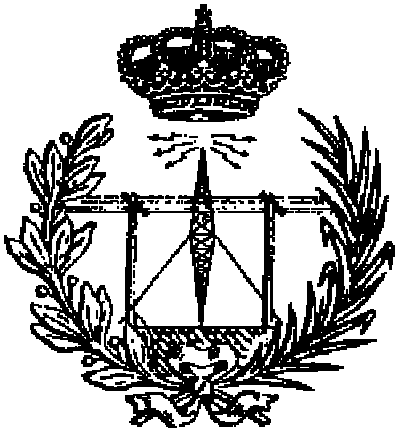
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN



PROYECTO FIN DE CARRERA

ENTORNO WEB PARA LA REALIZACIÓN DE PRÁCTICAS DE PROGRAMACIÓN EN C Y C++

TITULACIÓN : TELEMÁTICA

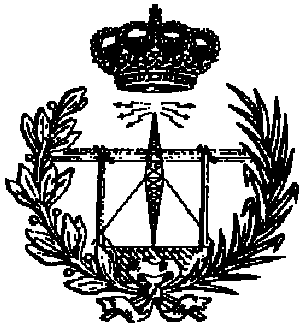
TUTOR : D. MIGUEL ÁNGEL QUINTANA SUÁREZ

AUTORA : Dª. ANA TERESA PULIDO CABRERA

FECHA : MAYO 2004

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN



PROYECTO FIN DE CARRERA

ENTORNO WEB PARA LA REALIZACIÓN DE PRÁCTICAS DE PROGRAMACIÓN EN C Y C++

Presidente: Secretario: Vocal:

Tutor: Autora:

NOTA :

TITULACIÓN : TELEMÁTICA

TUTOR : D. MIGUEL ÁNGEL QUINTANA SUÁREZ

AUTORA : Dª. ANA TERESA PULIDO CABRERA

FECHA : JUNIO 2004

Este Proyecto está dedicado a mis padres,
Agustín y Ana Teresa, sin cuyo apoyo, esfuerzo
y comprensión jamás hubiera sido posible.

ÍNDICE GENERAL

MEMORIA

PLIEGO DE CONDICIONES

PRESUPUESTO

ANEXOS

MEMORIA

1 Introducción

1.1 Introducción

El crecimiento continuo de la popularidad de Internet ha dado paso a la era de la información en las últimas décadas. La competencia entre los diferentes suministradores de acceso a Internet, la salida al mercado de nuevas tecnologías de acceso a la red con altas velocidades y la constante mejora de hardware y del software asociado, han creado el marco idóneo para que, hoy en día, medio mundo navegue por la red en busca de todo aquello que satisfaga sus necesidades.

Basada en la anterior premisa y con clara vocación docente, surge la aplicación objeto de este trabajo, que queda personalizada bajo el título *"Entorno web para la realización de prácticas de programación en C y C++"*.

1.2 Objetivos a cumplir y justificación

El objetivo principal es la creación de un entorno web orientado al alumnado, accesible desde cualquier lugar y plataforma, que permita la difusión de un entorno de trabajo fácil y cómodo desde el servidor que lo alberga.

Se pretende un entorno web que actúe como complemento al aprendizaje de material docente, siendo una herramienta para poner en prácticas los contenidos que progresivamente se van adquiriendo.

En un momento en que la popularización de Internet tiene como reflejo el hecho de que, no sólo se pueda obtener cualquier tipo de información relativa a cualquier campo en la red, sino que además se pueda conseguir diferentes aplicaciones según distintas finalidades, se hace necesario que el mundo académico no quede al margen de esta tendencia. Por ello, se plantea la necesidad de diversificar los métodos de enseñanza y, en definitiva, de acceso a la información que se quiere transmitir y un entorno de trabajo para poner en práctica toda esa información.

En este sentido la aplicación que nos ocupa tiene una clara justificación. Se complementan perfectamente los métodos clásicos de enseñanza, adaptándose a las nuevas tendencias y se favorece el uso de las últimas tecnologías.

La aplicación desarrollada es un servicio web, diseñado como complemento al aprendizaje de unos contenidos, que permite ponerlos en práctica. Está enmarcada dentro del *Departamento de Ingeniería Telemática* de la *ULPGC* y comprende la asignatura de *Programación Avanzada*. La aplicación tiene su implementación en un servidor perteneciente al departamento anteriormente mencionado y su accesibilidad es plena, tanto desde el entorno universitario, como desde fuera de él.

En cuanto al diseño web y aspecto gráfico, esta aplicación es un caso atípico en comparación con muchos de los sitios que se pueden encontrar en la red. Dada la naturaleza de los contenidos, tipo de usuarios y dedicación de la web, se ha elegido una interfase de manejo fácil e intuitivo que proporciona un acceso rápido y directo a las diferentes opciones, favoreciendo su uso por encima de otras consideraciones de carácter visual.

1.3 Descripción general de la aplicación

La idea principal consiste en que el usuario pueda crear programas desarrollados bajo los lenguajes de C y C++, poniendo así en práctica todos los conocimientos que haya podido ir adquiriendo poco a poco con respecto a dichos lenguajes. Al mismo tiempo que el usuario va creando sus propios ficheros, éste tiene una clara visualización de todos ellos,

proporcionando la aplicación una estructura de navegación, en forma de árbol desplegable, que permite el acceso al contenido de dichos ficheros. La estructura presentada del árbol siempre es definida por el propio usuario, a medida que crea diferentes directorios a partir de uno raíz impuesto por la propia aplicación. Además de poder crear, modificar y eliminar los archivos, el usuario tiene permiso para compilar, ejecutar y tener un seguimiento de los mismos.

Por otro lado, la aplicación muestra un entorno web, a través del cual, se permite su administración, teniendo en cuenta que siempre debe existir un administrador que lleve a cabo la gestión para asegurar el correcto funcionamiento.

1.4 Descripción del trabajo

El presente trabajo consta de cuatro partes diferenciadas: Memoria, Pliego de Condiciones, Presupuesto y Anexos.

La Memoria se organiza en nueve capítulos. El capítulo 1 (el presente) es de carácter introductorio e incluye los objetivos a cumplir con la realización de este Proyecto.

En el capítulo 2 se describen, de forma genérica, las diferentes herramientas utilizadas para la elaboración de la aplicación.

En los capítulos 3, 4 y 5 se detallan los tres tipos de análisis correspondientes a todo proyecto, el análisis previo, funcional y orgánico respectivamente.

Los capítulos 6 y 7 se corresponden con los manuales de uso de la aplicación. En cada uno de ellos se detallan todas las opciones y operaciones de uso, tanto para el bloque de gestión (Manual de administrador), como para el de trabajo (Manual de usuario).

En el capítulo 8 se redactan las conclusiones obtenidas con la realización de la aplicación, confirmando el cumplimiento, tanto del objetivo final del Proyecto, como de otros fines igualmente alcanzados.

En el capítulo 9, la Bibliografía, se detallan todas las fuentes bibliográficas empleadas en la realización del presente trabajo, asignando a cada cita los autores del libro o artículo, el título, la revista o editorial.

En el Pliego de Condiciones se especifican los requisitos mínimos para el correcto funcionamiento de la aplicación. También se describen los aspectos relativos a la instalación de la herramienta y la configuración del sistema para su uso correcto, además de cuestiones relativas a su mantenimiento.

En la parte correspondiente al Presupuesto se evalúa el valor total del sistema con desglose del coste de cada una de las partes.

Por último, en la parte de Anexos se detalla cómo se realiza la construcción del árbol jerárquico visible en la aplicación, utilizando un applet JavaScript descargado de Internet. Además, se especifican todos los ficheros que han sido creados en la implementación de la aplicación y una breve descripción por cada uno de ellos.

2 Herramientas utilizadas

2.1 El protocolo HTTP

El Protocolo de Transferencia de HiperTexto[5] (Hypertext Transfer Protocol) es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los clientes Web y los servidores HTTP. La especificación completa del protocolo HTTP 1/0 está recogida en el RFC 1945. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como el World Wide Web.

Desde el punto de vista de las comunicaciones, está soportado sobre los servicios de conexión TCP/IP, y funciona de la misma forma que el resto de los servicios comunes de los entornos UNIX: un proceso servidor escucha en un puerto de comunicaciones TCP (por defecto, el 80), y espera las solicitudes de conexión de los clientes Web. Una vez que se establece la conexión, el protocolo TCP se encarga de mantener la comunicación y garantizar un intercambio de datos libre de errores.

HTTP se basa en sencillas operaciones de solicitud / respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un objeto o recurso sobre el que actúan; cada objeto Web (documento HTML, fichero multimedia o aplicación CGI) es conocido por su URL.

Los recursos u objetos que actúan como entrada o salida de un comando HTTP están clasificados por su descripción MIME. De esta forma, el protocolo puede

intercambiar cualquier tipo de dato, sin preocuparse de su contenido. La transferencia se realiza en modo binario, byte a byte, y la identificación MIME permitirá que el receptor trate adecuadamente los datos. Las principales características del protocolo HTTP son:

- Toda la comunicación entre los clientes y servidores se realiza a partir de caracteres de 8 bits. De esta forma, se puede transmitir cualquier tipo de documento: texto, binario, etc., respetando su formato original.
- Permite la transferencia de objetos multimedia. El contenido de cada objeto intercambiado está identificado por su clasificación MIME.
- Existen tres verbos básicos (hay más, pero por lo general no se utilizan) que un cliente puede utilizar para dialogar con el servidor: GET, para recoger un objeto, POST, para enviar información al servidor y HEAD, para solicitar las características de un objeto (por ejemplo, la fecha de modificación de un documento HTML).
- Cada operación HTTP implica una conexión con el servidor, que es liberada al término de la misma. Es decir, en una operación se puede recoger un único objeto.
- No mantiene estado. Cada petición de un cliente a un servidor no es influida por las transacciones anteriores. El servidor trata cada petición como una operación totalmente independiente del resto.
- Cada objeto al que se aplican los verbos del protocolo está identificado a través de la información de situación del final de la URL.

HTTP se diseñó específicamente para el World Wide Web: es un protocolo rápido y sencillo que permite la transferencia de múltiples tipos de información de forma eficiente y rápida. Se puede comparar, por ejemplo, con FTP, que es también un protocolo de transferencia de ficheros, pero tiene un conjunto muy amplio de comandos, y no se integra demasiado bien en las transferencias multimedia.

2.1.1 Etapas de una transacción HTTP

Para profundizar más en el funcionamiento de HTTP, veremos primero, un caso particular de una transacción HTTP. En los siguientes apartados se analizarán las diferentes partes de este proceso. Cada vez que un cliente realiza una petición a un servidor, se ejecutan los siguientes pasos:

1.- Un usuario accede a una URL, seleccionando un enlace de un documento HTML o introduciéndola directamente en el campo *Location* del cliente Web.

2.- El cliente Web descodifica la URL, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional (el valor por defecto es 80) y el objeto requerido del servidor.

3.- Se abre una conexión TCP/IP con el servidor, llamando al puerto TCP correspondiente.

4.- Se realiza la petición. Para ello, se envía el comando necesario (GET, POST, HEAD, …), la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada (casi siempre HTTP/1.0) y un conjunto variable de información, que incluye datos sobre las capacidades del browser, datos opcionales para el servidor, …

5.- El servidor devuelve la respuesta al cliente. Consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.

6.- Se cierra la conexión TCP.

Este proceso se repite en cada acceso al servidor HTTP. Por ejemplo, si se recoge un documento HTML en cuyo interior están insertadas cuatro imágenes, el proceso anterior se repite cinco veces, una para el documento HTML y cuatro para las imágenes.

En la actualidad se ha mejorado este procedimiento, permitiendo que una misma conexión se mantenga activa durante un cierto periodo de tiempo, de forma que sea utilizada en sucesivas transacciones. Este mecanismo, denominado *HTTP Keep Alive*, es empleado por la mayoría de los clientes y servidores modernos. Esta mejora es

imprescindible en una Internet saturada, en la que el establecimiento de cada nueva conexión es un proceso lento y costoso.

2.1.2 Estructura de los mensajes HTTP

El diálogo con los servidores HTTP se establece a través de mensajes formados por líneas de texto, cada una de las cuales contiene los diferentes comandos y opciones del protocolo. Sólo existen dos tipos de mensajes, uno para realizar peticiones y otro para devolver la correspondiente respuesta. La estructura general de los dos tipos de mensajes se puede ver en la siguiente figura:

Mensaje de solicitud		**Mensaje de respuesta**
Comando HTTP + parámetros		Resultado de la solicitud
Cabeceras del requerimiento		Cabeceras de la respuesta
(línea en blanco)		(línea en blanco)
Información opcional		Información opcional

Figura 2.1 Tipos de mensajes

La primera línea del mensaje de solicitud contiene el comando que se solicita al servidor HTTP, mientras que la respuesta contiene el resultado de la operación, un código numérico que permite conocer el éxito o fracaso de la operación. Después aparece, para ambos tipos de mensajes, un conjunto de cabeceras (unas obligatorias y otras opcionales), que condicionan y matizan el funcionamiento del protocolo.

La separación entre cada línea del mensaje se realiza con un par CR-LF (retorno de carro más nueva línea). El final de las cabeceras se indica con una línea en blanco, tras la cual se pueden incluir los datos transportados por el protocolo, por ejemplo, el documento HTML que devuelve un servidor o el contenido de un formulario que envía un cliente.

2.1.3 Comandos del protocolo

Los comandos o métodos de HTTP representan las diferentes operaciones que se pueden solicitar a un servidor HTTP. El formato general de un comando es:

Nombre del comando	Objeto sobre el que se aplica	Versión de http utilizada

Figura 2.2 Formato de un comando

Cada comando actúa sobre un objeto del servidor, normalmente un fichero o una aplicación, que se toma de la URL de activación. La última parte de esta URL, que representa la dirección de un objeto dentro de un servidor HTTP, es el parámetro sobre el que se aplica el comando. Se compone de una serie de nombres de directorios y ficheros, además de parámetros opcionales para las aplicaciones CGI.

El estándar HTTP/1.0 recoge únicamente tres comandos, que representan las operaciones de recepción y envío de información y chequeo de estado:

GET: Se utiliza para recoger cualquier tipo de información del servidor. Se utiliza siempre que se pulsa sobre un enlace o se teclea directamente a una URL. Como resultado, el servidor HTTP envía el documento correspondiente a la URL seleccionada, o bien activa un módulo CGI, que generará a su vez la información de retorno.

HEAD: Solicita información sobre un objeto (fichero): tamaño, tipo, fecha de modificación… Es utilizado por los gestores de caches de páginas o los servidores proxy, para conocer cuándo es necesario actualizar la copia que se mantiene de un fichero.

POST: Sirve para enviar información al servidor, por ejemplo los datos contenidos en un formulario. El servidor pasará esta información a un proceso encargado de su tratamiento (generalmente una aplicación CGI). La operación que se realiza con la información proporcionada depende de la URL utilizada. Se utiliza, sobre todo, en los formularios.

Un cliente Web selecciona automáticamente los comandos HTTP necesarios para recoger la información requerida por el usuario. Así, ante la activación de un enlace, siempre se ejecuta una operación GET para recoger el documento correspondiente. El envío del contenido de un formulario utiliza GET o POST, en función del atributo de <FORM METHOD="...">. Además, si el cliente Web tiene un caché de páginas recientemente visitadas, puede utilizar HEAD para comprobar la última fecha de modificación de un fichero, antes de traer una nueva copia del mismo.

La última versión de HTTP, denominada 1.1, recoge otras novedades, como los siguientes comandos:

PUT: Actualiza información sobre un objeto del servidor. Es similar a POST, pero en este caso, la información enviada al servidor debe ser almacenada en la URL que acompaña al comando. Así se puede actualizar el contenido de un documento.

DELETE: Elimina el documento especificado del servidor.

LINK: Crea una relación entre documentos.

UNLINK: Elimina una relación existente entre documentos del servidor.

2.1.4 Cabeceras HTTP

Son un conjunto de variables que se incluyen en los mensajes HTTP, para modificar su comportamiento o incluir información de interés. En función de su nombre, pueden aparecer en los requerimientos de un cliente, en las respuestas del servidor o en ambos tipos de mensajes. El formato general de una cabecera es:

Nombre de la variable	:	Cadena ASCII con su valor

Figura 2.3 Formato de una cabecera

Los nombres de variables se pueden escribir con cualquier combinación de mayúsculas y minúsculas. Además, se debe incluir un espacio en blanco entre el signo ':' y su valor. En caso de que el valor de una variable ocupe varias líneas, éstas deberán comenzar, al menos, con un espacio en blanco o un tabulador.

2.1.4.1 Cabeceras comunes para peticiones y respuestas

- ***Content-Type:*** descripción MIME de la información contenida en este mensaje. Es la referencia que utilizan las aplicaciones Web para dar el correcto tratamiento a los datos que reciben.
- ***Content-Length:*** longitud en bytes de los datos enviados, expresado en base decimal.
- ***Content-Encoding:*** formato de codificación de los datos enviados en este mensaje. Sirve, por ejemplo, para enviar datos comprimidos (x-gzip o x-compress) o encriptados.
- ***Date:*** fecha local de la operación. Las fechas deben incluir la zona horaria en que reside el sistema que genera la operación. Los formatos de fecha a emplear están recogidos en los RFC 1036 y 1123.
- ***Pragma:*** permite incluir información variada relacionada con el protocolo HTTP en el requerimiento o respuesta que se está realizando.

2.1.4.2 Cabeceras sólo para peticiones del cliente

- ***Accept:*** campo opcional que contiene una lista de tipos MIME aceptados por el cliente.
- ***Authorization:*** clave de acceso que envía un cliente para acceder a un recurso de uso protegido o limitado. La información incluye el formato de autorización empleado, seguido de la clave de acceso propiamente dicha.

- ***From:*** campo opcional que contiene la dirección de correo electrónico del usuario del cliente Web que realiza el acceso.
- ***If-Modified-Since:*** permite realizar operaciones GET condicionales, en función de si la fecha de modificación del objeto requerido es anterior o posterior a la fecha proporcionada. Puede ser utilizada por los sistemas de almacenamiento temporal de páginas.
- ***Referer:*** contiene la URL del documento desde donde se ha activado el enlace. De esta forma, un servidor puede informar al creador de ese documento de cambios o actualizaciones en los enlaces que contiene. No todos los clientes lo envían.
- ***User-agent:*** cadena que identifica el tipo y versión del cliente que realiza la petición. Por ejemplo, los browsers de Netscape envían cadenas del tipo User-Agent: Mozilla/3.0 (WinNT; I)

2.1.4.3 Cabeceras sólo para respuestas del servidor HTTP

- ***Allow:*** informa de los comandos HTTP opcionales que se pueden aplicar sobre el objeto al que se refiere esta respuesta.
- ***Expires:*** fecha de expiración del objeto enviado.
- ***Last-modified:*** fecha local de modificación del objeto devuelto.
- ***Location:*** informa sobre la dirección exacta del recurso al que se ha accedido.
- ***Server:*** cadena que identifica el tipo y versión del servidor HTTP.
- ***WWW-Autenticate:*** cuando se accede a un recurso protegido o de acceso restringido, el servidor devuelve un código de estado 401, y utiliza este campo para informar de los modelos de autentificación válidos para acceder a este recurso.

2.1.5 Códigos de estado del servidor

Ante cada transacción con un servidor HTTP, éste devuelve un código numérico que informa sobre el resultado de la operación, como primera línea del mensaje de respuesta. Estos códigos aparecen en algunos casos en la pantalla del cliente, cuando se produce un error (en ocasiones se presenta un mensaje de error más elaborado, en forma de documento HTML). El formato de la línea de estado es:

Versión del protocolo http utilizado	Código numérico de estado (tres dígitos)	Descripción del código numérico

Figura 2.4 Formato de la línea de estado

Existen cinco categorías de mensajes de estado, organizadas por el primer dígito del código numérico de la respuesta:

- ***1xx:*** mensajes informativos. Por ahora (en HTTP/1.0) no se utilizan, y están reservados para un futuro uso.
- ***2xx:*** mensajes asociados con operaciones realizadas correctamente.
- ***3xx:*** mensajes de redirección, que informan de operaciones complementarias que se deben realizar para finalizar la operación.
- ***4xx:*** errores del cliente; el requerimiento contiene algún error, o no puede ser realizado.
- ***5xx:*** errores del servidor, que no ha podido llevar a cabo una solicitud.

2.2 El servidor Apache

Apache[6] es uno de los servidores web más utilizados en todo el mundo. Inicialmente NCSA (National Center for Supercomputing Applications) creó un servidor web que se convirtió en el número uno en poco tiempo.

Apache era, inicialmente, unos parches al servidor de WWW de NCSA conocido como httpd. Al igual que GNU/Linux, fue un proyecto que atrajo a mucha gente por el gran interés de su objetivo: lograr el servidor web más rápido, más eficiente y con mayor funcionalidad desde el enfoque del software libre. Y ha sido un objetivo que se ha cumplido. Sólo hacían sobras en Apache ciertos aspectos de rendimiento, fundamentalmente por no utilizar hebras de ejecución. Este aspecto se soluciona en Apache 2.0, el paso siguiente a Apache 1.0.

Con el tiempo los usuarios del servidor httpd comenzaron a intercambiar parches y mejoras y crearon un foro para la administración de dichos parches. Había nacido Apache Group. Se utilizó el código del servidor del NCSA y se creó un nuevo servidor llamado Apache, uniendo el código fuente con la multitud de parches que había existentes. Apache ha logrado convertirse en el servidor web por excelencia del mercado.

Apache es utilizado en prácticamente todas las plataformas (UNIX, Linux, Windows). Apache tiene muchas características, como la indexación de directorios, uso de sobrenombres con las carpetas, negación de contenidos, informes configurables sobre los errores http, ejecuciones SetUID y SetGID o programas CGI.

2.2.1 Características de Apache

Apache integra multitud de características que permiten personalizar el servidor y que dan acceso a sus funcionalidades más importante. Entre las principales características de Apache destacan:

- Admite la última versión del protocolo HTTP/1.1. Es completamente compatible con esta versión y con la anterior HTTP/1.0. Por ejemplo, antes de HTTP/1.1, el navegador tenía que esperar la respuesta del servidor antes de poder realizar otra petición. Con esta nueva versión, el navegador puede enviar peticiones en paralelo, con lo que se ahorra ancho de banda al no tener que incluir una cabecera de contenido en todas y cada una de las solicitudes.

- Apache es bastante simplista en su configuración. Posee tres archivos de texto plano en donde residen todas las directivas que conforman la configuración del servidor. Su configuración es sólo cuestión de usar un simple editor de texto. Además, estos tres ficheros se pueden unir en uno sólo con el propósito de agrupar todas las directivas juntas.

- Apache puede trabajar con CGI. El servidor utiliza el módulo mod_cgi en su versión 1.1. Entre las características que incluye este módulo se encuentra la personalización de las variables de entorno o la búsqueda de errores.

- Apache admite la autentificación http. Puede trabajar con la autentificación básica de la red. Está preparado para asimilar los mensajes basados en la autentificación. Para sus necesidades, Apache puede usar archivos de contraseñas tipo texto plano, DBM, SQL o incluso llamadas a programas externos especializados en la materia.

- Apache posee registros personalizables e información sobre el estado en todo momento del servidor. El servidor es muy flexible a la hora de registrar y visualizar el estado interno del servidor. De hecho, se puede acceder a esta información a través de un navegador web.

- Apache puede trabajar con SSI (Server Side Includes). A través de esta funcionalidad, el servidor puede ejecutar programas externos y CGIs simultáneamente a la carga de las páginas web y proporcionar también acceso directo a las variables CGI de entorno durante el acceso al recurso por parte del cliente. Dispone así de una serie de anexos que mejoran la flexibilidad del desarrollador de sitios web y el programador de CGIs.

2.3 Sistema operativo Linux

Linux[7] es un sistema operativo, compatible Unix. Dos características muy peculiares lo diferencian del resto de los sistemas que podemos encontrar en el mercado, la primera, es que es libre, esto significa que no tenemos que pagar ningún tipo de licencia a ninguna casa desarrolladora de software por el uso del mismo, la segunda, es que el sistema viene acompañado del código fuente. El sistema lo forman el núcleo del sistema

(kernel) más un gran número de programas / librerías que hacen posible su utilización. LINUX se distribuye bajo la GNU Public License: Inglés, por lo tanto, el código fuente tiene que estar siempre accesible.

El sistema ha sido diseñado y programado por multitud de programadores alrededor del mundo. El núcleo del sistema sigue en continuo desarrollo bajo la coordinación de Linus Torvalds, la persona de la que partió la idea de este proyecto, a principios de la década de los noventa. Día a día, más y más programas / aplicaciones están disponibles para este sistema, y la calidad de los mismos aumenta de versión a versión. La gran mayoría de los mismos vienen acompañados del código fuente y se distribuyen gratuitamente bajo los términos de licencia de la GNU Public License.

2.3.1 Qué es Linux

Linux[1] es un sistema operativo diseñado por cientos de programadores de todo el planeta. Su objetivo inicial es propulsar el software de libre distribución junto con su código fuente para que pueda ser modificado por cualquier persona, dando rienda suelta a la creatividad. El hecho de que el sistema operativo incluya su propio código fuente expande enormemente las posibilidades de este sistema. Este método también es aplicado en numerosas ocasiones a los programas que corren en el sistema, lo que hace que podamos encontrar muchísimos programas útiles totalmente gratuitos y con su código fuente.

Las funciones principales de este magnífico sistema operativo son:

- ***Sistema multitarea.*** En Linux es posible ejecutar varios programas a la vez sin necesidad de tener que parar la ejecución de cada aplicación.
- ***Sistema multiusuario.*** Varios usuarios pueden acceder a las aplicaciones y recursos del sistema Linux al mismo tiempo. Y, por supuesto, cada uno de ellos puede ejecutar varios programas a la vez (multitarea).
- ***Shells programables.*** Un shell conecta las órdenes de un usuario con el Kernel de Linux (el núcleo del sistema), y al ser programables se puede modificar para

adaptarlo a sus necesidades. Por ejemplo, es muy útil para realizar procesos en segundo plano.

- ***Independencia de dispositivos.*** Linux admite cualquier tipo de dispositivo (módems, impresoras) gracias a que cada una vez instalado uno nuevo, se añade al Kernel el enlace o controlador necesario con el dispositivo, haciendo que el Kernel y el enlace se fusionen. Linux posee una gran adaptabilidad y no se encuentra limitado como otros sistemas operativos.
- ***Comunicaciones.*** Linux es el sistema más flexible para poder conectarse a cualquier ordenador del mundo. Internet se creó y desarrolló dentro del mundo de Unix, y por lo tanto Linux tiene las mayores capacidades para navegar, ya que Unix y Linux son sistemas prácticamente idénticos. Con Linux podrá montar un servidor en su propia casa sin tener que pagar las enormes cantidades de dinero que piden otros sistemas.

Linux no sacrifica en ningún momento la creatividad, tal y como lo hacen algunas compañías informáticas. Linux es una ventana abierta por la que es posible huir hacia un mundo donde la verdadera informática puede ser disfrutada sin límites ni monopolios.

2.3.2 Características de Linux

- Multitarea: varios programas (realmente procesos) ejecutándose al mismo tiempo.
- Multiusuario: varios usuarios en la misma máquina al mismo tiempo (y sin licencias para todos).
- Multiplataforma: corre en muchas CPUs distintas, no sólo Intel.
- Funciona en modo protegido 386.
- Tiene protección de la memoria entre procesos, de manera que uno de ellos no pueda colgar el sistema.
- Carga de ejecutables por demanda: Linux sólo lee de disco aquellas partes de un programa que están siendo usadas actualmente.

- Política de copia en escritura para la compartición de páginas entre ejecutables: esto significa que varios procesos pueden usar la misma zona de memoria para ejecutarse. Cuando alguno intenta escribir en esa memoria, la página (4Kb de memoria) se copia a otro lugar. Esta política de copia en escritura tiene dos beneficios: aumenta la velocidad y reduce el uso de memoria.

- Memoria virtual usando paginación (sin intercambio de procesos completos) a disco: una partición o un archivo en el sistema de archivos, o ambos, con la posibilidad de añadir más áreas de intercambio sobre la marcha (se sigue denominando intercambio, es en realidad un intercambio de páginas). Un total de 16 zonas de intercambio de 128Mb de tamaño máximo pueden ser usadas en un momento dado con un límite teórico de 2Gb para intercambio.

- La memoria se gestiona como un recurso unificado para los programas de usuario y para la caché de disco, de tal forma que toda la memoria libre puede ser usada para caché y éste puede a su vez ser reducido cuando se ejecuten grandes programas.

- Librerías compartidas de carga dinámica (DLL's) y librerías estáticas también, por supuesto.

- Se realizan volcados de estado (core dumps) para posibilitar los análisis post-mortem, permitiendo el uso de depuradores sobre los programas, no sólo en ejecución, sino también tras abortar éstos por cualquier motivo.

- Casi totalmente compatible con POSIX, System V y BSD a nivel fuente.

- Mediante un módulo de emulación de iBCS2, casi completamente compatible con SCO, SVR3 y SVR4 a nivel binario.

- Todo el código fuente está disponible, incluyendo el núcleo completo y todos los drivers, las herramientas de desarrollo y todos los programas de usuario; además todo ello se puede distribuir libremente. Hay algunos programas comerciales que están siendo ofrecidos para Linux actualmente sin código fuente, pero todo lo que ha sido gratuito sigue siendo gratuito.

- Control de tareas POSIX.

- Pseudo-terminales (pty's).

- Emulación de 387 en el núcleo, de tal forma que los programas no tengan que hacer su propia emulación matemática. Cualquier máquina que ejecute Linux parecerá dotada de coprocesador matemático. Por supuesto, si tu ordenador ya tiene una FPU (unidad de coma flotante), será usada en lugar de la emulación, pudiendo incluso compilar tu propio kernel sin la emulación matemática y conseguir un pequeño ahorro de memoria.
- Soporte para muchos teclados nacionales o adaptados y es bastante fácil añadir nuevos dinámicamente.
- Consolas virtuales múltiples: varias sesiones de login a través de la consola entre las que se puede cambiar con las combinaciones adecuadas de teclas (totalmente independiente del hardware de video). Se crean dinámicamente y puedes tener hasta 64.
- Soporte para varios sistemas de archivo comunes, incluyendo minix-1, Xenix y todos los sistemas de archivo típicos de System V, y tiene un avanzado sistema de archivos propio con una capacidad de hasta 4 Tb y nombres de archivos de hasta 255 caracteres de longitud.
- Acceso transparente a particiones MS-DOS (o a particiones OS/2 FAT) mediante un sistema de archivos especial: no se necesita ningún comando especial para usar la partición MS-DOS, parece un sistema de archivos normal de Unix (excepto por algunas graciosas restricciones en los nombres de archivo, permisos, y esas cosas). Las particiones comprimidas de MS-DOS 6 no son accesibles en este momento, y no se espera que lo sean en el futuro. El soporte para VFAT (WNT, Windows 95) ha sido añadido al núcleo de desarrollo y estará en la próxima versión estable.
- Un sistema de archivos especial llamado UMSDOS que permite que Linux sea instalado en un sistema de archivos DOS.
- Soporte en sólo lectura de HPFS-2 del OS/2 2.1
- Sistema de archivos de CD-ROM que lee todos los formatos estándar de CD-ROM.
- TCP/IP, incluyendo ftp, telnet, NFS, etc.

2.3.3 Ejecutando comandos de Linux desde C

2.3.3.1 Función system

Se pueden ejecutar comandos desde un programa de C como si se estuviera en la línea de comandos de Linux usando la función *system()*[2].

```
int system (char *mandato)
```

Donde *mandato* puede ser el nombre de una instrucción Linux, un shell ejecutable o un programa del usuario. La función regresa el estado de salida del shell. La función tiene su prototipo en <stdlib.h>.

La función *system* es una llamada que esta construida de otras 3 llamadas del sistema: *execl()*, *wait()* y *fork()* (las cuales tienen su prototipo en <*unistd.h*>).

2.3.3.2 Funciones exec

El sistema operativo Linux ofrece una llamada al sistema llamada *exec*[2] para lanzar a ejecución un programa, almacenado en forma de fichero. Aunque en el fondo sólo existe una llamada, las bibliotecas estándares del C disponen de varias funciones, todas comenzando por *exec* que se diferencian en la manera en que se pasan parámetros al programa.

La declaración de la familia de funciones *exec* es la siguiente:

```
int execl (path, arg0, arg1, ..., argn, (char *0))
char *path, *arg0, *arg1, ..., *argn;

int execv (path, argv)
char *path, *argv[];

int execle (path, arg0, arg1, ..., argn, (char *)0, envp)
char *path, *arg0, *arg1, ..., *argn, *envp[];
```

```
int execve (path, argv, envp)
char *path, *argv[], *envp[];

int execlp (file, arg0, arg1, ..., argn, (char *)0)
char *file, *arg0, *arg1, ..., *argn;

int execvp (file, argv)
char *file, *argv[];
```

En todas estas funciones, *path* apunta a la ruta (absoluta o relativa) de un fichero ordinario ejecutable.

File apunta al nombre de un fichero ejecutable. La ruta del fichero se construye buscando el fichero en los directorios que se indican en la variable de entorno *path*. Tanto *path* como *file* se refieren a ficheros ejecutables o a ficheros de datos (*shell script*) para un intérprete de órdenes. Si el fichero no tiene un número mágico que lo identifique como directamente ejecutable, se le pasa a */bin/sh* como un fichero de órdenes para el intérprete.

Arg0, arg1, ..., argn son punteros a cadenas de caracteres y constituyen la lista de argumentos que se le pasa al nuevo programa. Por convenio, al menos *arg0* está presente siempre y apunta a una cadena idéntica a *path* o al último componente de *path*. Hay que resaltar que a continuación de *argn* pasamos un puntero *NULL ((char *)0)* para indicar el final de los argumentos.

Argv es un array de cadenas de caracteres que constituyen la lista de argumentos que va a recibir el nuevo programa. Por convenio, *argv* debe tener al menos un elemento, que debe apuntar a una cadena idéntica a *path* o al último componente de *path*. El final de *argv* se indica haciendo que a continuación de su último elemento significativo haya un puntero *NULL*.

Envp es un array de punteros a cadenas de caracteres que constituyen el entorno en el que se va a ejecutar el nuevo programa. *Envp* termina también con un puntero *NULL*.

Si el nuevo programa que pasa a ejecutarse está escrito en C, entonces recibe los parámetros *arg0, arg1, ..., argn* o *argv* y *envp* a través de la función principal *main*.

2.4 HTML

HTML[8] es un lenguaje sencillo pensado para presentar información en la WWW. HTML (HyperText Markup Languaje), como su nombre indica es un lenguaje de marcas para la creación de hipertextos. Por hipertexto entenderemos texto con una presentación agradable, con inclusión de elementos multimedia (gráficos, video, audio) y con la presencia de hiperenlaces que permiten relacionar otras fuentes de información en documentos hipertextos. Es el lenguaje utilizado para representar documentos en la WWW (*World Wide Web*).

Como se ha dicho, es un lenguaje de marcas ya que en él, las instrucciones son trozos de texto resaltados convenientemente que definirán la estructura lógica del documento. Por tanto, un documento HTML constará de texto que será el contenido y la información del documento y de instrucciones HTML que resaltarán este contenido y le darán un formato fácil y agradable de leer y con la posibilidad de relacionar documentos y fuentes de información mediante hiperenlaces. Las marcas del lenguaje HTML especifican:

- La estructura lógica del documento:
    - Cabeceras, tipos y tamaños de las fuentes.
    - Párrafos de texto.
    - Centrado.
    - Enumeraciones o listas.
    - Formularios.
    - Tablas.
- Distintos estilos que definen el texto:
    - Negrita.
    - Cursiva.
    - Diferentes efectos: (direcciones de correo, citas textuales, etc…).
- Inclusión de hipertextos para acceder a otros documentos relacionados
- Inclusión de imágenes y ficheros multimedia.

El lenguaje HTML actualmente está en su versión 2.0, aunque se encuentra en desarrollo la 3.0. En este estándar se definen las etiquetas básicas y las estructuras de datos que forman el lenguaje. En este manual explicaremos todas las estructuras definidas en la especificación 2.0 añadiendo las extensiones incluidas por NetScape en su versión 2.0. La unión de ambos constituye la especificación del lenguaje HTML más extendida en la actualidad.

2.4.1 Etiquetas del lenguaje HTML

El lenguaje HTML es un lenguaje de marcas, estas marcas serán fragmentos de texto destacado de una forma especial que permiten la definición de las distintas instrucciones de HTML, tanto los efectos a aplicar sobre el texto como las distintas estructuras del lenguaje. A estas marcas las denominaremos etiquetas y serán la base principal del lenguaje HTML. En documento HTML será un fichero texto con etiquetas que variarán la forma de su presentación.

Una etiqueta será un texto incluido entre los símbolos *menor que* < y *mayor que* >**.** El texto incluido dentro de los símbolos será explicativo de la utilidad de la etiqueta.

Existe normalmente una etiqueta de inicio y otra de fin, la de fin contendrá el mismo texto que la de inicio añadiéndole al principio una barra inclinada /. El efecto que define la etiqueta tendrá validez para todo lo que este incluido entre las etiquetas de inicio y fin, ya sea texto plano u otras etiquetas HTML.

```
<ETIQUETA> Elementos afectados por la etiqueta </ETIQUETA>
```

Algunas etiquetas no necesitarán la de fin, serán aquellas en las que el final este implícito, por ejemplo <*P*> párrafo, <*BR*> salto de línea ó <*IMG*> inclusión de una imagen. Definen un efecto que se producirá en un punto determinado sin afectar a otros elementos.

El uso de mayúsculas o minúsculas en las etiquetas es indiferente, se interpretarán del mismo modo en ambos casos, pero lo normal es expresarlas en mayúsculas para que destaquen con más nitidez del texto normal.

2.4.1.1 Atributos de las etiquetas

Las etiquetas pueden presentar modificadores que llamaremos atributos que permitirán definir diferentes posibilidades de la instrucción HTML. Estos atributos se definirán en la etiqueta de inicio y consistirán normalmente en el nombre del atributo y el valor que toma separados por un signo de igual. El orden en que se incluyan los atributos es indiferente, no afectando al resultado. Si se incluyen varias veces el mismo atributo con distintos valores, el resultado obtenido será imprevisible dependiendo de como lo interprete el navegador. Cuando el valor que toma el atributo tiene más de una palabra deberá expresarse entre comillas, en otro caso no será necesario.

Un ejemplo de atributo será:

```
<A HREF="http://www.ulpgc.es">Pagina de la ULPGC</A>
```

Igualmente una etiqueta podría presentar varios atributos:

```
<HR ALIGN=LEFT NOSHADE SIZE=5 WIDTH=50%>
```

En este caso la etiqueta <*HR*> presenta cuatro atributos. El segundo atributo *NOSHADE* es un caso especial que no presenta valor. El orden en que se especifiquen los atributos no afectará al resultado final.

2.4.1.2 Etiquetas correctas

Todo texto que se encuentre entre los caracteres < y > se considerará una etiqueta, si la etiqueta no fuera una de las válidas del lenguaje HTML no se tendrá en cuenta, sin causar ningún tipo de error. Dejándose el texto o las etiquetas a las que afectaba como si no

existiera la etiqueta extraña. Cuando se comete un error sintáctico al expresar una etiqueta o un atributo no se producirá ningún error, simplemente no se obtendrá el efecto que deseábamos.

El lenguaje HTML es un lenguaje que evoluciona muy rápidamente y cada nueva versión de los programas navegadores presenta etiquetas nuevas que causan efectos más espectaculares o atributos nuevos de las etiquetas ya existentes. Esto causa que los programas más antiguos no entiendan estas nuevas etiquetas y por tanto las considere erróneas y no realice la acción que deseábamos. Dándose el caso de atributos que son válidos sólo para un único navegador.

Cuando creemos código HTML hay que hacerlo lo más estándar posible para permitir que el documento pueda ser visto de forma efectiva por distintos navegadores en máquinas distintas. Por tanto, debemos renunciar a efectos espectaculares que sólo tienen validez en un navegador e intentar comprobar cómo se ve el documento en una variedad de navegadores, ya que las personas que se conectan a nuestras páginas no tendrán, en la mayoría de los casos, el mismo que nosotros. También es interesante como se vería el documento en los distintos tamaños de la ventana del navegador, teniendo en cuenta que todos no tienen un monitor con la misma resolución.

2.4.2 Estructura de un documento HTML

2.4.2.1 Estructura básica

Un documento HTML está definido por una etiqueta de apertura <*HTML*> y una etiqueta de cierre <*/HTML*>. Dentro de éste, se dividen dos partes fundamentales, la cabecera, delimitada por la etiqueta <*HEAD*> y el cuerpo, delimitado por la etiqueta <*BODY*>. Por tanto la estructura de un documento HTML será:

```
<HTML>
<HEAD>
   Definiciones de la cabecera
</HEAD>
```

```
<BODY>
   Instrucciones HTML
</BODY>
</HTML>
```

Ninguno de estos elementos es obligatorio, pudiendo componer documentos HTML que se muestren sin ningún problema sin incluir estas etiquetas de identificación. Si se utilizan elementos que forzosamente deban ser incluidos en la cabecera (como la etiqueta de titulo), no serán reconocidos correctamente si no se incluyen entre las etiquetas de *<HEAD>*.

2.4.2.2 Comentarios

Para insertar comentarios dentro de un documento HTML utilizaremos la etiqueta especial <!--, definiéndose un comentario de la forma:

```
<!-- Esto es un comentario -->
```

Los comentarios son útiles para identificar el documento, pudiendo indicar al comienzo del documento su título, autor y la fecha en el que fue realizado, esto se hace antes de la etiqueta *<HTML>*. También, aunque ya con menos utilidad, para comentar cualquier instrucción o circunstancia del documento.

Los comentarios no se muestran en el documento HTML y el único modo de verlo es viendo el código HTML fuente del documento, cosa que permiten algunos navegadores de la WWW.

2.4.3 Cabecera de un documento HTML

La cabecera de un documento HTML está delimitada por las etiquetas *<HEAD>* y *</HEAD>*, en ésta se incluirán las definiciones generales que afectarán a todo el documento. Todas sus etiquetas son opcionales y se utilizarán sólo en casos muy

determinados, sólo la etiqueta <*TITLE*> tiene un uso general y aunque es opcional se recomienda incluirla en todos los documentos que creemos.

A continuación se citan los distintos componentes que pueden formar la cabecera de un documento HTML.

2.4.3.1 <TITLE>: Título del documento

Especificará el título del documento HTML, todo documento HTML debe tener un título. El título debe guardar relación con el contenido del documento y definirlo de forma general, su tamaño no está limitado aunque se recomienda que no sea excesivamente extenso. Dentro de esta etiqueta no se podrán usar ninguna de las restantes etiquetas HTML.

El título no forma parte del documento en sí, y no se incluye ni se muestra dentro del documento, normalmente se muestra en la parte superior de la ventana del programa navegador.

Se utiliza principalmente para etiquetar e identificar la página en los *bookmarks* (marcadores) y las *history list* (lista de documentos accedidos) y también se utiliza por algunos servidores de búsqueda como resultado de una búsqueda para poder intuir el contenido del documento. El título deberá guardar relación con el contenido del documento y definirlo de forma general.

La etiqueta <*TITLE*>, debe ser usada dentro de las etiquetas que definen la cabecera de la siguiente forma:

```
<HEAD>
<TITLE> Título del documento HTML </TITLE>
</HEAD>
```

2.4.3.2 <BASE>: URL base del documento.

Especificará la URL que se tomará como base del documento HTML, ésta se utilizará para las referencias a URL relativas, que se encuentren en los hiperenlaces y en las referencias de las imágenes. No tiene mucho uso y lo normal es utilizarlo en documentos, obtenidos de otros servidores o directorios e incluidos fuera de contexto, de esta forma los enlaces siguen siendo válidos.

Presenta un atributo HREF, que indicará la URL base del documento, el formato será el siguiente:

```
<BASE HREF="URL">
```

2.4.3.3 <ISINDEX>: El documento es un índice

Indica que el documento es un índice, y por tanto se deberá realizar una búsqueda en él. Presentará un indicador preguntando la palabra clave de la búsqueda.

No se utiliza en documentos normales, sino en documentos realizados con cgi-bin para indicar que les debe pasar una palabra clave para realizar la búsqueda. Presenta dos atributos, PROMPT que indica el texto que aparecerá como inductor de la búsqueda y ACTION que especifica el fichero cgi que trata la consulta, por defecto será el documento actual.

2.4.3.4 <META>: Indica refresco del documento

Aunque tiene más utilidades, la principal es indicar documentos con refresco automático. Se indicará un documento que debe sustituir al actual en un determinado número de segundos.

El formato es el siguiente:

```
<META HTTPEQUIV="REFRESH"
CONTENT="número_segundos;URL=URL_de_refresco">
```

Se indica el número de segundos que deben pasar antes del refresco y el documento HTML que sustituye al actual. Si se indica cero segundos, la transición entre uno y otro documento será inmediata. Si no se indica URL el documento actual se refrescará.

2.4.4 Cuerpo de un documento HTML

El cuerpo de un documento HTML estará delimitado por las etiquetas <*BODY*> y <*/BODY*> y en él se incluirán todas las instrucciones HTML y el texto que forman el documento, es similar al BEGIN ó { de un lenguaje de programación. Al igual que la cabecera <*HEAD*> es opcional, pero se recomienda para la buena identificación de las distintas zonas del documento. Si un documento no presenta ninguna de las etiquetas de identificación de sus distintas partes (<*HTML*>, <*HEAD*> ó <*BODY*>) se considerará que todo lo que se defina pertenece al cuerpo del documento.

2.4.4.1 Atributos de <BODY>

La etiqueta <*BODY*> presenta algunos atributos que son de definición global para todo el documento, éstos definirán los colores y el fondo del documento HTML.

Los atributos de <*BODY*> son:

```
<BODY BACKGROUND="URL" BGCOLOR=#rrvvaa TEXT=#rrvvaa
LINK=#rrvvaa VLINK=#rrvvaa >
```

BACKGROUND: Definirá la imagen que se utilizará de fondo del documento HTML, la URL definida será el camino a una imagen. Ésta se muestra debajo del texto y las imágenes del documento HTML. En el caso de que la imagen no rellene todo el fondo del documento, ésta será reproducida tantas veces como sea necesario.

BGCOLOR*:* Indicará el color del fondo del documento HTML, sólo se utilizará si no se ha definido una imagen de fondo, o si esta imagen no puede obtenerse.

TEXT*:* Especificará el color del texto normal dentro del documento HTML. Por defecto será normalmente negro.

LINK*:* Indicará el color que tendrán los hiperenlaces que no han sido accedidos. Por defecto será azul.

VLINK*:* Color de los enlaces que ya han sido visitados. Por defecto es un color azul más oscuro.

2.5 JavaScript

JavaScript[9] es un lenguaje compacto, y basado en objetos, diseñado para el desarrollo de aplicaciones cliente-servidor a través de Internet. Netscape Navigator 2.0 es capaz de interpretar sentencias JavaScript embebidas en programas CGI.

En una aplicación cliente para un browser, las sentencias JavaScript embebidas en un documento HTML pueden reconocer y responder a eventos generados por el usuario, como clics del ratón, información en formularios y navegación de documento a documento.

Por ejemplo, se puede escribir una función JavaScript que verifique que la información ingresada por el usuario sea correcta. Sin que haya transmisión de main por la red, un documento HTML con JavaScript embebido es capaz de interpretar la información ingresada por el usuario, verificar que sea correcta y alertar al usuario en caso que no lo sea. Se puede también ejecutar archivos de audio, applet's o comunicar con una extensión de Netscape (plug-in's) en respuesta a la apertura o cierre de una página.

2.5.1 Scripts en el browser

JavaScript puede ser embebido en un documento HTML de dos maneras:

- Como funciones y sentencias usando el elemento < SCRIPT >.
- Como manejadores de eventos usando atributos manejadores en HTML.

2.5.1.1 El elemento Script

Una script embebido en HTML, usando este elemento, utiliza el formato:

```
<SCRIPT LANGUAGE="JavaScript">
   sentencias;
</SCRIPT>
```

2.5.1.2 Definiendo y llamando funciones

Los scripts que se ubican entre los elementos <*SCRIPT*> y <*/SCRIPT*> se ejecutan inmediatamente después de cargar la página, en cambio las funciones no se ejecutan hasta que sean llamadas.

Es importante entender la diferencia entre definir y llamar una función: Definirla sólo especifica un nombre para ella y qué hace cuando es llamada; Llamarla implica ejecutarla utilizando los parámetros que le son pasados por la sentencia JavaScript que la llama.

Generalmente se deben definir las funciones para una página en el encabezado del documento, esto es, dentro del elemento <*HEAD*>. Debido a que el encabezado es cargado primero, esta práctica asegura que las funciones sean cargadas antes que el usuario tenga oportunidad de hacer algo que provoque una llamada a la función.

2.6 Hojas de estilos. CSS

Una Hoja de Estilo en Cascada[10] (Cascading Style Sheet, CSS) es una hoja de estilo para página web, derivada de múltiples fuentes, con un orden definido de prioridad cuando las definiciones de cualquier elemento de estilo entran en conflicto. La recomendación de Hoja de Estilo en Cascada, nivel 1 (CSS1) del World Wide Web Consortium, que está implementada en las más recientes versiones de los navegadores de la Web de Netscape y Microsoft, especifica las posibles hojas de estilo o expresiones que pueden determinar cómo un elemento dado se presenta en una página web.

Las CSS proporcionan mayor control sobre la apariencia de una página web al creador de la misma que al diseñador del navegador o al usuario. Con las CSS, las fuentes de definición de estilo para un elemento dado de un documento están en este orden de prioridad:

1.- El atributo de estilo STYLE en la etiqueta de un elemento individual.

2.- El elemento STYLE que define una hoja específica que contenga declaraciones de estilo o un elemento de enlace LINK que enlaza con un documento separado que contiene el elemento STYLE. En una página web, el elemento STYLE se coloca entre las etiquetas de TITLE y BODY.

3.- Una hoja de estilo importada que utilice la notación CSS@import para importar y conjuntar automáticamente una hoja externa con su actual hoja de estilo.

4.- Atributos de estilo especificados por el usuario a su navegador.

5.- La hoja de estilo por omisión que asume el navegador.

En general, la hoja de estilo del creador de la página web tiene prioridad, pero se recomienda que los navegadores proporcionen formas mediante las cuales, el usuario pueda controlar los atributos de estilo en algunos aspectos. Dado que lo más probable es que los distintos navegadores decidirán implementar la CSS1 de maneras un tanto distintas, el creador de la página web debe probar la página con distintos navegadores.

2.7 PHP

PHP[4] corresponde a las iniciales de Personal Home Page, Procesador de Hipertexto. PHP es un lenguaje de programación, con una sintaxis similar a los lenguajes C y Perl, que se interpreta por un servidor web Apache y genera código HTML dinámico. Es decir, nos permite crear un programa que se puede ejecutar en el servidor desde un programa visualizador de páginas web y dar respuestas en función de los datos que introduzca el usuario.

El cliente nunca verá el código del programa PHP, sólo le llegarán las páginas HTML que genere el programa. A diferencia de JavaScript, que se ejecuta en las máquinas clientes, un programa PHP se ejecuta en el servidor web.

En la figura 2.5 se puede apreciar el esquema de funcionamiento de un programa PHP: el cliente realiza una petición de un programa a un servidor web como si se tratara de cualquier otra página; el cliente no sabrá distinguirlo. El servidor web, por la extensión de la página (.php por ejemplo), se la envía al intérprete PHP y éste, una vez ejecutado el programa, le devuelve los resultados al navegador cliente.

Figura 2.5 Esquema de funcionamiento de PHP

Para poder usar PHP necesitamos unos mínimos conocimientos de HTML. Aunque siempre existe la posibilidad de generar mediante un programa la página HTML y después, mediante un editor de textos añadir el código PHP.

El lenguaje de programación PHP dispone de funciones para realizar las operaciones habituales de los lenguajes de programación, usar ficheros, tratamiento de cadenas de texto, etc. Quizás una de las características más interesantes que incorpora, es la facilidad para consultar bases de datos y generar páginas en función de los resultados obtenidos en la correspondiente consulta. Las consultas se pueden realizar, bien en modo nativo o bien, mediante ODBC. En modo nativo, es compatible con Oracle, Postgres, Informix, MySQL y muchas otras.

Además, en este lenguaje podemos definir clases y usar ciertas características de la programación orientada a objeto.

2.7.1 Qué ventajas tiene PHP

PHP presenta múltiples ventajas frente a otros lenguajes de programación que necesariamente harán que este lenguaje se imponga como una alternativa para el desarrollo de todo tipo de aplicaciones.

2.7.1.1 Interfaz

En primer lugar se ejecuta a través de una interfaz que le resulta familiar al usuario: el cliente web. No es necesario que el usuario aprenda nuevas combinaciones de teclas, ni nada parecido, para aprender a usar el programa.

Tampoco es necesario tener que instalar ningún software adicional en la estación cliente para usar un programa PHP, a parte del propio navegador web.

De la misma forma, la ejecución de un programa PHP se puede realizar desde un cliente web de cualquier plataforma: el usuario puede escoger su sistema operativo y su cliente web preferidos.

2.7.1.2 Acceso en red

El propio diseño de PHP lleva incorporada esta virtud. El programa se ejecuta en un servidor al cual, se puede acceder desde cualquier puesto de una red. EL servidor siempre podría limitar el acceso a sólo determinados puestos y además obligar a la autenticación de un usuario para poder acceder a ciertas partes de un programa.

2.7.1.3 Protección del código

Al tener el código ejecutable albergado en el servidor, este código está protegido, tanto de la manipulación de los usuarios, como de la presencia de virus.

2.7.2 Por qué PHP

Hay múltiples razones para escoger PHP como lenguaje de programación para entornos web en comparación con otros existentes en el mercado. Algunas de estas razones son:

- Es muy fácil de aprender. Conociendo los lenguajes C, Perl, Java o programación en Shell de Unix, puede decirse que se conoce los fundamentos de PHP.
- Se puede hacer cualquier cosa con PHP.
- Está ampliamente probado como herramienta. Más de millón y medio de servidores en todo el mundo lo avalan como una plataforma para desarrollar aplicaciones de portales, comercio electrónico, aplicaciones en intranets, etc.
- Porque se puede usar prácticamente en cualquier plataforma.
- Porque, aunque no dispone de soporte comercial, existen numerosas listas de correo en las que se pueden obtener soluciones a los problemas que se pueden presentar.

- Porque existen recursos en la web que pueden facilitar el desarrollo de nuestras aplicaciones. Existen bibliotecas de clases que resuelven los problemas más frecuentes con los que se puede encontrar el programador.
- Porque no requiere unos recursos desmesurados para funcionar.

2.7.3 Qué se necesita

2.7.3.1 Servidor Apache

Para poder usar nuestros programas PHP, evidentemente, lo primero que necesitamos es un servidor web Apache. Además, este servidor web Apache tiene que estar compilado para incorporar las características PHP o admitir módulos dinámicos. La documentación de PHP, y también la de Apache, explican cómo podemos compilar PHP y adaptarlos a nuestras necesidades.

PHP admite una gran variedad de módulos, pero las versiones precompiladas no los incorporan todos. Si necesita alguna funcionalidad concreta, es muy probable que tenga que compilar su propia versión de PHP.

2.7.3.2 Plataforma para PHP

PHP funciona asociado al servidor Apache y se distribuyen los programas fuentes en C. Así es fácil compilarlo en una u otra plataforma que disponga de un compilador de C, realizando las modificaciones necesarias.

2.7.3.3 Sistema gestor de bases de datos

La facilidad con que PHP es capaz de consultar una base de datos le confiere una potencia excepcional a la hora de generar páginas web dinámicas. En la figura 2.6 podemos ver la relación que se puede establecer entre PHP y un sistema gestor de bases de

datos: nuestro programa puede realizar consultas y obtener los resultados para procesarlos. Este gestor de base de datos puede estar ubicado en la misma máquina que ejecuta PHP o puede estar en otra máquina distinta y realizar las consultas a través de la red.

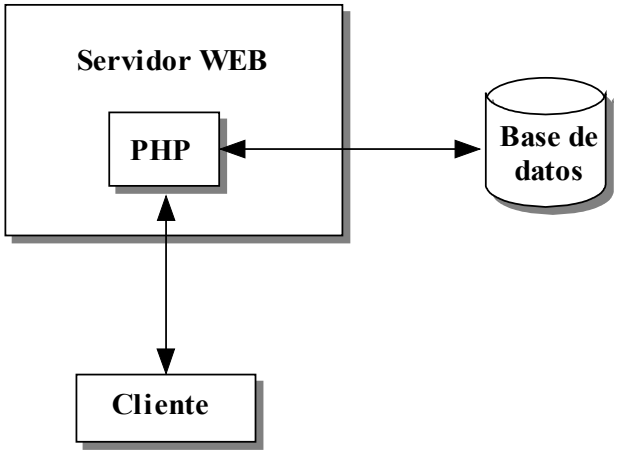


Figura 2.6 PHP y Base de Datos

2.7.4 Cómo es el lenguaje PHP

El objetivo de PHP es generar código HTML, por lo que su relación dentro de un programa, es muy estrecha. Tanto es así, que el lenguaje PHP tiene que convivir en el mismo fichero con el lenguaje HTML y obliga a establecer unos criterios para tener los códigos respectivos convenientemente separados.

Como sabemos, los controles HTML son de la forma <control> y </control>, pues de una forma similar se establecen unos controles para distinguir qué partes del fichero son interpretadas por PHP y cuáles se envían al programa navegador cliente.

- Una primera forma es: `<? código php ?>`
- Otra forma posible sería: `<? php código php ?>`
- También podemos poner: `<script language="php">código php</script>`

- Y por último: `<% código php %>` al estilo de otro sistema similar de generación de páginas web dinámicas.
- El comportamiento de los programas escritos en PHP es el siguiente:
    - Todos los trozos de código que estén comprendidos entre <? ?>, Apache los pasa al intérprete PHP para que lo ejecute.
    - Todos los trozos de código que estén fuera de los controles <? ?>, Apache los suministra al cliente tal como los encuentra.
    - Los controles de código PHP pueden ir en cualquier posición del programa, pero no pueden anidarse.

Observar que cuando se ha hecho referencia a la forma <? ?> se está refiriendo a cualquier forma válida de separación de los códigos PHP y HTML mencionados anteriormente.

En la figura 2.7 podemos observar el tratamiento que le asigna PHP a las distintas secciones de código que trata en sus programas.

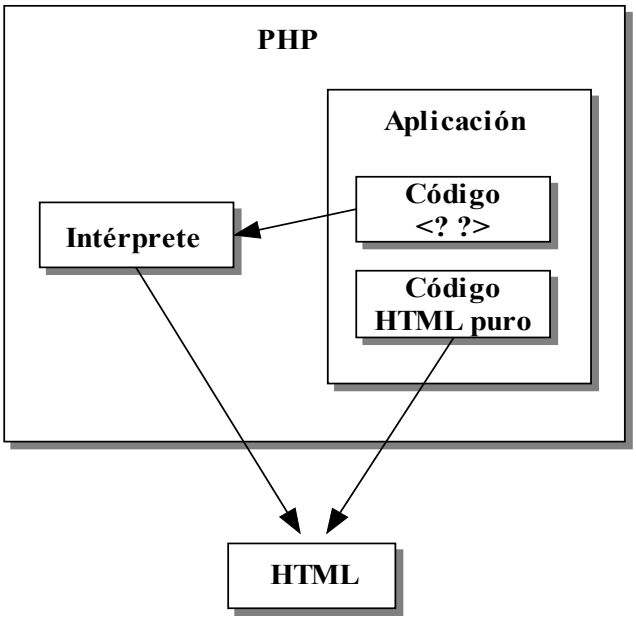


Figura 2.7 Códigos PHP y HTML

Las variables siempre comienzan con el carácter "$" y no precisan ser declaradas. Éstas se adaptan automáticamente al tipo que almacenan, ya sea entero, carácter, cadena, double,... y son sensibles a mayúsculas y minúsculas.

PHP tiene una sintaxis muy parecida a la del C. Para introducir comentarios de una línea se puede usar la doble barra "//" o el símbolo "#". Mediante **/*** y ***/** creamos comentarios multilíneas.

Se pueden crear funciones. Las variables creadas dentro de una función son locales a ésta y el paso de parámetros es el típico en C. Opcionalmente, podemos pasarle parámetros a las funciones que se tratarán como variables locales y, así mismo, podemos devolver un resultado con la instrucción "return valor;". Esto produce la terminación de la función retornando un valor.

Una parte esencial de PHP, es la manipulación y presentación en pantalla de cadenas de texto (*strings*). Una cadena es cualquier conjunto de caracteres entrecomillados. PHP considera como cadena todo lo que encuentre entre un par de comillas, por eso todas las cadenas deben comenzar y terminar con el mismo tipo de comillas, simples o dobles. El lenguaje provee de una amplia librería de funciones para manipular cadenas: conversiones, comparaciones, substituciones...

2.7.5 Sesiones

Una sesión puede definirse como el tiempo durante el cual, un usuario navega en un sitio web. Dicha sesión se crea en el momento en el que el usuario accede a dicho sitio. A partir de ese momento, todas sus peticiones y visitas a las diferentes páginas del sitio web pueden ser seguidas por medio de la sesión que inició.

El uso de las sesiones se hace imprescindible en aquellos websites que ofrecen servicios de personalización o requieren de una identificación por parte del usuario ante unos datos "sensibles".

La acción de sesión, como tal, lleva consigo la generación de un identificador que identificará esa sesión con el usuario que provocó su generación, de tal forma que es posible el realizar un seguimiento de dicho usuario comprobando su identificador de sesión.

Las variables de sesión se diferencian de las variables clásicas en que éstas residen en el servidor, son específicas de un solo usuario definido por un identificador y pueden ser utilizadas en la globalidad de nuestras páginas.

2.7.5.1 Iniciación de la sesión

Las sesiones han de ser iniciadas al principio de nuestro script. Antes de abrir cualquier etiqueta o de imprimir cualquier cosa. En caso contrario, recibiremos un error. Existen dos funciones para inicializar una sesión:

- ***session_start().*** Esta función crea una nueva sesión para un nuevo visitante o bien recupera la que está siendo llevada a cabo.
- ***session_register("NombreVariable1",…,"NombreVariableN").*** Esta función, además de crear o recuperar la sesión para la página en la que se incluye, también sirve para introducir una nueva variable de tipo sesión. Cada variable hallada es registrada como variable global en la sesión en curso para poder ser utilizada mientras dure la sesión.

2.7.5.2 Otras funciones para el manejo de sesiones

Además de las dos funciones vistas en el apartado anterior, para realizar todas las operaciones con sesiones, PHP dispone de un conjunto de funciones encargadas de ello. A continuación describimos algunas de las funciones más utilizadas en el control de sesiones:

- ***session_id().*** Devuelve el identificador de la sesión actual. Si se especifica un *id*, reemplazará el identificador actual por el especificado.

- ***session_is_registered("NombreVariable").*** Indica si una variable ha sido registrada en la sesión. Esta función devuelve TRUE si hay una variable registrada en la sesión actual cuyo nombre es *NombreVariable* o FALSE en caso contrario.

- ***session_unregister("NombreVariable").*** Desregistra (olvida) la variable global llamada *NombreVariable* de la sesión actual. Esta función devuelve TRUE cuando la variable es eliminada de la sesión correctamente o FALSE en el caso contrario.

- ***session_destroy().*** Destruye todos los datos asociados con la sesión actual. Esta función devuelve TRUE si se ha destruido la sesión correctamente o FALSE si ha habido algún problema al intentarlo.

- ***session_unset().*** Elimina y libera el espacio ocupado por todas las variables de la sesión actual registradas.

2.7.6 Funciones de acceso a MySQL

PHP dispone de un conjunto de funciones bastante amplio para la gestión y control de bases de datos MySQL. Entre las más importantes podemos destacar las siguientes:

- ***mysql_affected_rows.*** Devuelve el número de filas afectadas en la última sentencia INSERT, UPDATE o DELETE sobre el servidor asociado con el identificador de enlace especificado. Si el identificador de enlace no ha sido especificado, se asume por defecto el último enlace.

- ***mysql_close.*** Cierra el enlace con la base MySQL que esta asociada con el identificador de enlace especificado. Si no se especifica el identificador de enlace, se asume por defecto el último enlace. Devuelve verdadero si éxito, falso si error.

- ***mysql_connect.*** Establece una conexión a un servidor MySQL. Todos los argumentos son opcionales, y si no hay, se asumen los valores por defecto

("localhost", usuario propietario del proceso del servidor, password vacía). Devuelve un identificador de enlace positivo si tiene éxito, o falso si error.

- ***mysql_error.*** Devuelve el texto del mensaje de error de la última operación MySQL.

- ***mysql_fetch_array.*** Extrae la fila de resultado como una matriz asociativa. Devuelve una matriz que corresponde a la sentencia extraída, o falso si no quedan más filas. Además de guardar los datos en el índice numérico de la matriz, guarda también los datos en los índices asociativos, usando el nombre de campo como clave.

- ***mysql_fetch_object.*** Extrae una fila de resultado como un objeto. Devuelve un objeto con las propiedades que corresponden a la última fila extraída, o falso si no quedan más filas. Es similar a *mysql_fetch_array()*, con la diferencia que devuelve un objeto en lugar de una matriz. Indirectamente, quiere decir que sólo se puede acceder a los datos por el nombre del campo, y no por su posición.

- ***mysql_fetch_row.*** Devuelve una matriz que corresponde a la fila seleccionada, o falso si no quedan más líneas. Selecciona una fila de datos del resultado asociado al identificador de resultado especificado. La fila es devuelta como una matriz. Cada columna del resultado es guardada en un offset de la matriz, empezando por el offset 0.

- ***mysql_query.*** Envía una sentencia a la base activa en el servidor asociado al identificador de enlace. Si no se especifica un *identificador_de_enlace*, se asumirá el último enlace abierto. Si no hay ningún enlace abierto, la función intenta establecer un enlace como si se llamara función *mysql_connect()* sin argumentos, y lo utiliza. Devuelve TRUE (no-cero) o FALSE para indicar si la sentencia se ha ejecutado correctamente o no. Un valor TRUE significa que la sentencia era correcta y pudo ser ejecutada en el servidor. No indica nada sobre el número de filas devueltas. Es perfectamente posible que la sentencia se ejecute correctamente pero que no devuelva ninguna fila.

- ***mysql_result.*** Devuelve el contenido de una celda de un resultado MySQL. Cuando se trabaja con un gran resultado, debe considerarse la utilización de una

función que devuelva una fila entera ya que estas funciones son mucho más rápidas que *mysql_result()*.

- ***mysql_select_db.*** Selecciona una base de datos MySQL. Establece la base activa que estará asociada con el identificador de enlace especificado. Si no se especifica un identificador de enlace, se asume el último enlace abierto. Si no hay ningún enlace abierto, la función intentará establecer un enlace como si se llamara a *mysql_connect()*. Devuelve TRUE si éxito, FALSE si error.

2.8 Bases de Datos / MySQL

2.8.1 Bases de datos

Una base de datos[11] es un conjunto o colección de datos almacenados en un equipo informático de forma integrada y compartida para proporcionar un acceso directo a dicha información, sin redundancias perjudiciales o innecesarias. Los datos deben estar interrelacionados y estructurados, de forma que, se elimine del todo o en parte cualquier redundancia entre los mismos.

2.8.1.1 Modelo de datos

Un modelo de datos es un conjunto de conceptos que permiten describir, a distintos niveles de abstracción, la estructura de una base de datos, a la cual denominamos esquema. Según el nivel de abstracción de la arquitectura ANSI a tres niveles en el que se encuentre la estructura descrita, el modelo que permite su descripción será un modelo externo, conceptual o interno.

- ***Nivel externo:*** Orientado hacia el usuario en particular, comprende los datos que necesita con las estructuras propias del lenguaje de programación que va a emplear.
- ***Nivel conceptual:*** Ayuda a describir los datos para el conjunto de usuarios.

- ***Nivel interno:*** Orientado a la máquina.

2.8.1.2 Diseño de una base de datos

Entidad

Se puede definir una entidad como cualquier objeto real o abstracto que existe en la realidad, del cual queremos almacenar información en la base de datos.

Llamaremos tipo de entidad a la estructura genérica y ocurrencia de entidad a cada una de las realizaciones concretas de ese tipo de entidad.

La representación gráfica de un tipo de entidad es un rectángulo etiquetado con el nombre del tipo de entidad. Entre las entidades se pueden diferenciar dos tipos:

- ***Entidades Fuertes o Propias:*** Aquellas cuyas ocurrencias son identificables por sí mismas. Los atributos que las identifican son propios de la entidad.
- ***Entidades Débiles o Regulares:*** Aquellas cuyas ocurrencias son identificables solamente por estar asociadas a otra u otras entidades. Algunos de los atributos que las identifican se refieren a otra entidad.

Interrelación

Se entiende por interrelación una asociación, vinculación o correspondencia entre entidades. Denominaremos tipo de interrelación a la estructura genérica que describe un conjunto de interrelaciones. En un tipo de interrelación se pueden distinguir los siguientes elementos:

- ***Relación:*** Una relación es una asociación de varias entidades. Se representa mediante un rombo.

- ***Grado:*** Es el número de tipos de entidad que participan en un tipo de interrelación. Puede ser de grado 2 cuando se asocian dos tipos de entidad; de grado 3 cuando se asocian tres tipos de entidad; o en general de grado n.
- ***Tipo de correspondencia:*** Número máximo de ocurrencias de un tipo de entidad que pueden intervenir por cada ocurrencia del otro tipo de entidad asociado en la interrelación. El tipo de correspondencia es:
    - ***1:1*** (una a una): A cada ocurrencia de una entidad corresponde no más de una ocurrencia de la otra y a la inversa.
    - ***1:n*** (una a muchas): A cada ocurrencia de la primera entidad pueden corresponderle varias ocurrencias de la segunda y a cada ocurrencia de la segunda le corresponde no más de una de la primera.
    - ***n:m*** (muchas a muchas): A cada ocurrencia de la primera entidad pueden corresponderle más de una ocurrencia de la segunda y viceversa.

Atributo

Cada una de las propiedades o características que tiene una entidad o una interrelación se denomina atributo. Se representa mediante una elipse.

Cardinalidad de una entidad

La cardinalidad de una entidad en una relación mide el número máximo y mínimo de la participación de dicha entidad en la relación. Utilizaremos la siguiente notación para expresar la cardinalidad:

- ***1:*** Indica que a cada elemento le corresponde otro en la otra entidad (obligatoriedad).
- ***c:*** Indica que a un elemento de una entidad le puede corresponder uno o ningún elemento de la otra entidad (no obligatoriedad).

- ***m:*** Indica que a un elemento de una entidad le pueden corresponder uno o más elementos de la otra entidad (obligatoriedad).
- ***n:*** Indica que a un elemento de una entidad le puede corresponder ninguno, uno o más elementos de la otra entidad (no obligatoriedad).

Claves

Una *clave candidata* de una relación es un conjunto de atributos que identifican unívoca y mínimamente cada tupla (fila) de la relación. Siempre hay al menos una clave candidata. Una relación puede tener más de una clave candidata, se pueden distinguir entre:

- ***Clave primaria:*** Aquella clave candidata para identificar las tuplas de la relación. Cuando sólo existe una clave candidata, ésta será la clave primaria.
- ***Claves alternativas:*** Aquellas claves candidatas que no han sido escogidas como clave primaria.

La *clave ajena* de una relación es aquel o aquellos atributos de una entidad que son clave primaria en otra.

Restricciones

Existen restricciones, estructuras u ocurrencias no permitidas, siendo preciso distinguir entre restricciones inherentes y restricciones semánticas.

- ***Restricciones inherentes:*** Son las restricciones derivadas de la misma estructura del modelo, que no tienen que ser definidas por el usuario e imponen limitaciones a la hora de modelar el mundo real. Las características propias que se han de cumplir son:
    - No hay dos tuplas (o filas) iguales.

- El orden de las tuplas no es significativo.
- El orden de los atributos no es significativo.
- Cada atributo sólo puede tomar un único valor del dominio sobre el que está definido, no admitiéndose por tanto los grupos repetitivos. Se dice que una tabla que cumpla esta condición está normalizada.

- ***Restricciones semánticas:*** Son facilidades que el modelo ofrece a los usuarios a fin de que éstos puedan reflejar en el esquema, lo más fielmente posible, la semántica del mundo real.

Grafo relacional

Es una forma de representar el esquema relacional. Cada esquema de relación representa una tabla. Ha de aparecer como mínimo, su nombre y sus atributos, indicando su clave primaria, las claves alternativas y las claves ajenas.

Normalización

Con la teoría de la normalización se consigue una formalización en el diseño lógico de bases de datos relacionales, para desarrollar programas que automaticen el diseño. Un diseño inadecuado puede presentar los siguientes problemas:

- Incapacidad para almacenar ciertos hechos.
- Redundancias y posibilidad de inconsistencias.
- Ambigüedades.
- Pérdida de información (aparición de tuplas espurias).
- Pérdida de ciertas restricciones de integridad que dan lugar a independencias entre los datos.

- Aparición en la base de datos, como consecuencia de las redundancias, de estados que no son válidos en el mundo real; es lo que se llama anomalías de inserción, borrado y modificación.

- ***Formas normales:*** La teoría de la normalización se centra en lo que se conoce como formas normales. Un esquema de relación está en una determinada forma normal si satisface un conjunto específico de restricciones.

    - ***Primera forma normal (1FN):*** Una tabla está en 1FN si, y sólo si los valores que componen el atributo de una tupla son atómicos. Es decir, en un atributo no deben aparecer valores repetitivos y por tanto tienen que ser elementales y únicos.

    - ***Segunda forma normal (2FN):*** Se dice que una relación está en 2FN si:

        - Se encuentra en 1FN.
        - Cada atributo no principal tiene dependencia funcional completa respecto a cada una de las claves.

    - ***Tercera forma normal (3FN):*** Se dice que una relación está en 3FN si:

        - Se encuentra en 2FN.
        - No existe ningún atributo no principal que dependa transitivamente de alguna de las claves de relación.

2.8.2 SQL. Structured Query Language

El lenguaje SQL[12] (Lenguaje de Consulta Estructurado) es un lenguaje estándar de comunicación con bases de datos. Por tanto, se trata de un lenguaje normalizado que nos permite trabajar con cualquier tipo de lenguaje (ASP o PHP) en combinación con cualquier tipo de base de datos (MS Access, SQL Server, MySQL,...).

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

2.8.2.1 Comandos

Existen dos tipos de comandos SQL:

- Los DLL que permiten crear y definir nuevas bases de datos, campos e índices.
- Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DLL

- ***CREATE:*** Utilizado para crear nuevas tablas, campos e índices.
- ***DROP:*** Empleado para eliminar tablas e índices.
- ***ALTER:*** Utilizado para modificar las tablas agregando campos o cambiando la definición de éstos.

Comandos DML

- ***SELECT:*** Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
- ***INSERT:*** Utilizado para cargar lotes de datos en la base de datos en una única operación.
- ***UPDATE:*** Utilizado para modificar los valores de los campos y registros especificados.
- ***DELETE:*** Utilizado para eliminar registros de una tabla de una base de datos.

2.8.2.2 Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que se desean seleccionar o manipular.

- ***FROM:*** Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
- ***WHERE:*** Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
- ***GROUP BY:*** Utilizada para separar los registros seleccionados en grupos específicos.
- ***HAVING:*** Utilizada para expresar la condición que debe satisfacer cada grupo.
- ***ORDER BY:*** Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

2.8.2.3 Operadores lógicos

- ***AND:*** Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
- ***OR:*** Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
- ***NOT:*** Negación lógica. Devuelve el valor contrario de la expresión.

2.8.2.4 Operadores de comparación

- ***"<":*** Menor que.
- ***">":*** Mayor que.
- ***"<>":*** Distinto de:

- ***"<=":*** Menor o igual que.
- ***">=":*** Mayor o igual que.
- ***"=":*** Igual que.
- ***BETWEEN:*** Utilizado para especificar un intervalo de valores.
- ***LIKE:*** Utilizado en la comparación de un modelo.
- ***IN:*** Utilizado para especificar registros de una base de datos.

2.8.2.5 Funciones de agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

- ***AVG:*** Utilizada para calcular el promedio de los valores de un campo determinado.
- ***COUNT:*** Utilizada para devolver el número de registros de la selección
- ***SUM:*** Utilizada para devolver la suma de todos los valores de un campo determinado
- ***MAX:*** Utilizada para devolver el valor más alto de un campo especificado.
- ***MIN:*** Utilizada para devolver el valor más bajo de un campo especificado.

2.8.3 MySQL

2.8.3.1 Introducción

MySQL[13] es un sistema de gestión de bases de datos relacionales. Se diseñó para proporcionar accesos rápidos a datos dentro de pequeños sistemas. MySQL comprende un servidor de bases de datos multi-hilo y multiusuario, además de varias herramientas que permiten a un usuario o aplicación cliente, comunicarse con dicho servidor.

Utiliza SQL (Structured Query Language) como lenguaje de consulta haciendo uso de los estándares ANSI SQL92 y ODBC 2.5.

La filosofía de MySQL es proporcionar un sistema de base de datos capaz de manejar rápidamente tareas simples. Su software es capaz de obtener buenos resultados, aunque se disponga de muy pocos recursos del sistema. La mayoría de las bases de datos actuales requieren grandes plataformas hardware e inmensas cantidades de memoria antes de que puedan proporcionar buenos resultados en el acceso a datos almacenados.

MySQL se ha diseñado para proporcionar resultados excepcionales aún trabajando en pequeñas plataformas (como PC's). Su gestor de bases de datos ha sido diseñado para manejar grandes cantidades de datos (alrededor de 50 millones de registros), sin perder rapidez en el acceso, además de proporcionar hasta un número prácticamente ilimitado de conexiones clientes. Debido a estas características, MySQL satisface todas nuestras pretensiones para la realización de las aplicaciones.

La gran ventaja de MySQL sobre otras bases de datos es su fácil conectividad en entornos Web. Actualmente, multitud de lenguajes de programación, tales como Perl o PHP, soportan el acceso directo a las bases de datos MySQL. Esto hace que la combinación MySQL/PHP permita acceder al contenido de la base de datos directamente desde el código HTML.

2.8.3.2 Sistema de privilegios

El sistema de privilegios de acceso garantiza que todos los usuarios realicen sólo las operaciones que les son permitidas. Cuando un usuario se conecta al servidor MySQL, su identidad es determinada por el host, desde el cual se realiza la conexión y el nombre de usuario especificado. El sistema concede privilegios de acuerdo con estos datos.

El sistema de privilegios está basado en el contenido de 5 tablas de la base de datos mysql: *host*, *user*, *db*, *tables_priv*, *colums_priv*.

La tabla *user* contiene información sobre los usuarios, desde que máquinas pueden acceder a nuestro servidor MySQL, su clave y de sus diferentes permisos.

La tabla *host* nos informa sobre qué máquinas podrán acceder a nuestro sistema, así como a las bases de datos que tendrán acceso y sus diferentes permisos.

Finalmente, las tablas *db*, *tables_priv* y *columns_priv* nos proveen de un control individual de las bases de datos, tablas y columnas (campos).

He aquí una breve descripción de los diferentes permisos:

- ***Select_priv:*** Permite utilizar la sentencia SELECT
- ***Insert_priv:*** Permite utilizar la sentencia INSERT
- ***Update_priv:*** Permite utilizar la sentencia UPDATE
- ***Delete_priv:*** Permite utilizar la sentencia DELETE
- ***Create_priv:*** Permite utilizar la sentencia CREATE o crear bases de datos
- ***Drop_priv:*** Permite utilizar la sentencia DROP o eliminar bases de datos
- ***Reload_priv:*** Permite recargar el sistema mediante mysqladmin reload
- ***Shutdown_priv:*** Permite parar el servidor mediante mysqladmin shutdown
- ***Process_priv:*** Permite manejar procesos del servidor
- ***File_priv:*** Permite leer y escribir ficheros usando comando como SELECT INTO OUTFILE y LOAD DATA INFILE
- ***Grant_priv:*** Permite otorgar permisos a otros usuarios
- ***Index_priv:*** Permite crear o borrar índices
- ***Alter_priv:*** Permite utilizar la sentencia ALTER TABLE

Si dejamos en blanco los campos User, Host o Db, haremos referencia a cualquier usuario, servidor o base de datos. Conseguiremos el mismo efecto poniendo el símbolo % en el campo.

Un Host debe ser un "host local", un número IP, o una expresión SQL. Si en la tabla *db* la columna Host está vacía, significa cualquier host de la tabla host. Si en la tabla *host* o *user* dicha columna está vacía, significa que cualquier host puede crear una conexión TCP con nuestro servidor.

En la instalación se crea el usuario root sin contraseña, por lo que se debe poner una contraseña para éste. Se debe crear distintos usuarios con distintos permisos y poner una contraseña para todos ellos. Entre ellos, el usuario administrador de MySQL, con todos los permisos, y como recomendación de seguridad, el usuario nobody sólo con el permiso de ver (SELECT).

2.8.3.3 Tipos de datos

Cada columna en una tabla está hecha de un tipo de datos. Hay tres tipos generales de datos: tipos numéricos, tipos de fechas y horas y tipos de cadenas de caracteres.

A continuación se describen estos tipos y sus principales características. La terminología utilizada es la siguiente:

- M indica el tamaño máximo a mostrar. El máximo es de 255.
- D indica el número de dígitos que siguen al punto decimal. El valor máximo es 30.
- Los corchetes (‘[‘ y ‘]’) indican partes opcionales.

Tipos numéricos

- ***INT[(M)]:*** Un entero de tamaño normal. El rango con signo es -2147483648 a 2147483647. El rango sin signo es 0 a 4294967295.
- ***FLOAT[(M,D)]:*** Un número de punto flotante de precisión simple. No puede ser sin signo. Se permite valores entre -3.402823466E+38 y -1.175494351E-38. Los números de punto flotante están hechos para ser muy precisos. Por ejemplo, si se especifica FLOAT(6,2), entonces se permitirían seis números a la izquierda del punto decimal y 2 a la derecha.

Fecha y hora

- ***DATE:*** Una fecha. El rango soportado es '1000-01-01' a '9999-12-31'. MySQL pone las fechas en formato 'YYYY-MM-DD', pero permite asignar valores a columnas DATE usando cadenas o números.
- ***TIME:*** Una hora. Este rango es '-838:59:59' a '838:59:59'. MySQL pone las horas en formato 'HH:MM:SS', pero permite asignar valores para columnas TIME usando cadenas o números.

Cadenas de caracteres

- ***CHAR(M) [BINARY]:*** Son cadenas de caracteres de longitud fija a las que se añade espacios a la derecha hasta llegar a la longitud que especificó en M. Estas cadenas se compararán por MySQL en forma no sensible a las mayúsculas, a menos que se le dé la palabra clave BINARY. El tamaño máximo es de 255 caracteres.
- ***VARCHAR(M) [BINARY]:*** Son cadenas de longitud variable. Todos los espacios extra se removerán cuando el valor sea almacenado en la base de datos. La longitud máxima es de 255 caracteres.

- ***BLOB o TEXT:*** Las columnas BLOB o TEXT tienen una longitud máxima de 65535(2^{16} - 1). Pueden almacenar texto y son buenos manteniendo registros grandes. La diferencia es que las búsquedas BLOB son sensibles a las mayúsculas, mientras que las búsquedas TEXT no. Estas columnas son usadas cuando los registros están entre 255 y 65535 caracteres.

3 Análisis previo

3.1 Introducción

Hay varias características del lenguaje C que hacen de éste, un lenguaje peculiar en el mundo de la programación. Nació en los años setenta y, hoy en día, sigue siendo un lenguaje bastante utilizado por muchos programadores. Fue normalizado por parte del *American National Standards Institute* y se ajusta a la definición de ANSI C.

Existe una gran relación entre el lenguaje C y Linux ya que las primeras implementaciones de UNIX se hicieron en lenguaje ensamblador, pero en 1973 se rescribió el sistema en lenguaje C, el cual fue creado para recodificar UNIX de una forma independiente de la máquina.

Hoy en día, es un lenguaje de propósito general. Se ha utilizado para el desarrollo de aplicaciones tan dispares como: hojas de cálculos, gestores de bases de datos, compiladores, etc...

Es un lenguaje de medio nivel. Este lenguaje permite programar a alto nivel (pensado a nivel lógico y no en la máquina física) y a bajo nivel (con lo que se puede obtener la máxima eficiencia y un control absoluto de cuanto sucede en el interior del ordenador).

El lenguaje C es portátil. Los programas escritos en C son fácilmente transportables a otros sistemas.

Es un lenguaje potente y eficiente. Usando C, un programador puede casi alcanzar la eficiencia del código ensamblador.

A todas estas características, hay que añadir que este lenguaje impone poca disciplina y da mucha libertad. Por todo esto, es muy común la enseñanza de este lenguaje en las diferentes carreras universitarias con asignaturas relacionadas con el aprendizaje de la programación.

También hay que resaltar las ventajas que ofrece la programación orientada a objetos. Esto, unido a las características del lenguaje C anteriormente mencionadas, hace destacar importancia al aprendizaje del lenguaje C++.

3.2 Definición del problema

La asignatura *Programación Avanzada* es una asignatura perteneciente a la titulación de *Ingeniero Técnico de Telecomunicación*, especialidad de *Telemática*, de la *ULPGC*. Esta asignatura pretende ampliar los conocimientos de los alumnos en programación y la incorporación de éstos a la programación orientada a objetos a medida que aprenden dos lenguajes, el lenguaje C y el lenguaje C++.

Para afianzar estos conocimientos aprendidos, existe una planificación de prácticas a desarrollar en el laboratorio.

El temario práctico es evaluado mediante la presentación de un trabajo por cada una de las prácticas establecidas. Estos trabajos disponen de un plazo máximo de presentación, establecido actualmente, en una semana antes a la realización del examen teórico. Para la realización de las prácticas es obligatoria la asistencia al laboratorio en el horario asignado. Para poder tener acceso al examen de la asignatura es indispensable haber realizado las prácticas en el curso académico en cuestión.

Hoy en día, la *Escuela Universitaria de Ingeniería Técnica de Telecomunicaciones* de la *ULPGC* dispone de un laboratorio que cuenta con equipos en donde el alumno puede

realizar las prácticas de *Programación Avanzada*. Estos equipos tienen instalado el compilador del lenguaje C y C++ que convierte el código fuente en un fichero objeto y éste en un fichero ejecutable para su posterior ejecución. En caso de no serle suficiente el horario asignado, el alumno puede hacer uso, siempre y cuando le sea factible, del horario libre del laboratorio.

Una vez realizadas las prácticas, y dentro del plazo límite establecido para su presentación, el alumno debe hacer entrega de los trabajos correspondientes al profesor responsable de la asignatura. Este proceso se realiza de forma manual mediante la entrega de un CD o de un disquete. Únicamente mediante esta entrega, el profesor puede observar el trabajo realizado por el alumno para, posteriormente, efectuar su evaluación.

En resumen, en las prácticas de la asignatura de *Programación Avanzada*, existe una serie de inconvenientes que se le presenta, tanto al alumno, como al profesor. Por un lado, el alumno está obligado a un horario dictaminado por la Escuela para el uso de sus equipos. Por otro lado, existe la necesidad del traslado físico para presentar las prácticas al profesor. Dependiendo sólo de estas entregas, el profesor será capaz de valorar el trabajo realizado por el alumno.

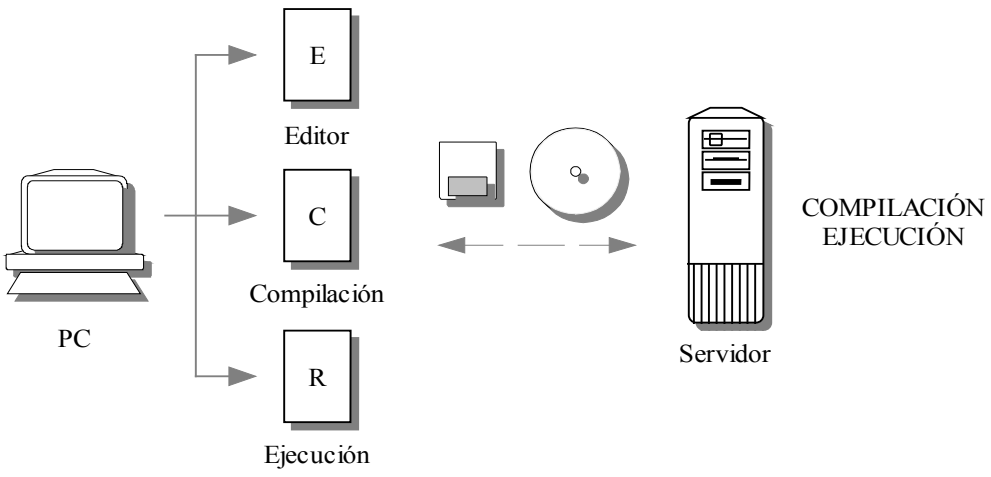


Figura 3.1 Sistema actual

3.3 Posibles soluciones

Con la metodología seguida en el sistema actual establecido, surge una serie de consideraciones a tener en cuenta.

La imposición, por parte de la Escuela, a que el alumno deba asistir al laboratorio para realizar las prácticas, no excluye la posibilidad de que el alumno pueda disponer de un equipo fuera del entorno universitario con el que pueda avanzar dicho trabajo. En cualquier caso, la máquina utilizada para ejecutarlas debe cumplir con un requisito indispensable. Para que se pueda ejecutar un programa escrito en C o en C++, se necesita que el equipo tenga instalado el compilador del lenguaje.

Según lo anterior, si el alumno no desea estar restringido solamente a un horario determinado, se hace necesario tener que conseguir el compilador y el aprendizaje de su instalación.

Otra de las cosas a tener en cuenta, es la incapacidad, por parte del profesor, de poder ver, en todo momento, el trabajo realizado por el alumno durante el transcurso del curso escolar. Esto conlleva a que el alumno no pueda discutir, de manera óptima, el código implementado. En consecuencia, se podría plantear la obligatoriedad de que los trabajos sean presentados en diferentes etapas, y por tanto, que el alumno deba hacer las entregas correspondientes. Con ello, el profesor vería el trabajo del alumno cada cierto tiempo, según los intervalos entre las etapas.

De igual manera, la entrega manual de las prácticas de forma periódica a través de un CD o de un disquete, no se considera óptima. La pérdida de estos elementos, su deterioro y la tendencia a confundir unos con otros, son algunos de los motivos.

Lo que se pretende es desarrollar un entorno web que posibilite obtener el máximo provecho a las prácticas con el menor número de inconvenientes, tanto para el alumno, como para el profesor responsable. Asimismo, este entorno debe satisfacer las necesidades de la asignatura.

El entorno web hace innecesaria la entrega manual del código, sustituyendo la copia del mismo en un CD o disquete, por la copia directa en un equipo remoto, al cual se puede tener acceso desde cualquier máquina con acceso a la red. Por consiguiente, el profesor podría consultar, en todo momento, el trabajo realizado por el alumno accediendo a dicho equipo.

También, a través de este entorno, el alumno podría implementar el código y mandarlo a ejecutar desde cualquier máquina con Internet, sin tener que estar instalado el compilador en la máquina utilizada, ya que la ejecución tendría lugar en dicho equipo remoto. El alumno queda, por tanto, libre de ninguna instalación y de ninguna restricción horaria por parte de la Escuela, salvo la correspondiente a su horario de prácticas obligatorio.

3.4 Solución adoptada

Del apartado anterior, se resuelve que, la solución adoptada para resolver el listado de problemas surgidos en el sistema actual, es la creación de un entorno web.

El modelo básico de implementación que se propone consta de:

- Un editor de texto desde el cual se pueda crear todo tipo de ficheros, permitiendo guardar archivos con extensión .c y .cpp para su compilación. Al mismo tiempo, a través de este editor, el usuario podrá visualizar el contenido de todos los ficheros ya creados.
- Un administrador de archivos con el cual, el usuario gestione todos sus ficheros, desde su creación hasta su eliminación.
- Un módulo de compilación y ejecución que permita configurar el modo de ejecución, permitiendo el paso de parámetros, ficheros de entrada/salida y tiempo máximo permitido para la ejecución.
- Un administrador de procesos mediante el cual, el usuario gestione sus procesos lanzados al servidor. Desde este administrador, se podrá visualizar un listado de

todos los procesos lanzados, distinguiendo los que aún continúan vivos de los que hayan finalizado. Además, el usuario podrá acabar con aquellos que sigan en ejecución.

Características necesarias a implementar:

- El entorno debe proporcionar la posibilidad de guardar los ficheros creados por el usuario en el servidor sin necesidad de contar con terceras aplicaciones como el FTP.
- El acceso debe realizarse mediante un navegador web y posibilitar su uso a través de Internet. Como se va a posibilitar el acceso al entorno a través de Internet, habrá que restringir el acceso a la información a usuarios no autorizados. Por ello, se incorporará un sistema de seguridad que actuará permitiendo ver el contenido de las páginas o denegando su visualización, según el usuario esté o no registrado. Además, sería recomendable poder crear distintos tipos de usuario, incluyendo un perfil de administrador que permita la gestión del resto.
- También debe permitirse, desde el cliente, la compilación y ejecución en el servidor mostrando al cliente el resultado.

Características adicionales que son deseables:

- Un sistema de alarma ante procesos vivos en el servidor.
- Una herramienta de publicación y visualización de noticias/comentarios.

Lenguajes a utilizar:

- Se utilizará el lenguaje HTML como base para la definición y creación de las páginas, apoyado en las Hojas de Estilos CSS.
- Junto al código HTML se añadirán scripts en JavaScript para ciertos detalles.

- Se utilizará como lenguaje de programación el lenguaje PHP añadiendo información dinámica al código HTML. También se utilizará el lenguaje C y scripts de Linux para controlar los procesos en el servidor.
- Se utilizará como Sistema de Gestión de Bases de Datos, MySQL.

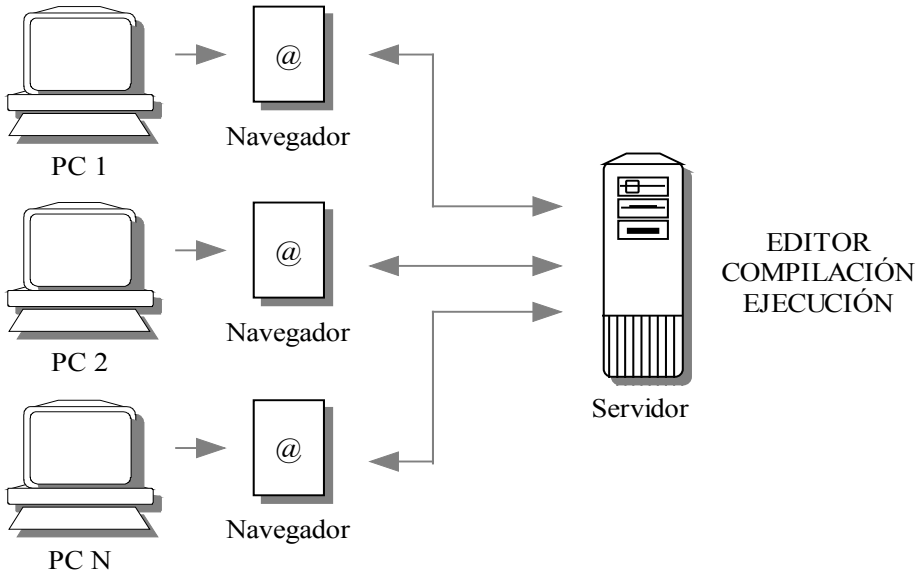


Figura 3.2 Sistema a implementar

4 Análisis funcional

4.1 Introducción

El *Análisis funcional* analiza los requerimientos de las aplicaciones, las descripciones a muy alto nivel de las actividades ejecutadas dentro del sistema y el flujo de datos que se intercambia entre estas actividades.

Mediante esta tarea se identifican los diferentes subsistemas que componen el sistema procedente del *Análisis previo* y se determinan las necesidades de cada uno de ellos. Cada uno de estos subsistemas se divide en otros más pequeños que reciben el nombre de Unidad Funcional. Las Unidades Funcionales se definen, de tal forma, que tengan un alto porcentaje de independencia.

4.2 Mapa funcional

El mapa funcional, o árbol funcional, es la representación gráfica de los resultados del análisis funcional. Su forma en "árbol", dispuesto horizontalmente, refleja la metodología seguida para su elaboración en la que, una vez definido el propósito clave, éste se desagrega sucesivamente en las funciones constitutivas.

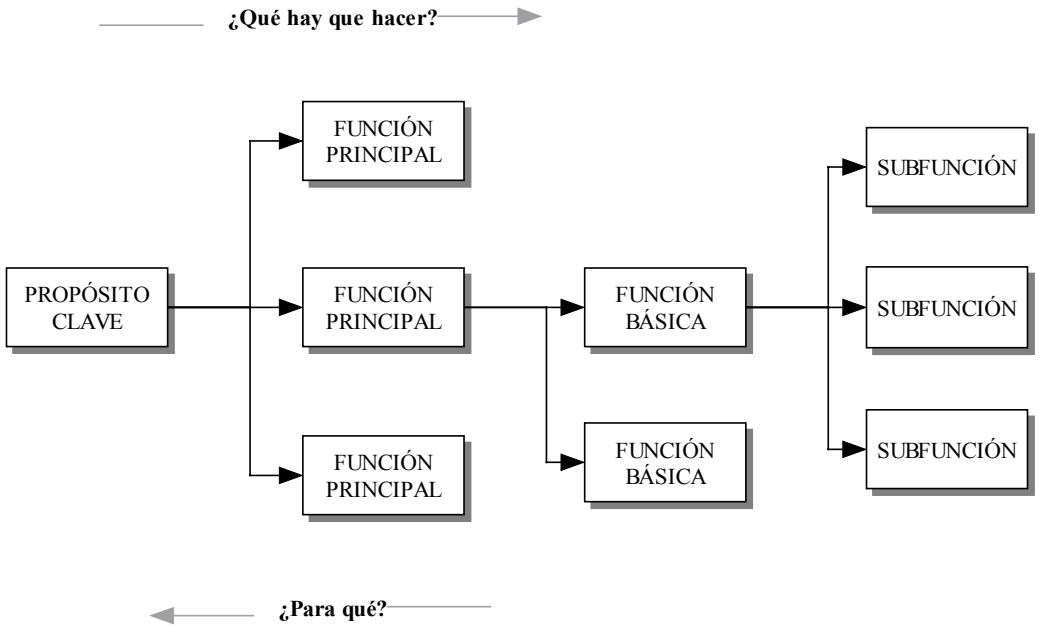


Figura 4.1 Esquema general de un mapa funcional

4.2.1 Mapa funcional de la aplicación

Basándonos en el modelo de la solución adoptada en el análisis previo, podemos subdividir la aplicación en bloques principales bien diferenciados.

En la siguiente figura se muestra el mapa funcional de la aplicación:

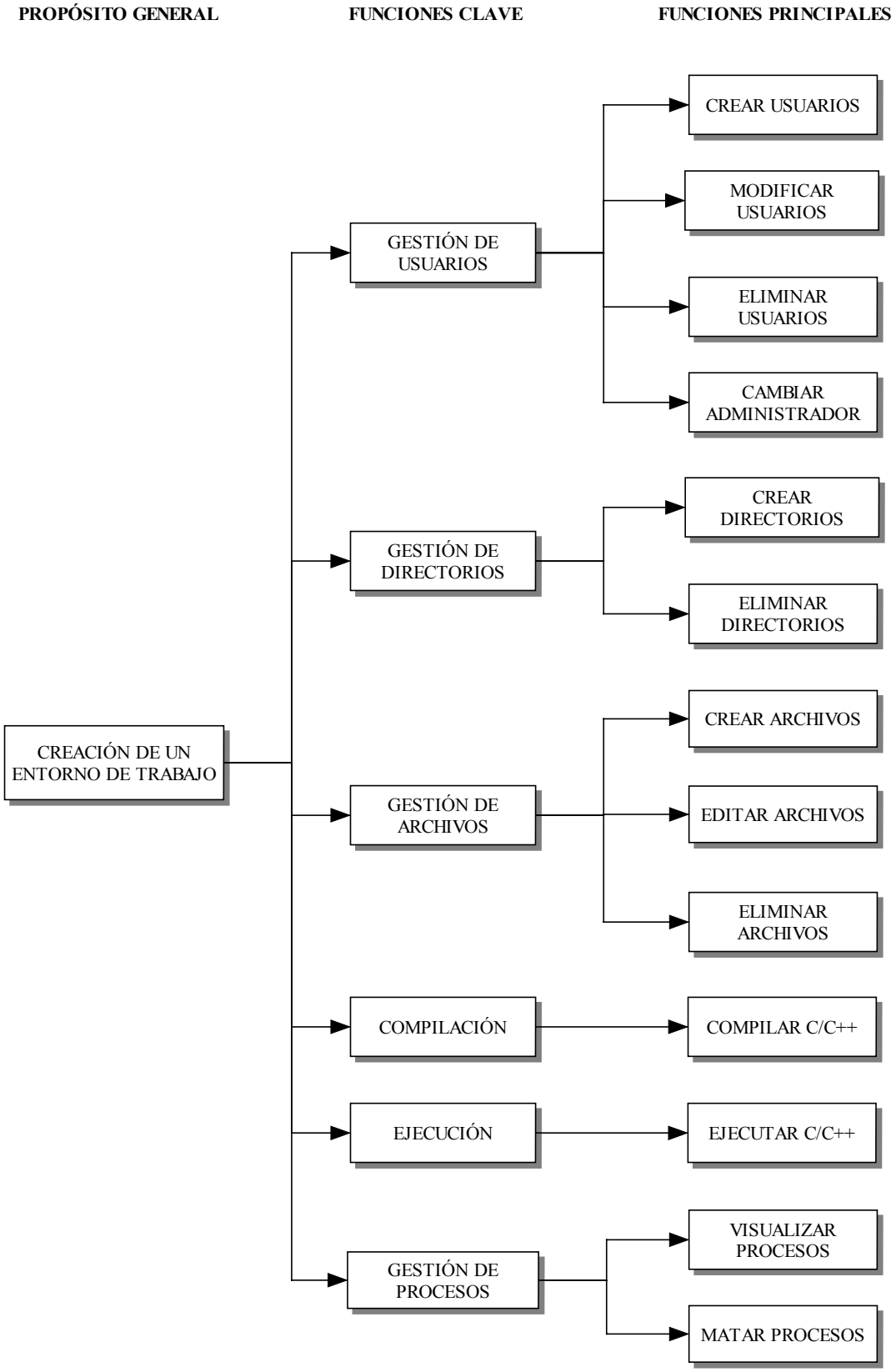


Figura 4.2 Mapa funcional de la aplicación

4.3 Organigramas funcionales

4.3.1 Gestión de usuarios

Los datos pertenecientes a los usuarios de la aplicación se almacenan en dos tablas dentro de un sistema de bases de datos MySQL. El módulo de *Gestión de usuarios* realiza funciones de lectura, inserción, edición y eliminación sobre dichas tablas. Además, se realizan operaciones de lectura y escritura en el sistema de archivos del servidor, donde se crea la estructura de directorios perteneciente a los usuarios de la aplicación.

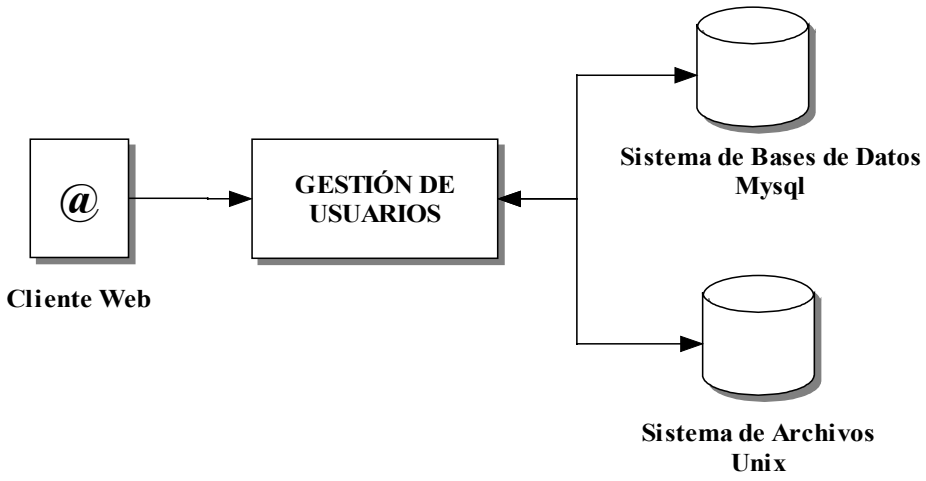


Figura 4.3 Organigrama funcional del módulo de Gestión de usuarios

4.3.2 Gestión de directorios

El módulo de *Gestión de directorios* realiza operaciones de lectura y escritura de directorios sobre la carpeta de trabajo de cada usuario.

Figura 4.4 Organigrama funcional del módulo de Gestión de directorios

4.3.3 Gestión de archivos

El módulo de *Gestión de archivos* realiza operaciones de lectura y escritura de archivos sobre la carpeta de trabajo de cada usuario.

Figura 4.5 Organigrama funcional del módulo de Gestión de archivos

4.3.4 Compilación

Este módulo permite al usuario compilar los ficheros pertenecientes a su carpeta de trabajo. En él se incluyen las distintas etapas que cubre el compilador para obtener el código ejecutable. Realiza operaciones de escritura sobre el directorio de trabajo de cada usuario.

Figura 4.6 Organigrama funcional del módulo de Compilación

4.3.5 Ejecución

El módulo de *Ejecución* realiza funciones de lectura y escritura sobre el directorio personal de cada usuario, permitiendo la ejecución de los ficheros pertenecientes a la carpeta de trabajo.

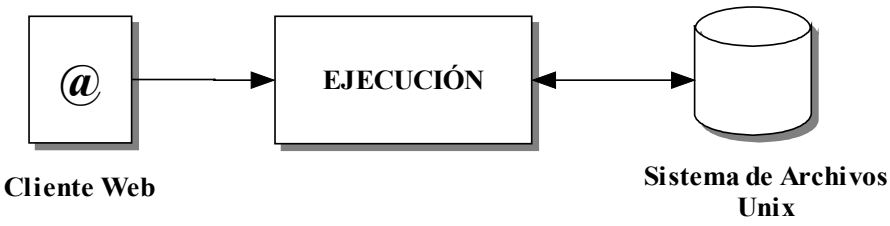


Figura 4.7 Organigrama funcional del módulo de Ejecución

4.3.6 Gestión de procesos

El módulo de *Gestión de procesos* realiza operaciones de lectura y escritura sobre un único fichero perteneciente a uno de los directorios de la carpeta personal de cada usuario.

Figura 4.8 Organigrama funcional del módulo de Gestión de Procesos

4.4 Gestión de usuarios

La *Gestión de usuarios* consiste en la administración de todos los usuarios pertenecientes a la aplicación. Esta función sólo la puede desempeñar un único usuario con permiso de gestor. Las tareas que puede realizar son las siguientes:

- Crear nuevas cuentas de usuarios con el propósito de permitirles el acceso a la aplicación.
- Modificar los datos de los usuarios dados de alta.
- Eliminar la cuenta de un usuario, anulando de esta forma su acceso.
- Activar cuentas de usuario que estén desactivadas y viceversa.
- Visualizar los usuarios que pertenecen a la aplicación.
- Ordenar a todos los usuarios por un determinado dato de los mismos.
- Cambiar el administrador de la aplicación.

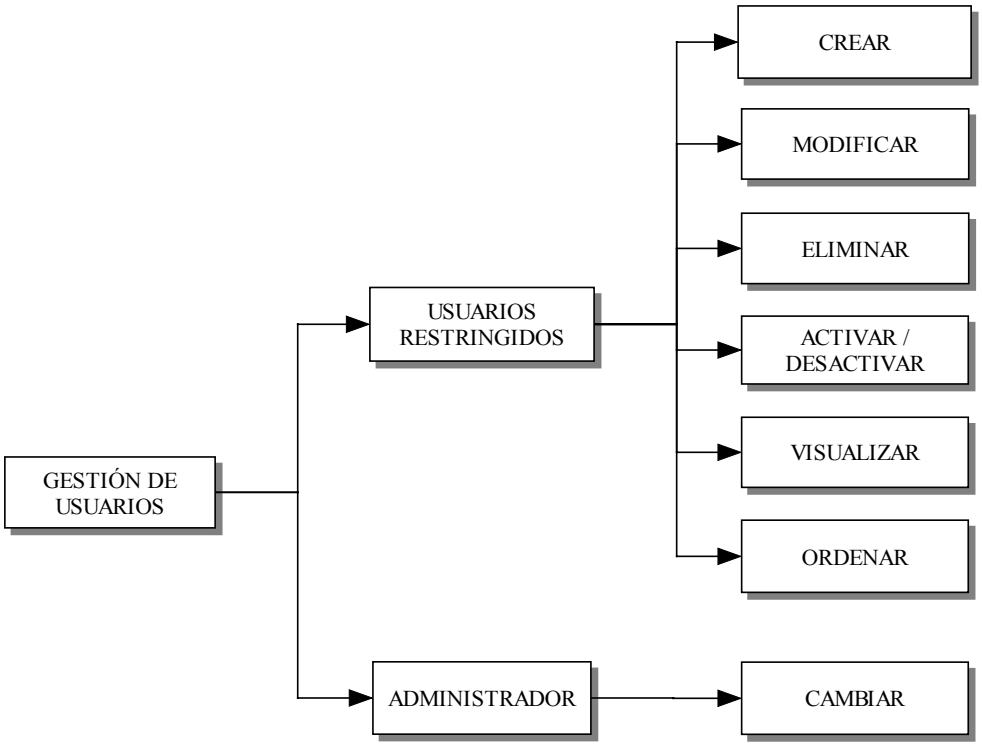


Figura 4.9 Unidades funcionales del módulo de Gestión de usuarios

Esta *Gestión de usuarios* constituye uno de los ejes fundamentales de la aplicación, ya que de ella depende quiénes podrán tener acceso a las diferentes opciones del sistema.

Existen dos tipos de usuarios:

- ***Administrador:*** Un único usuario en la aplicación con permiso de administración. Este usuario tiene acceso a todas las diferentes opciones que presenta la aplicación sin ninguna restricción. Es el encargado de realizar el proceso de gestión de usuarios.
- ***Usuarios restringidos:*** Usuarios sin permiso de administración. Sólo tienen acceso a la interfaz de trabajo y no a la de administración.

4.4.1 Gestión de usuarios restringidos

4.4.1.1 Crear cuentas de usuarios

Los datos de las cuentas de cada usuario son guardados en una base de datos pertenecientes al módulo de *Gestión de usuarios*.

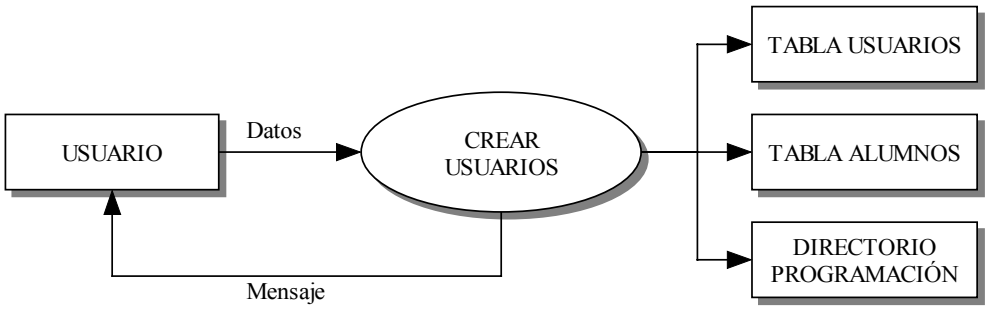


Figura 4.10 Flujo de datos de la unidad funcional Crear usuarios

A la vez que se crea una nueva cuenta de usuario, se crea una estructura de directorios en el sistema de ficheros UNIX como se muestra en la figura 4.11.

Los directorios raíz de los usuarios cuelgan de un directorio padre llamado *Programación*. Cada usuario tiene un directorio raíz referenciado por el dato NIF introducido al crearse su cuenta. De este directorio cuelgan cinco subdirectorios:

- ***Configuración:*** directorio en el que se guardan los archivos de configuración que el usuario va creando dentro de la aplicación. Existe una opción donde el usuario puede visualizar el contenido del mismo.
- ***Ejecución:*** en este directorio se almacenan todos los archivos de salida especificados por el usuario en los ficheros de configuración. La aplicación ofrece una opción donde visualizar el contenido de este directorio.
- ***Prácticas:*** directorio en el que cada usuario guarda sus archivos y carpetas personales. Es el directorio raíz mostrado en la interfaz de trabajo.

- ***Procesos:*** en él se encuentran dos archivos; uno llamado *ejecucion.txt*, donde se va almacenando información sobre todos los procesos lanzados por el usuario, y otro llamado *stderr.txt*, donde se guardan los resultados de la compilación de archivos.
- ***Temporal:*** el directorio *Temporal* contiene un único fichero, llamado *temporal.txt*, cuyo contenido es el del último archivo perteneciente al directorio *Prácticas* que haya guardado el usuario.

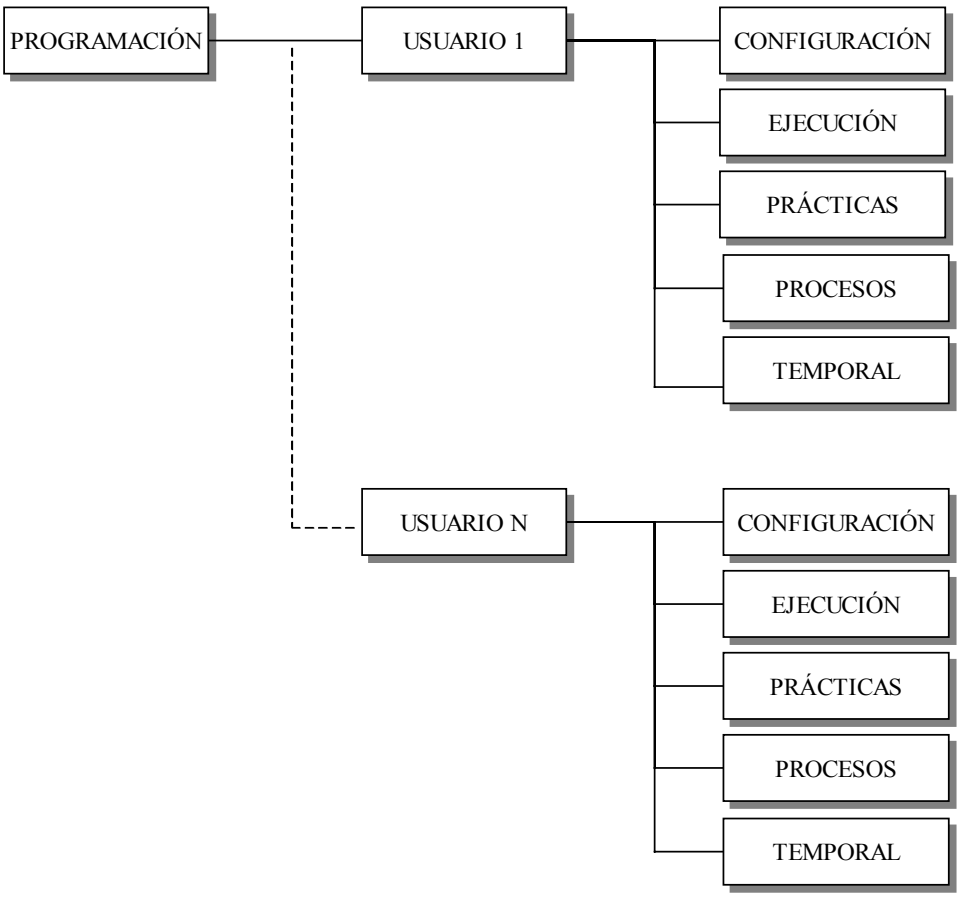


Figura 4.11 Estructura de directorios de los usuarios

Los datos de entrada pueden ser sólo introducidos por el usuario administrador y son:

- ***NIF:*** campo obligatorio a la hora de crear una nueva cuenta de usuario. Este campo debe ser único para cada usuario.
- ***Nombre:*** nombre del usuario.
- ***Apellidos:*** apellidos del usuario.
- ***Teléfono:*** teléfono de contacto del usuario.
- ***E-mail:*** dirección de correo del usuario.
- ***Curso:*** curso al que corresponde el usuario.
- ***Estado:*** dos casos posibles, activo e inactivo. En caso de tener el estado inactivo, el usuario tendrá el acceso denegado a la aplicación.
- ***Usuario:*** es el identificador de la cuenta del usuario. Este campo es obligatorio y único para cada usuario.
- ***Contraseña:*** clave, que junto con el *Usuario*, permiten al usuario acceder a la aplicación, dependiendo siempre del campo *Estado*.

La información de salida de la unidad es:

- Mensaje de verificación en caso de crearse la nueva cuenta.
- Mensaje de error si la cuenta no pudo ser creada.
- En caso de crearse la cuenta, se envía un correo electrónico a la dirección especificada en el campo *E-mail* indicando al usuario su nombre de *Usuario* y *Contraseña* para acceder a la aplicación.

4.4.1.2 Modificar cuentas de usuarios

Los datos introducidos al crear una nueva cuenta de usuario pueden ser modificados a excepción del *NIF*. Estos cambios de información se realizan en dos opciones diferentes de la aplicación dependiendo de los campos a modificar. En una parte, correspondiente a una de las opciones, se realiza la modificación de los campos *Usuario* y *Contraseña*,

campos que sólo afectan a la entrada del usuario en el sistema. En otra opción, se realiza la edición del resto de campos.

Los datos de entrada son, para el primer caso:

- Usuario
- Contraseña

Y para el segundo:

- Nombre
- Apellidos
- Teléfono
- E-mail
- Curso
- Estado

Los datos de salida para ambos casos:

- Mensaje de verificación, en caso de no existir error.
- Mensaje de error, en caso contrario.

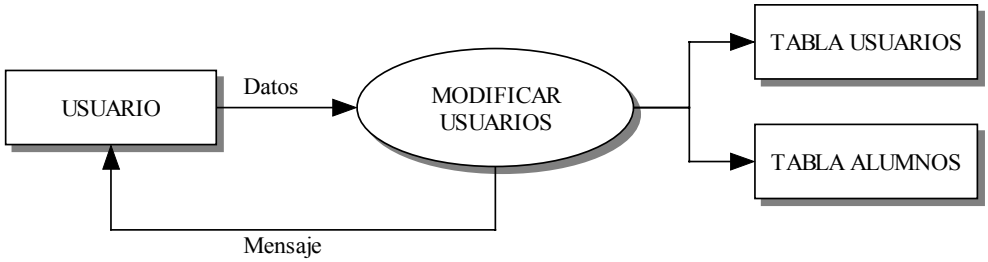


Figura 4.12 Flujo de datos de la unidad funcional Modificar usuarios

4.4.1.3 Eliminar cuentas de usuarios

En esta unidad se pueden eliminar todas las cuentas de usuarios, ya que, en esta parte, sólo se gestionan los usuarios con restricción. Es decir, en el supuesto de eliminarse todas las cuentas presentadas en este módulo, la aplicación queda con el usuario administrador. Por tanto, no es necesario tomar ninguna medida de seguridad para evitar que la aplicación quede sin usuario capaz de gestionarla.

Cada vez que se elimina una cuenta de usuario, se borran todos los datos correspondientes a la misma de la base de datos.

Al mismo tiempo, se elimina la estructura de directorios del usuario creada en el directorio *Programación*. Se borra el directorio raíz especificado por el campo *NIF* del usuario y los cinco subdirectorios, *Configuración*, *Ejecución*, *Prácticas*, *Procesos* y *Temporal*, junto con el contenido de ambos.

El dato de entrada:

- El identificador del usuario, *NIF*, seleccionado del listado de usuarios.

Los datos de salida son:

- Mensaje de verificación si se pudo realizar la eliminación.
- Mensaje de error si la cuenta no pudo ser eliminada.

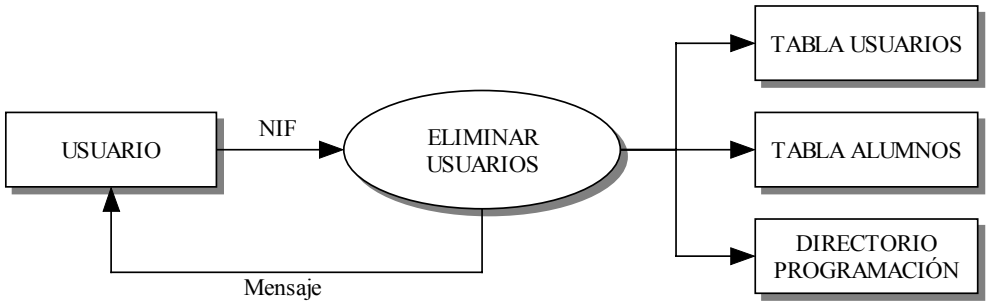


Figura 4.13 Flujo de datos de la unidad funcional Eliminar usuarios

4.4.1.4 Activar/desactivar cuentas de usuarios

El estado de una cuenta de usuario puede tener uno de estos dos valores:

- ***Activo:*** el usuario puede acceder a la aplicación.
- ***Inactivo:*** el usuario tiene el acceso denegado.

La aplicación permite activar cualquier cuenta de usuario que se encuentre inactiva y viceversa.

Para que un usuario pueda acceder a la aplicación, a parte de tener que insertar los datos de acceso correctamente en la entrada, debe tener su cuenta de usuario en estado activo.

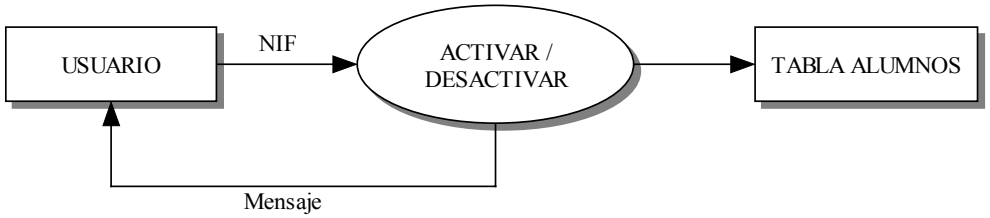


Figura 4.14 Flujo de datos de la unidad funcional Activar/desactivar usuarios

4.4.1.5 Visualizar usuarios

La aplicación debe ser capaz de mostrar un listado de todos los usuarios que pertenecen a ésta, visualizando los datos relacionados a cada uno de ellos, incluyendo el estado en el que se encuentran. En el listado sólo se muestran los usuarios restringidos.

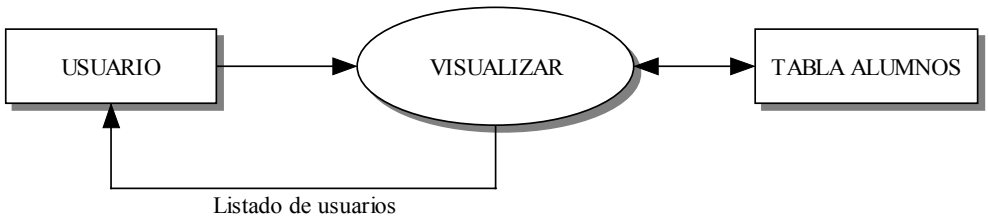


Figura 4.15 Flujo de datos de la unidad funcional Visualizar usuarios

4.4.1.6 Ordenar usuarios

El listado de usuarios mostrado por el sistema, se presenta ordenado alfabéticamente según el campo *Apellidos*. El usuario administrador, único con permiso para acceder a este listado, puede ordenarlo por cualquiera de los siguientes campos, según las necesidades de éste:

- NIF
- Nombre
- Apellidos
- Teléfono
- E-mail
- Curso
- Estado

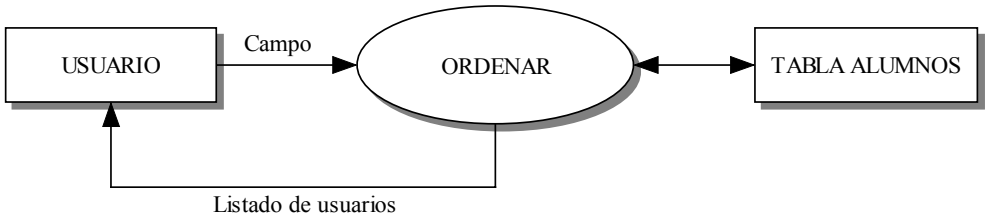


Figura 4.16 Flujo de datos de la unidad funcional Ordenar usuarios

4.4.2 Gestión del usuario administrador

4.4.2.1 Cambiar de usuario administrador

La figura del usuario administrador es imprescindible para el funcionamiento de la aplicación. Por este motivo, su gestión se desarrolla de forma independiente a la del resto de usuarios, de manera que no se permitan realizar las mismas operaciones sobre él. El usuario gestor, ni se crea, ni se elimina, sólo se modifica.

La aplicación siempre arranca con un único administrador y éste sólo puede ser editado con nuevos valores. Si los valores que se editan son los correspondientes al nombre de *Usuario* y *Contraseña*, sólo se modifican los datos de acceso del gestor. En caso contrario, los nuevos valores corresponden a un nuevo usuario, convirtiéndose este nuevo usuario, en el nuevo administrador de la aplicación.

Figura 4.17 Flujo de datos de la unidad funcional Cambiar administrador

Los datos de entrada para el cambio de administrador son:

- Nombre de Usuario del administrador actual
- Contraseña del administrador actual
- NIF del nuevo administrador
- Nombre de Usuario del nuevo administrador
- Contraseña del nuevo administrador

Los datos de salida son:

- Mensaje de verificación en caso de haberse realizado el cambio
- Mensaje de error en caso contrario

4.5. Gestión de directorios

La aplicación permite al usuario explorar las carpetas situadas en el directorio *Prácticas* dentro de su estructura de directorios del sistema de archivos UNIX. La interfaz diseñada proporciona un acceso rápido a las operaciones posibles a realizar sobre los directorios del usuario.

El módulo de *Gestión de directorios* proporciona las siguientes funcionalidades:

- Crear nuevos directorios
- Eliminar directorios

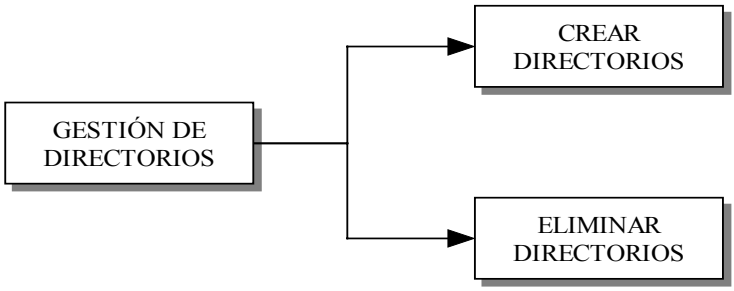


Figura 4.18 Unidades funcionales del módulo de Gestión de directorios

Todos los usuarios pueden acceder al módulo de *Gestión de directorios*. Esta gestión sólo incluye la del directorio *Prácticas* dentro del directorio raíz del usuario. Cada cliente puede crear y eliminar carpetas de manera ilimitada. Al mismo tiempo, todas estas carpetas pueden ser visualizadas desde la interfaz de trabajo de la aplicación.

4.5.1 Crear directorios

Cuando un usuario va a crear un nuevo directorio debe, previamente, seleccionar la ruta donde lo va a crear, siempre dentro del directorio *Prácticas*. Para ello, el usuario puede ir accediendo al contenido de este directorio de manera que tenga conocimiento, antes de la creación de la nueva carpeta, del contenido del mismo.

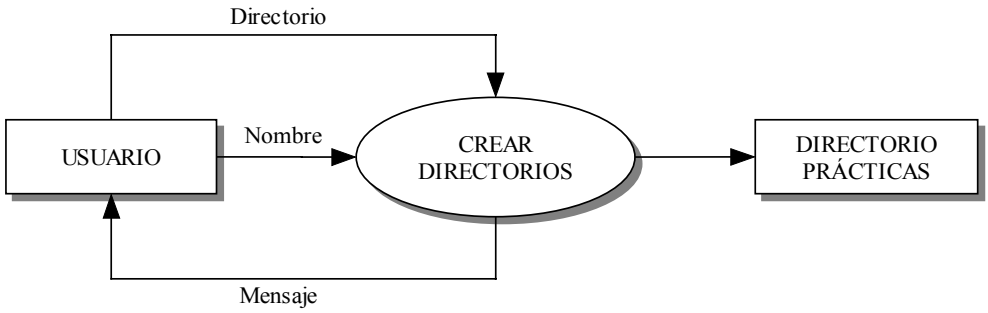


Figura 4.19 Flujo de datos de la unidad funcional Crear directorios

Los datos de entrada que debe especificar el usuario son:

- Directorio donde se va a crear la nueva carpeta.
- Nombre de la nueva carpeta.

En caso de existir algún fallo en la creación del nuevo directorio, se presenta un mensaje de notificación de error como dato de salida.

4.5.2 Eliminar directorios

La aplicación permite al usuario eliminar tantas carpetas como haya creado dentro de su directorio *Prácticas*. En el caso de que el usuario seleccione para eliminar una carpeta que no esté vacía, la aplicación manda a eliminar el directorio y todo su contenido.

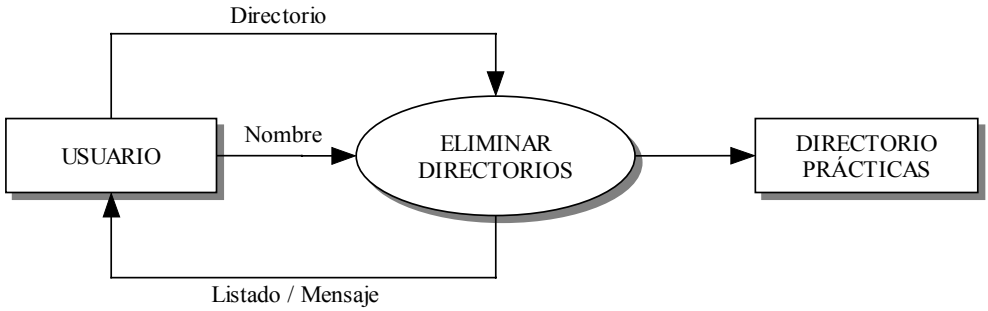


Figura 4.20 Flujo de datos de la unidad funcional Eliminar directorios

Los datos de entrada son:

- Directorio donde se encuentra la carpeta a eliminar.
- Nombre de la carpeta que se quiere eliminar.

Los datos de salida son:

- Mensaje de verificación indicando que el directorio ha sido eliminado.

- Mensaje de error en caso contrario.
- Actualización del directorio *Prácticas* en el interfaz.

4.6 Gestión de archivos

A diferencia del módulo de *Gestión de directorios*, en el módulo de *Gestión de archivos*, el usuario es partícipe de la administración de ficheros en los cinco directorios pertenecientes al directorio raíz del usuario, *Configuración*, *Ejecución*, *Prácticas*, *Procesos* y *Temporal*. Si bien, en el directorio *Prácticas* se le otorga al usuario con la total libertad para la creación, edición y eliminación de ficheros, en los directorios *Configuración*, *Ejecución*, *Procesos* y *Temporal* sólo está implicado de manera indirecta en la gestión de sus archivos, según realice ciertas operaciones en la aplicación. Por tanto, se va a considerar que son estas operaciones las que van a controlar la gestión de los archivos pertenecientes a estos cuatro últimos directorios y no el propio usuario.

Todos los usuarios, tanto el usuario administrador como los usuarios restringidos, tienen acceso al módulo de *Gestión de archivos*.

Existen dos interfaces desde donde el usuario puede acceder a los archivos de su directorio *Prácticas*.

El módulo de *Gestión de archivos* proporciona las siguientes funcionalidades:

- Crear nuevos archivos de texto.
- Editar archivos.
- Eliminar archivos

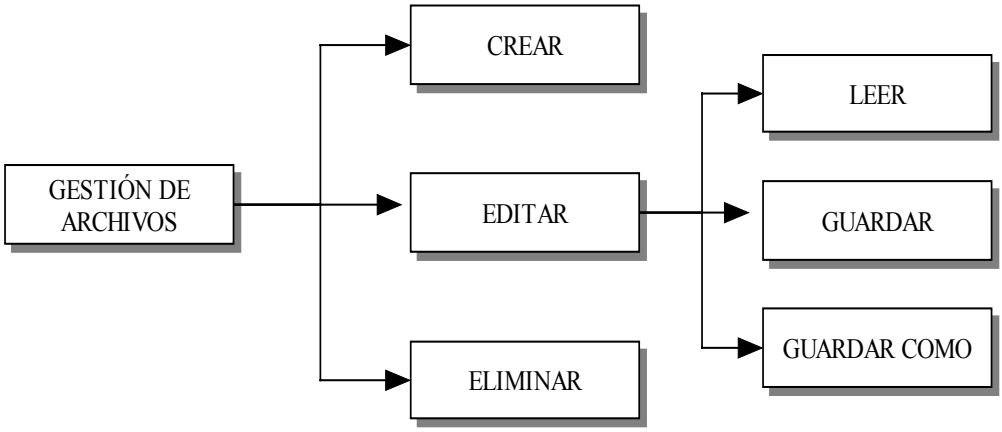


Figura 4.21 Unidades funcionales del módulo de Gestión de archivos

4.6.1 Crear archivos

Un usuario puede crear tantos ficheros como desee dentro de su directorio *Prácticas* perteneciente a su directorio raíz.

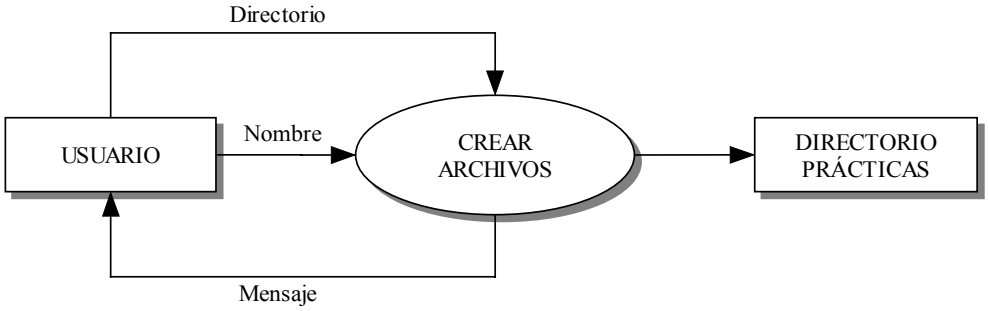


Figura 4.22 Flujo de datos de la unidad funcional Crear archivos

Los datos de entrada son:

- Directorio donde se va a crear el archivo.
- Nombre del archivo.

El dato de salida es:

- Mensaje de error en caso de no poderse crear el nuevo archivo.

4.6.2 Edición de archivos

Consiste en un pequeño editor en el que el usuario puede modificar y mantener la estructura jerárquica de un documento de forma fácil y sencilla, siendo capaz de leer y escribir documentos sobre ficheros de texto. Todo usuario puede trabajar desde el editor con cualquier archivo que se encuentre en su directorio personal *Prácticas*.

El contenido del área de texto puede ser modificado fácilmente por el usuario mediante el teclado.

El módulo de *Edición de archivos* proporciona las siguientes funcionalidades:

- Cargar el contenido de un archivo de texto en el área de texto.
- Guardar el contenido del área de texto dentro del mismo archivo cuyo contenido es el cargado en el área de texto.
- Guardar el contenido del área de texto dentro de un nuevo archivo.

4.6.2.1 Leer archivos

El editor permite cargar el contenido de un archivo de texto perteneciente al usuario en el área de texto de la interfaz de *Edición de archivos*.

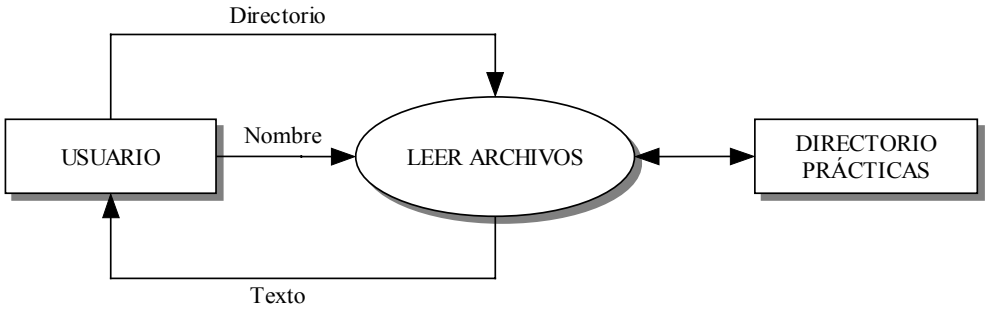


Figura 4.23 Flujo de datos de la unidad funcional Leer archivos

Esta operación la puede realizar el usuario mediante dos formas. Una mediante el árbol jerárquico presentado en la interfaz de trabajo, y otra mediante la opción especifica de *Abrir archivo*. En caso de que el usuario seleccione la primera opción, éste no debe introducir ningún dato de entrada. Los datos de entrada para el segundo caso son:

- Directorio donde se encuentra el archivo.
- Nombre del archivo que se va a leer.

La salida para ambos casos es:

- Texto contenido en el archivo.

4.6.2.2 Guardar

Esta unidad permite al usuario modificar el contenido de cualquier archivo de texto cuyo contenido haya sido cargado previamente en el editor. El contenido del archivo es reemplazado por el contenido del área de texto del editor.

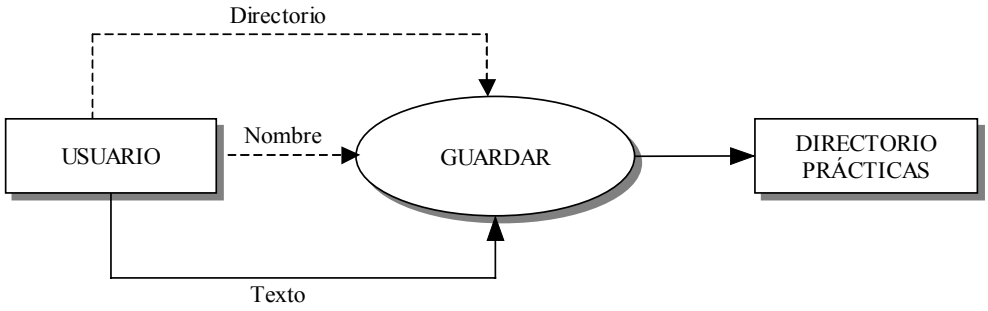


Figura 4.24 Flujo de datos de la unidad funcional Guardar

Los datos de entrada son:

- Directorio donde está ubicado el archivo. Este dato no es introducido explícitamente por el usuario.
- Nombre del archivo. Este dato no es introducido explícitamente por el usuario.
- Texto contenido en el editor.

Esta unidad no presenta dato de salida.

4.6.2.3 Guardar como

Esta unidad permite al usuario crear nuevos ficheros de texto. El proceso permite guardar todo el contenido del área de texto contenido en el campo de edición dentro de un nuevo archivo.

Mediante esta operación se consigue la transferencia de todo el texto introducido por teclado por el usuario a la máquina remota, con lo cual, el usuario realiza la conexión hacia cualquier carpeta perteneciente al usuario dentro de su directorio *Prácticas*.

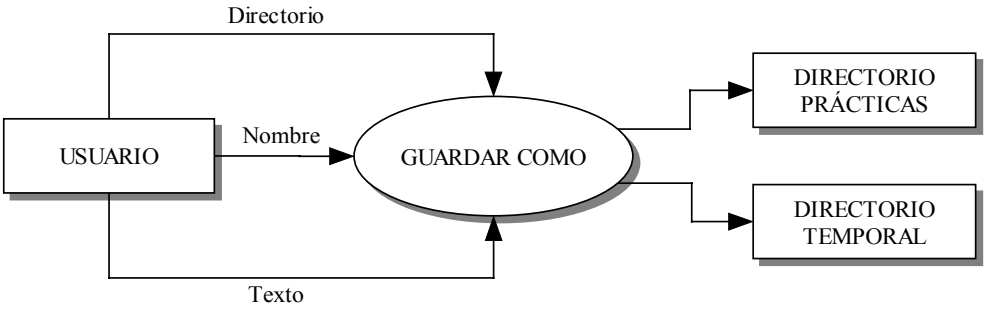


Figura 4.25 Flujo de datos de la unidad funcional Guardar como

Los datos de entrada especificados por el usuario:

- Directorio donde se va a guardar el archivo dentro del directorio *Prácticas*.
- Nombre del archivo que se va a crear.
- Contenido del archivo.

En caso de existir algún error en la operación, se muestra, como dato de salida, un mensaje de notificación.

4.6.3 Eliminar archivos

El usuario puede eliminar cualquier archivo que haya creado dentro de su directorio *Prácticas* y que se encuentre abierto en el área de texto correspondiente a la *Edición de archivos*. Si un archivo tiene asociado un fichero ejecutable, al borrar el archivo, el ejecutable también es eliminado.

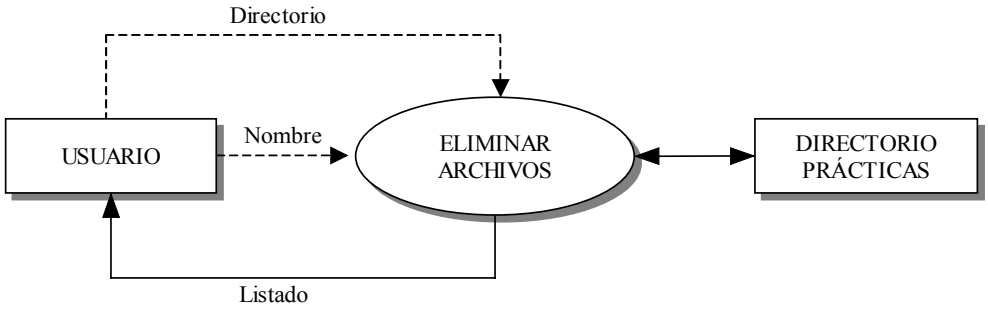


Figura 4.26 Flujo de datos de la unidad funcional Eliminar archivos

Los datos de entrada especificados por el usuario:

- Directorio donde está ubicado el archivo. Este dato no es introducido explícitamente por el usuario.
- Nombre del archivo. Este dato no es introducido explícitamente por el usuario.

Realizada la operación, la aplicación actualiza el directorio de trabajo del usuario mostrando el contenido de esta carpeta sin el fichero eliminado.

4.7 Compilación

Este módulo se encarga de realizar la compilación de cualquier programa. Un programa es un fichero de texto ordinario cuyo contenido corresponde con una colección de instrucciones y de datos. A este contenido se le llama código fuente. El compilador se encarga de traducir el código fuente a código objeto, crea un fichero de salida con esta información y lo marca como ejecutable.

4.7.1 Compilar C/C++

Consiste en una unidad funcional a través de la cual se puede realizar la compilación de cualquier fichero que esté abierto en el área de texto correspondiente al módulo de *Gestión de archivos*. Para que esta unidad realice su funcionamiento óptimo, el archivo abierto debe corresponder con un programa desarrollado bajo el lenguaje C o C++.

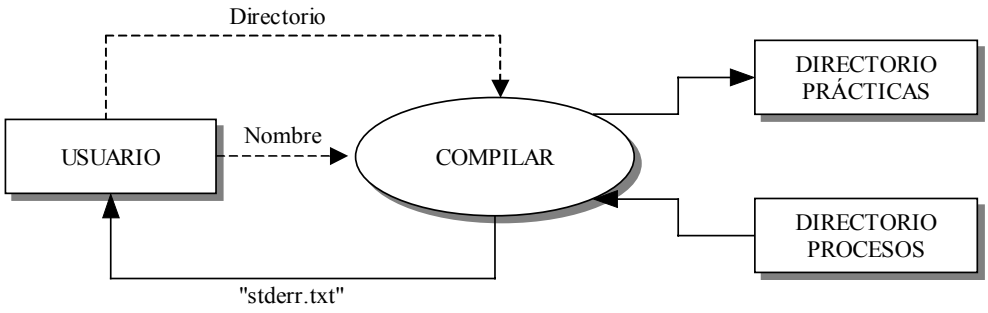


Figura 4.27 Flujo de datos de la unidad funcional Compilar C/C++

Todos los usuarios tienen acceso al módulo de *Compilación*.

Los datos de entrada son:

- Directorio de ubicación del archivo a compilar. Este dato es introducido implícitamente por el usuario.
- Nombre del archivo a compilar. Este dato es introducido implícitamente por el usuario.

El dato de salida es:

- Contenido del fichero *stderr.txt* correspondiente al directorio *Procesos* dentro de la carpeta personal del usuario. Dentro de este fichero se almacena información sobre el resultado de la compilación indicando si se ha producido error o no.

4.8 Ejecución

Este módulo realiza la operación de ejecutar los archivos "ejecutables" resultantes del módulo de *Compilación*.

4.8.1 Ejecutar C/C++

Todos los usuarios tienen acceso al módulo de *Ejecución*. Los usuarios solamente pueden ejecutar los archivos que se encuentren en su directorio *Prácticas* dentro de su directorio raíz y que tengan asociado el fichero ejecutable creado en el módulo de *Compilación*.

Para ejecutar un archivo, el fichero debe estar abierto en el área de texto perteneciente al módulo de *Gestión de archivos*.

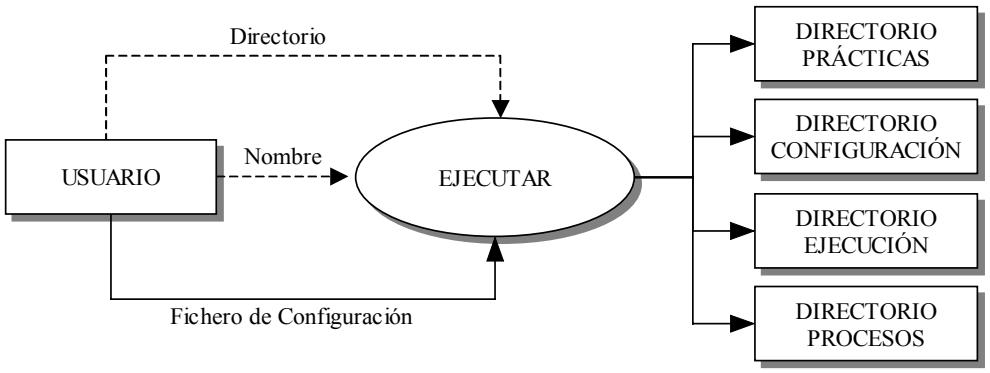


Figura 4.28 Flujo de datos de la unidad funcional Ejecutar C/C++

En esta unidad funcional existe una gran variedad de datos de entrada, los cuales son:

- Directorio de ubicación del archivo a ejecutar. Este dato no es introducido explícitamente por el usuario.

- Nombre del archivo a ejecutar. Este dato no es introducido explícitamente por el usuario.
- Fichero de configuración. En este fichero se incluyen:
    - Parámetros: valores de entrada que necesita el archivo a ejecutar. Se trata de una lista de parámetros separados por un espacio en blanco.
    - Fichero de entrada: en caso de que el archivo a ejecutar lea de un fichero los datos de entrada que necesita en su ejecución, se debe especificar el nombre y ruta de este archivo.
    - Fichero de salida: nombre del fichero donde se almacena el resultado de la ejecución.
    - Timeout: tiempo, especificado en segundos, que está el archivo en ejecución. Transcurrido el timeout, si la ejecución no ha finalizado, el sistema lo fuerza a la finalización.

4.9 Gestión de procesos

Cuando un programa es leído del disco por el núcleo y es cargado en memoria para ejecutarse, se convierte en Proceso. En un proceso, no sólo hay una copia del programa, sino que además, el núcleo le añade información adicional para poder manejarlo.

En este módulo se muestra el listado de procesos que tiene el usuario y parte de la información que genera el núcleo sobre ellos. La aplicación almacena esta información en el fichero *ejecucion.txt* dentro del directorio *Procesos* perteneciente al directorio raíz de cada usuario. Además, este módulo permite al propio usuario a finalizar aquellos procesos vivos que tenga.

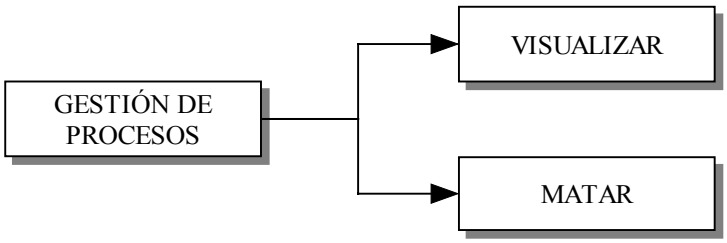


Figura 4.29 Unidades funcionales del módulo de Gestión de procesos

Todos los usuarios tienen acceso a este módulo de *Gestión de procesos*. Los usuarios sólo pueden gestionar sus propios procesos, a excepción del usuario administrador, el cual puede gestionar los procesos de todos los usuarios de la aplicación.

4.9.1 Visualizar procesos

En esta unidad se muestra toda la información almacenada en el fichero *ejecucion.txt*. El usuario visualiza un listado de todos los procesos que ha generado en el sistema e información relacionada con ellos. Esta información es:

- Ubicación del proceso
- Nombre del proceso
- Estado del proceso. Los caso posibles que puede tomar son:
    - Timeout: si el proceso finalizó debido a este parámetro.
    - En ejecución: si aún continua vivo.
    - Normal: en caso de no ocurrir los dos anteriores.
- Finalizado / Matar:
    - Finalizado: el proceso ha finalizado.
    - Matar: enlace a finalizar el proceso.

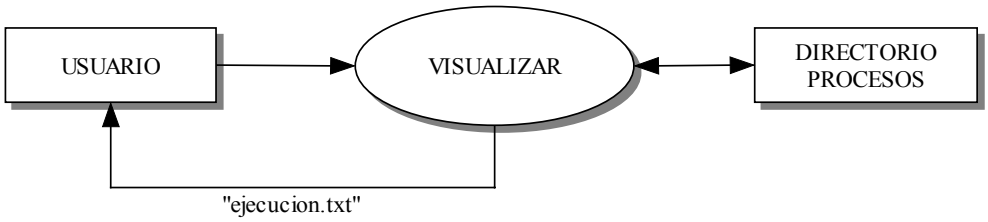


Figura 4.30 Flujo de datos de la unidad funcional Visualizar procesos

Esta unidad funcional no tiene dato de entrada. El dato de salida es el contenido del fichero *ejecucion.txt.*

4.9.2 Matar procesos

Manda a ejecutar la finalización de un proceso que esté vivo. El resultado es la muerte del proceso y modificación del contenido del fichero *ejecucion.txt* para marcar al proceso mandado a matar como finalizado.

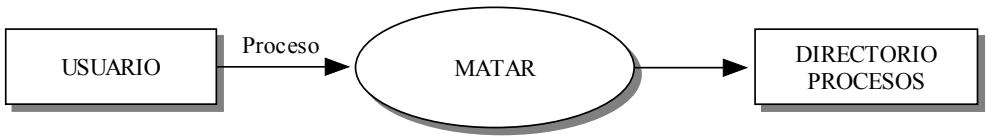


Figura 4.31 Flujo de datos de la unidad funcional Matar procesos

Esta unidad no requiere de ningún dato de entrada puesto que existe por cada proceso vivo un enlace a ella. Como dato de salida sólo está la modificación del fichero *ejecucion.txt.*

5 Análisis orgánico

5.1 Introducción

En este capítulo, y siguiendo la estructura realizada en el *Análisis funcional*, se explica la estructura interna de la Base de Datos y se profundiza en el diseño de la herramienta, detallando las funciones de cada uno de los scripts que se han desarrollado para este Proyecto y las relaciones existentes entre ellos.

5.2 Modelos de datos

Es necesario resaltar que la funcionalidad de este Proyecto no se fundamenta en un sistema de Base de Datos. No obstante, su utilización es vital para el funcionamiento de la aplicación, aunque su carácter principal no esté basado en ella.

Se ha desarrollado una Base de Datos común para toda la aplicación, cuyo diagrama Entidad–Relación se describe a continuación.

5.2.1 Diagrama Entidad – Relación

La figura 5.1 representa el diagrama Entidad-Relación de los datos que serán manejados por la aplicación. Para ello, se han definido las siguientes entidades:

- ***Usuarios:*** Representa el conjunto de usuarios que acceden a la aplicación.

- ***Alumnos:*** Representa sólo a los usuarios sin privilegios de administración.

A través de la entidad *Usuarios* se entra en la aplicación. Sólo un elemento de la entidad *Usuarios* puede acceder a la información contenida en la entidad *Alumnos* y sólo un elemento de la entidad *Usuarios* no es contenido en la entidad *Alumnos*, el administrador de la aplicación.

Figura 5.1 Diagrama Entidad-Relación de la aplicación

5.2.2 Diagramas Entidad – Atributo

Una vez definidas las entidades que entran en juego en el modelo, pasamos a describir mediante los diagramas Entidad-Atributo, los atributos de cada una de las entidades definidas.

Los atributos de la entidad *Usuarios* se representan en la figura 5.2. Esta entidad guardará los valores de conexión de cada usuario registrado en el sistema. Esta combinación de valores será única y estará formada por el atributo *login* y el atributo *pass*. Además, distinguirá el tipo de usuario, es decir, si éste tiene privilegios de administrador para acceder al bloque de administración.

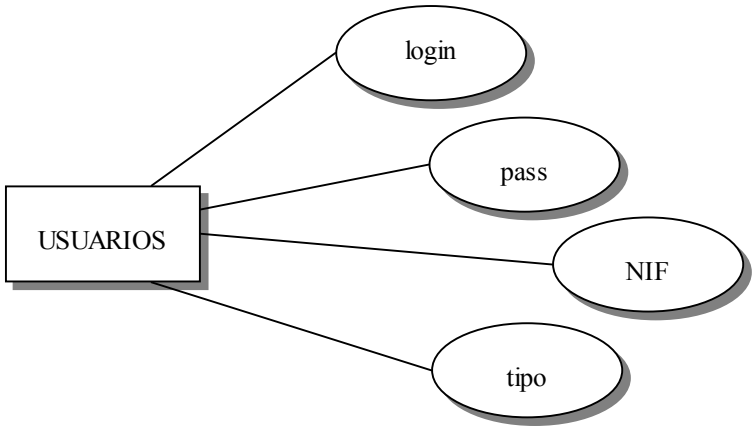


Figura 5.2 Diagrama Entidad-Atributo. (Entidad Usuarios)

Los atributos de la entidad *Alumnos* se representan en la figura 5.3. En ella se almacena toda la información personal de cada usuario, a excepción del usuario administrador de la aplicación. La cuenta de cada alumno puede estar activada o bloqueada. Si la cuenta está activada, el alumno puede acceder a la aplicación, en caso contrario, su acceso será denegado.

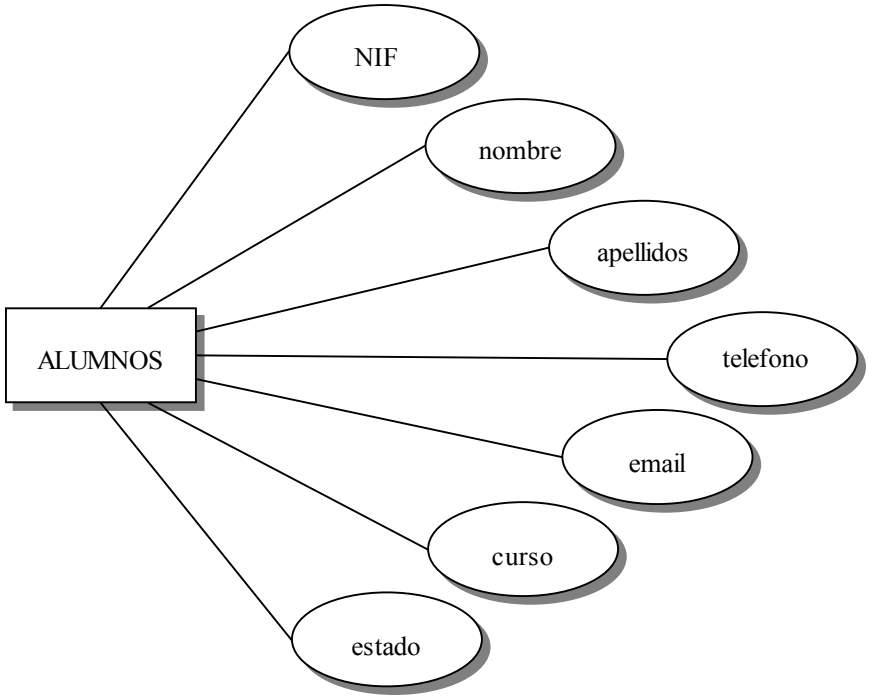


Figura 5.3 Diagrama Entidad-Atributo. (Entidad Alumnos)

5.2.3 Relación

A partir de la relación binaria del diagrama Entidad-Relación, véase figura 5.1, y teniendo en cuenta el grado y la obligatoriedad de la correspondencia, podemos convertir esta relación en tablas del modelo relacional. La relación que se identifica en el diagrama es la siguiente:

Relación "Administrador": relación entre la entidad *Usuarios* con cardinalidad 1 y la entidad *Alumnos* con cardinalidad 1, lo que significa que un usuario, si éste no es administrador de la aplicación, obligatoriamente tiene unos datos personales registrados en el sistema y que estos datos únicamente pertenecen a un sólo usuario. Este tipo de relación, entidad con cardinalidad 1 y entidad con cardinalidad 1, podría suprimirse si los atributos de cada una de ellas fuesen almacenados en una sola entidad.

5.2.4 Modelo relacional

En función del modelo Entidad-Relación se han definido las siguientes tablas:

- Usuarios
- Alumnos

Los campos de cada una de las tablas se describen en los apartados sucesivos.

5.2.4.1 Tabla Usuarios

La tabla *Usuarios* almacena los valores de conexión y el tipo de usuario de todos los usuarios registrados en el sistema. En la figura 5.4 se detallan los distintos campos de la tabla.

Tabla: Usuarios		
Nombre	**Tipo**	**Descripción**
login	Varchar(16) No Null	Clave primaria. Clave de acceso del usuario.
pass	Varchar(16) No Null	Contraseña del usuario.
NIF	Varchar(9) No Null	Identificador del usuario.
tipo	Varchar(13) No Null	Indica si el usuario es administrador o no.

Figura 5.4 Descripción de la tabla Usuarios

5.2.4.2 Tabla Alumnos

La tabla *Alumnos* almacena los datos personales de todos los usuarios del sistema, a excepción del administrador de la aplicación. En la figura 5.5 se detallan los campos de la tabla *Alumnos*.

Tabla: Alumnos		
Nombre	**Tipo**	**Descripción**
NIF	Varchar(9) No Null	Clave primaria. Identificador del usuario.
nombre	Varchar(32)	Nombre del usuario.
apellidos	Varchar(56)	Apellidos del usuario.
telefono	Varchar(24)	Teléfono del usuario.
email	Varchar(128)	Dirección de correo del usuario.
curso	Varchar(7)	Curso correspondiente del usuario.
estado	Varchar(8)	Estado de la cuenta del usuario.

Figura 5.5 Descripción de la tabla Alumnos

5.3 Diagramas de ejecución de scripts

En los siguientes apartados se describe el desarrollo de la aplicación mediante módulos que incluyen los diagramas que representan la invocación y ejecución de los

diferentes scripts que componen la aplicación, indicando además, en que puntos se accede a la Base de Datos.

En las figuras 5.6 y 5.7 se representan los iconos utilizados y sus significados.

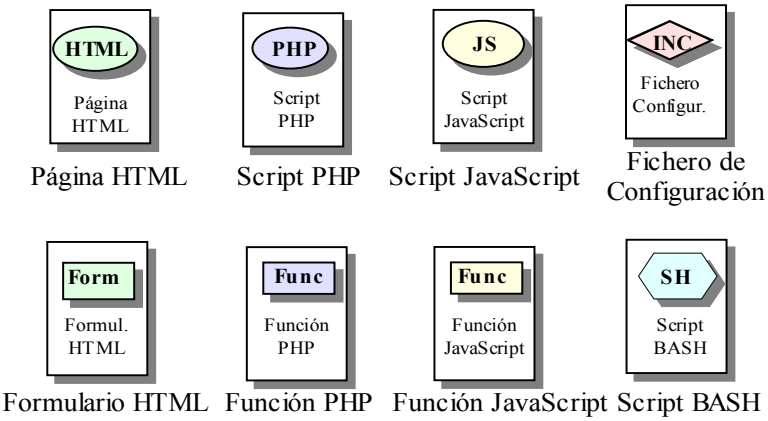


Figura 5.6 Elementos de los Diagramas de Relaciones I (D.R.)

Figura 5.7 Elementos de los Diagramas de Relaciones II (D.R.)

Existen tres archivos de inclusión pertenecientes a la aplicación que se caracterizan porque sus llamadas son continuas debido a la importancia de la información que en ellos se almacena. Por este motivo, no se especificarán sus llamadas en los módulos, pero sí se especificarán en los diagramas que los representan. Estos tres archivos son:

- ***BaseDatos.inc:*** almacena información con respecto a la Base de Datos MySQL. Estos valores corresponden con el nombre del servidor de Base de Datos MySQL, nombre de usuario y contraseña del usuario MySQL, y por último, el nombre de la Base de Datos a la que accederá la aplicación.

- ***Directorios.inc:*** almacena información referente a las ubicaciones de los diferentes scripts utilizados en la aplicación y del directorio principal de donde colgarán las carpetas personales de los usuarios dados de alta.
- ***Titulos.inc:*** almacena información de los títulos presentados en las páginas web solicitadas. Existen dos títulos para la aplicación, uno perteneciente al bloque de trabajo y otro para el bloque de administración.

A continuación se describen cada uno de los módulos.

5.3.1 Módulo de autentificación de usuario

El "Módulo de autentificación de usuario" es la primera presentación que ofrece la aplicación y actúa como si de un filtro se tratara. En él, se solicita al cliente que introduzca unos valores mediante los cuales, se comprueba en el servidor si el usuario tiene autorización para acceder a los diferentes bloques de la aplicación.

Este módulo corresponde con el script *"index.php"*. El script, ejecutado en el servidor, muestra en el cliente un formulario con dos campos para rellenar y dos botones habilitados. Una vez pulsado el botón de conformidad, el script realiza la conexión a la Base de Datos MySQL y pasa a comprobar que los valores introducidos corresponden con los de algún usuario dado de alta en la aplicación, comprobando que se encuentran en la tabla *usuarios*.

Si dichos valores no se hallan en la tabla, el script vuelve a mostrar el mismo formulario. En caso contrario, el script registra al usuario en la sesión y pasa a averiguar si éste tiene privilegios de administrador o no, extrayendo la información del campo *tipo*, correspondiente al cliente registrado, de la tabla *usuarios*. En el supuesto de tener el usuario privilegios de gestor, el script llama a *"FramesAdministrador.php"*. Si por el contrario, el usuario no es administrador, se comprueba que no tiene el acceso denegado y se pasa a cargar el script *"FramesAlumno.php"*. Para ello, el script *"index.php"* lee el campo *estado* de la tabla *alumnos*, correspondiente al usuario registrado en la sesión.

El diagrama de relaciones de este módulo se presenta en la figura 5.8.

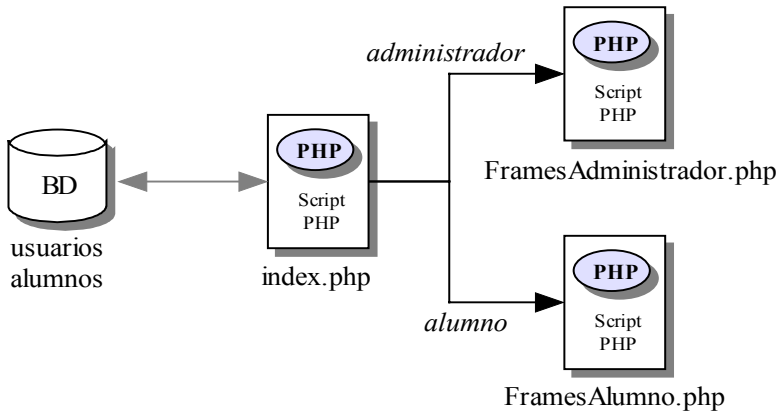


Figura 5.8 D.R. del Módulo de Autentificación

El "Módulo de autentificación de usuario" se encarga de dotar seguridad a toda la aplicación de acceso restringido. La técnica utilizada es comprobar, al principio de todas las páginas, que el usuario que solicita el servicio ha sido previamente autorizado, verificando su registro en la sesión. En caso de no estar el usuario validado, las páginas no son ejecutadas.

5.3.2 Bloque de administración

Este bloque proporciona el módulo de gestión de toda la aplicación y sólo es accesible desde el script *"BanerAdministrador.php"*, el cual pertenece al "Bloque de trabajo" que se detalla en el apartado 5.3.5 de la presente memoria. El "Bloque de administración" corresponde con el script *"FramesAdministracion.php"* y devuelve al cliente una página con cuatro marcos: *topFrame*, *linkFrame*, *mainFrame* y *downFrame*, situados respectivamente en la parte superior, izquierda, derecha superior y derecha inferior de la página.

El diagrama de relaciones de petición de los marcos se muestra en la figura 5.9.

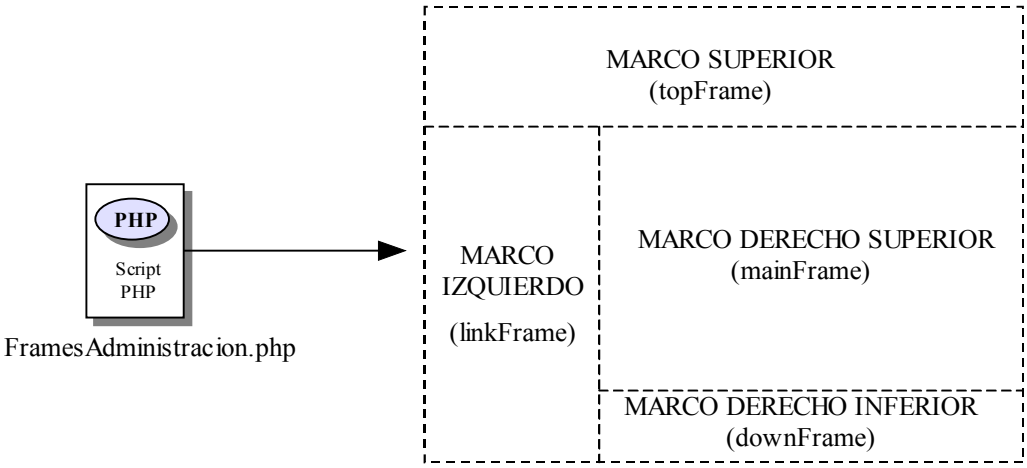


Figura 5.9 D.R. del Bloque de Administración. Petición de marcos

De igual forma, los marcos solicitan la ejecución de nuevos scripts. El marco *topFrame* llama a ejecutar el script *"BanerAdministracion.php"*. Este script realiza nuevas peticiones al servidor de nuevos scripts y también contiene funciones JavaScript.

El marco *linkFrame* realiza la petición al servidor de la ejecución del script *"EnlacesAdministracion.php"*. Este script presenta una serie de opciones que conforman todas las acciones propias del administrador de la aplicación. Estas opciones solicitan a su vez, la ejecución de otros scripts.

El marco *mainFrame* ordena, por defecto, la ejecución del script *"Listado.php"*. Este script se describe en el apartado 5.3.3.1.

Por último, el marco *downFrame* solicita la ejecución del script *"Firma.php"*, el cual muestra únicamente un vínculo en el cliente. Este enlace abre una nueva ventana de navegación.

En la figura 5.10 se muestra el diagrama de relaciones con la construcción de los cuatro marcos.

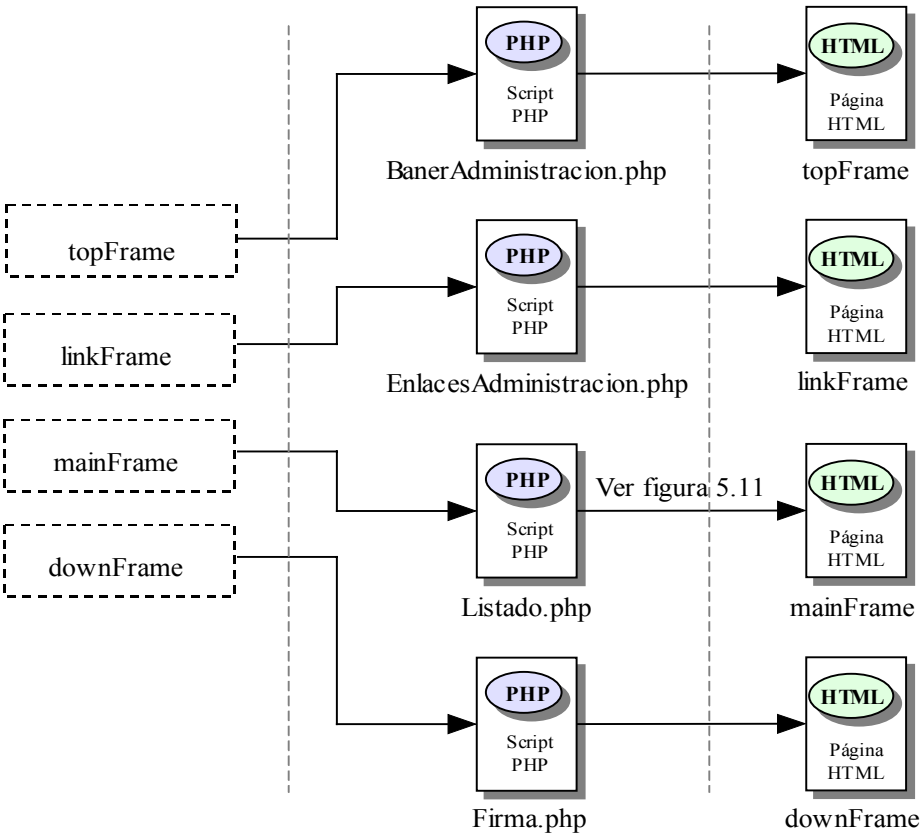


Figura 5.10 D.R. del Bloque de Administración. Construcción de marcos

5.3.3 Gestión de usuarios

Esta área se encarga de la administración de los usuarios de la aplicación. Incluye la inserción de nuevos alumnos, sus modificaciones y eliminaciones. Todas estas opciones son accesibles únicamente por el administrador de la aplicación y se presentan en forma de vínculos dentro del marco *linkFrame*.

5.3.3.1 Módulo de visualización de usuarios

Este módulo ofrece información sobre todos los usuarios dados de alta en el sistema, a excepción del administrador de la aplicación.

En el cliente, el script *"EnlacesAdministracion.php"* presenta varios vínculos. Uno de estos vínculos pide la ejecución en el servidor del script *"Listado.php"* y como consecuencia, se carga una nueva página en el marco *mainFrame*. Este script realiza una consulta a la tabla *alumnos* en el servidor de Base de Datos MySQL, y obtiene los datos de los alumnos pertenecientes a la aplicación. Esta consulta recibe un parámetro llamado *orden*, que coincide con el atributo por el que realizar la ordenación, y se efectúa gracias a la inclusión del fichero *"BaseDatos.inc"* y mediante una llamada a la función *ConsultarBD()*. Esta función se encuentra en el script *"conexion.php"*, el cual es incluido automáticamente por *"Listado.php"*. A su vez, el script *"Listado.php"* presenta una serie de vínculos que vuelven a mandar la ejecución de sí mismo. Cada uno de estos vínculos carga al parámetro *orden* con distintos valores.

En la figura 5.11 se muestra el diagrama de relaciones.

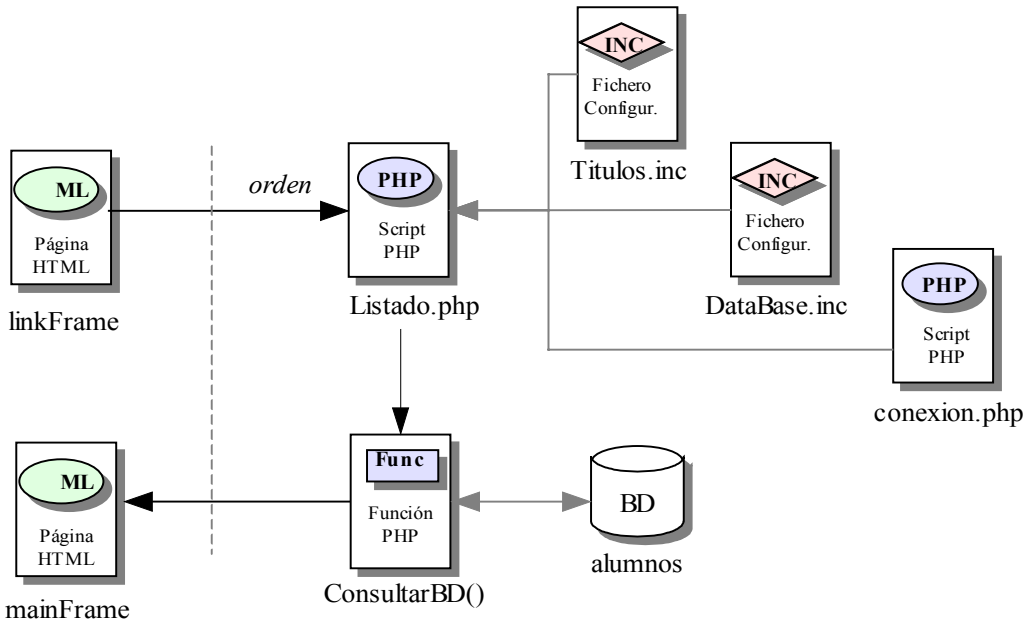


Figura 5.11 D.R. del Módulo de Visualización de Usuarios

5.3.3.2 Módulo de crear cuentas de usuarios

En este módulo se realiza el proceso de añadir nuevas entradas en las tablas *usuarios* y *alumnos*. La inserción de un nuevo registro se realiza en cuatro pasos:

1.- Carga del formulario de inserción.

2.- Envío del formulario con los datos a insertar.

3.- Comprobación de los datos introducidos en el formulario.

4.- Inserción del nuevo registro en la Base de Datos.

Para la carga del formulario, el script *"EnlacesAdministracion.php"* presenta una página, situada en el marco *linkFrame*, que contiene un vínculo que realiza la petición al servidor para que se ejecute el script *"Nuevo1.php"*. Este script envía el formulario de inserción al cliente en una nueva página situada en el marco *mainFrame*. En la figura 5.12 se muestra el diagrama de relaciones de la carga del formulario de inserción mencionado.

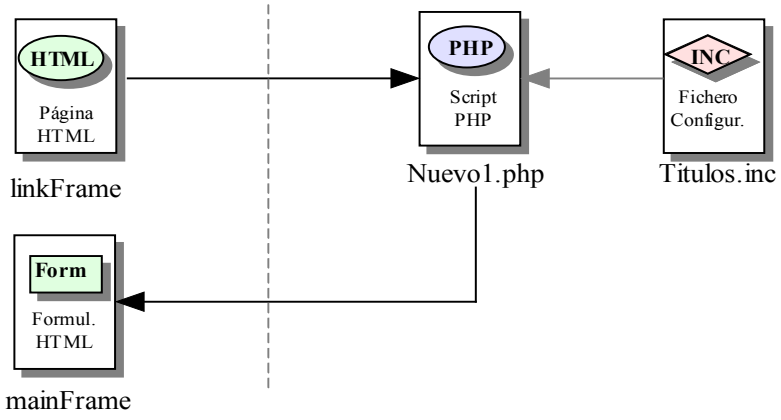


Figura 5.12 D.R. del Módulo de Crear Cuentas. Creación del formulario

Antes de enviar los datos del nuevo usuario, el script llama a la función JavaScript *Verificar()*. Esta función comprueba que los campos obligatorios del formulario han sido rellenados. Una vez comprobado esto, el script ordena la ejecución en el servidor de un nuevo script llamado *"Nuevo2.php"*.

"Nuevo2.php" incluye otro script, *"conexion.php"*, para posteriormente realizar, con la llamada a la función *ConsultarBD()*, una consulta al servidor de Base de Datos MySQL. Mediante esta consulta se comprueba que no existe, en las tablas *usuarios* y *alumnos*, ningún usuario con los mismos datos identificativos del nuevo usuario a insertar.

En caso de existir, es decir, si en las tablas se encuentran ciertos valores iguales a los enviados en el formulario, se carga la página *"Nuevo3.html"* en el cliente, informando que ya existe un usuario con los mismos valores identificativos. Esta página contiene un vínculo que realiza una llamada a la función JavaScript *Volver()*, la cual ordena la ejecución en el servidor del script *"Listado.php"*. En caso de no haber ningún usuario registrado en la Base de Datos con los mismos datos recibidos en el formulario, se insertan estos datos del nuevo usuario, en las tablas *usuarios* y *alumnos* y se crea una estructura de directorios para el nuevo alumno, dentro del directorio de usuarios especificado en el archivo de configuración *"Directorios.inc"*. Al mismo tiempo, el script envía al nuevo usuario un correo electrónico. Realizados estos pasos, el script pasa a cargar una nueva página en el cliente, *"Nuevo2.html"*.

La página *"Nuevo2.html"* muestra en el cliente los datos del nuevo usuario que fueron insertados en las tablas y ofrece dos botones de selección. Ambos botones realizan una llamada a la función JavaScript *Volver()*. Según cuál de los dos botones se seleccione, la función mandará a ejecutar en el servidor el script *"Nuevo1.php"* o *"Listado.php"*.

La figura 5.13 muestra el diagrama de relaciones.

