el coste depende de su uso de los recursos. Es una evolución del alojamiento Web y de los servidores privados.

Figura 8: Ámbito de las aplicaciones RIA.



- **Platform as a Service (PaaS)**: En este caso el proveedor proporciona una plataforma sobre la que el cliente

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

puede desarrollar sus aplicaciones. Suelen incluir facilidades para diseñar una aplicación, desarrollarla, testearla, desplegarla y alojarla.

- **Software as a Servide (SaaS)**: El proveedor de software como servicio permite a sus clientes el uso de servicios bajo demanda. El cliente debe pagar una licencia por el uso de estos servicios.

En los tres casos el objetivo principal es centralizar y agrupar los recursos para reducir costes. Además de esta ventaja, existen otras tales como la independencia de la ubicación, mayor fiabilidad, mejor escalabilidad, mejor seguridad, mayor sostenibilidad, etc.
La mayor parte de estos beneficios tienen su origen en la concentración de recursos y en la alta especialización de las compañías proveedoras de Cloud Computing en gestionar y mantener grandes infraestructuras.

3 Bibliografía

Historia del W3C - http://www.w3c.es/consorcio/historia
W3C: Protocols - http://www.w3.org/Protocols/
W3C: Web Services - http://www.w3.org/2002/ws/
W3C: Web Services Glossary- http://www.w3.org/TR/ws-gloss/
W3C: XMLHttpRequest Level 2 - http://www.w3.org/TR/XMLHttpRequest2/
CGI - http://hoohoo.ncsa.uiuc.edu/cgi/
Fast CGI - http://www.fastcgi.com/
AJAX: A New Approach to Web Applications -
Autor: *Jesse James Garrett*
http://adaptivepath.com/ideas/essays/archives/000385.php
From EDI To XML And UDDI: A Brief History Of Web Services
Autor: *Jason Levitt*
http://www.informationweek.com/news/software/development/showArticle.jhtml?articleID=6506480
Rich Internet Applications (RIA): A Convergence of User Interface Paradigms of Web and Desktop Exemplified by JavaFX
Autor: *Florian Moritz*
http://www.flomedia.de/diploma/documents/DiplomaThesisFlorianMoritz.pdf
Wikipedia
http://en.wikipedia.org/wiki/Ajax_%28programming%29
http://en.wikipedia.org/wiki/XMLHttpRequest
http://en.wikipedia.org/wiki/Web_service
http://en.wikipedia.org/wiki/Electronic_Data_Interchange
http://en.wikipedia.org/wiki/Web_service
http://en.wikipedia.org/wiki/Java_ee
http://en.wikipedia.org/wiki/Rich_internet_applications
http://en.wikipedia.org/wiki/Cloud_computing
http://en.wikipedia.org/wiki/Infrastructure_as_a_service
http://en.wikipedia.org/wiki/Platform_as_a_service
http://en.wikipedia.org/wiki/Software_as_a_service

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

References

1. António Pereira, Filipe Felisberto, Luis Maduro, Miguel Felgueiras (2012). Fall Detection on Ambient Assisted Living using a Wireless Sensor Network. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1
2. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. Information Sciences, 180(10), 2029–2043. https://doi.org/10.1016/j.ins.2009.12.032
3. Canizes, B., Pinto, T., Soares, J., Vale, Z., Chamoso, P., & Santos, D. (2017). Smart City: A GECAD-BISITE Energy Management Case Study. In 15th International Conference on Practical Applications of Agents and Multi-Agent Systems PAAMS 2017, Trends in Cyber-Physical Multi-Agent Systems (Vol. 2, pp. 92–100). https://doi.org/10.1007/978-3-319-61578-3_9
4. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
5. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
6. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
7. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
8. Chamoso, P., de La Prieta, F., Eibenstein, A., Santos-Santos, D., Tizio, A., & Vittorini, P. (2017). A device supporting the self-management of tinnitus. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 10209 LNCS, pp. 399–410). https://doi.org/10.1007/978-3-319-56154-7_36
9. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
10. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
11. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
12. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
13. Choon, Y. W., Mohamad, M. S., Deris, S., Illias, R. M., Chong, C. K., Chai, L. E., … Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. PLoS ONE, 9(7). https://doi.org/10.1371/journal.pone.0102744
14. Corchado, J. M., & Aiken, J. (2002). Hybrid artificial intelligence methods in oceanographic forecast models. Ieee Transactions on Systems Man and Cybernetics Part C-Applications and Reviews, 32(4), 307–313. https://doi.org/10.1109/tsmcc.2002.806072
15. David Griol, Jesús García-Herrero, José Manuel Molina (2013). Combining heterogeneous inputs for the development of adaptive and multimodal interaction systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
16. Fdez-Riverola, F., & Corchado, J. M. (2003). CBR based system for forecasting red tides. Knowledge-Based Systems, 16(5–6 SPEC.), 321–328. https://doi.org/10.1016/S0950-7051(03)00034-0
17. Fernández-Riverola, F., Díaz, F., & Corchado, J. M. (2007). Reducing the memory size of a Fuzzy case-based reasoning system applying rough set techniques. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 37(1), 138–146. https://doi.org/10.1109/TSMCC.2006.876058
18. Fyfe, C., & Corchado, J. (2002). A comparison of Kernel methods for instantiating case based reasoning systems. Advanced Engineering Informatics, 16(3), 165–178. https://doi.org/10.1016/S1474-0346(02)00008-3
19. Fyfe, C., & Corchado, J. M. (2001). Automating the construction of CBR systems using kernel methods. International Journal of Intelligent Systems, 16(4), 571–586. https://doi.org/10.1002/int.1024
20. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. Expert Systems with Applications, 41(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

21. García, O., Chamoso, P., Prieto, J., Rodríguez, S., & De La Prieta, F. (2017). A serious game to reduce consumption in smart buildings. In Communications in Computer and Information Science (Vol. 722, pp. 481–493). https://doi.org/10.1007/978-3-319-60285-1_41
22. Glez-Bedia, M., Corchado, J. M., Corchado, E. S., & Fyfe, C. (2002). Analytical model for constructing deliberative agents. International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications, 10(3).
23. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. BMC Bioinformatics, 10. https://doi.org/10.1186/1471-2105-10-187
24. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
25. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
26. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
27. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
28. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. Applied Mathematics and Computation, 270, 756–767. https://doi.org/10.1016/j.amc.2015.08.059
29. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. Expert Systems with Applications, 36(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
30. Méndez, J. R., Fdez-Riverola, F., Díaz, F., Iglesias, E. L., & Corchado, J. M. (2006). A comparative performance study of feature selection methods for the anti-spam filtering domain. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4065 LNAI, 106–120. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-33746435792&partnerID=40&md5=25345ac884f61c182680241828d448c5
31. Pablo Campillo-Sánchez, Juan Antonio Botía, Jorge Gómez-Sanza (2013). Development of Sensor Based Applications for the Android Platform: an Approach Based on Realistic Simulation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 1
32. Palomino, C. G., Nunes, C. S., Silveira, R. A., González, S. R., & Nakayama, M. K. (2017). Adaptive agent-based environment model to enable the teacher to create an adaptive class. Advances in Intelligent Systems and Computing (Vol. 617). https://doi.org/10.1007/978-3-319-60819-8_3
33. Sigeru Omatu, Hideo Araki, Toru Fujinaka, Mitsuaki Yano, Michifumi Yoshioka, Hiroyuki Nakazumi, Ichiro Tanahashi (2012). Mixed Odor Classification for QCM Sensor Data by Neural Network. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
34. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. International Journal of Ambient Computing and Intelligence, v 1, n 1(1), 15–26. https://doi.org/10.4018/jaci.2009010102
35. Sittón-Candanedo, I., Alonso, R. S., Corchado, J. M., Rodríguez-González, S., & Casado-Vara, R. (2019). A review of edge computing reference architectures and a new global edge proposal. Future Generation Computer Systems, 99, 278-294.
36. Angelo Costa, Stella Heras, Javier Palanca, Paulo Novais, Vicente Julián (2016). Persuasion and Recommendation System Applied to a Cognitive Assistant. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
37. David Griol, Jose M. Molina (2016). A proposal to manage multi-task dialogs in conversational interfaces. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
38. Marco Antonio Ameller, María Angélica González (2016). Minutiae filtering using ridge-valley method. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1
39. Elton S Siqueira, Patrick Cisuaka Kabongo, Tiancheng Li, Carla D. Castanho, Li Weigang (2016). On Chinese and Western Family Trees: Mechanism and Performance. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

40. Eduardo Facchini, Eduardo Mario Dias, Alexandre Pelegi Abreu, Maria Lídia Rebello Pinho Dias (2016). Brazil in Search of Transparency E-Gov. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1
41. Ana Oliveira Alves, Tiago Dias, David Silva (2015). A Real-Time, Distributed and Context-Aware System for Managing Solidarity Campaigns. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 2
42. Eduardo Mario Dias, Eduardo Facchini, Antônio Carlos De Moraes, Mauricio Lima Ferreira, Willian Reginato Este, Maria Lídia Rebello, Pinho Dias (2014). A Future Look. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
43. Carlos Alberto Ochoa, Lourdes Yolanda Margain, Francisco Javier Ornelas, Sandra Guadalupe Jiménez, Teresa Guadalupe Padilla (2014). Using multi-objective optimization to design parameters in electro-discharge machining by wire. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
44. Vanessa N. Cooper, Hisham M. Haddad, Hossain Shahriar (2014). Android Malware Detection Using Kullback-Leibler Divergence. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
45. Saverio Giallorenzo, Maurizio Gabbrielli, Fabrizio Montesi (2014). Service-Oriented Architectures: from Design to Production exploiting Workflow Patterns. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
46. Muhammad Amin Khan, Felix Freitag (2014). Sparks in the Fog: Social and Economic Mechanisms as Enablers for Community Network Clouds. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
47. Johannes Fähndrich, Sebastian Ahrndt, Sahin Albayrak (2014). Formal Language Decomposition into Semantic Primes. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
48. Merce Teixido, Tomás Palleja, Marcel Tresanchez, Davinia Font, Javier Moreno, Alicia Fernández, Jordi Palacín, Carlos Rebate (2013). Optimization of the virtual mouse HeadMouse to foster its classroom use by children with physical disabilities. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
49. Ana Karin Chávez Valdivia (2017). Between the Profiles Pay Per View and the Protection of Personal Data: the Product is You. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1
50. Pawel Pawlewski, Kamila Kluska (2017). Modeling and simulation of bus assembling process using DES/ABS approach. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1
51. Davide Carneiro, Daniel Araújo, André Pimenta, Paulo Novais (2016). Real Time Analytics for Characterizing the Computer User's State. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 4
52. Roussanka Loukanova (2016). Relationships between Specified and Underspecified Quantification by the Theory of Acyclic Recursion. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 4
53. David Griol, Jose Manuel Molina (2016). From VoiceXML to multimodal mobile Apps: development of practical conversational interfaces. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 3
54. Jean Louis Monino, Soraya Sedkaoui (2016). The Algorithm of the Snail: An Example to Grasp the Window of Opportunity to Boost Big Data. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 3

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc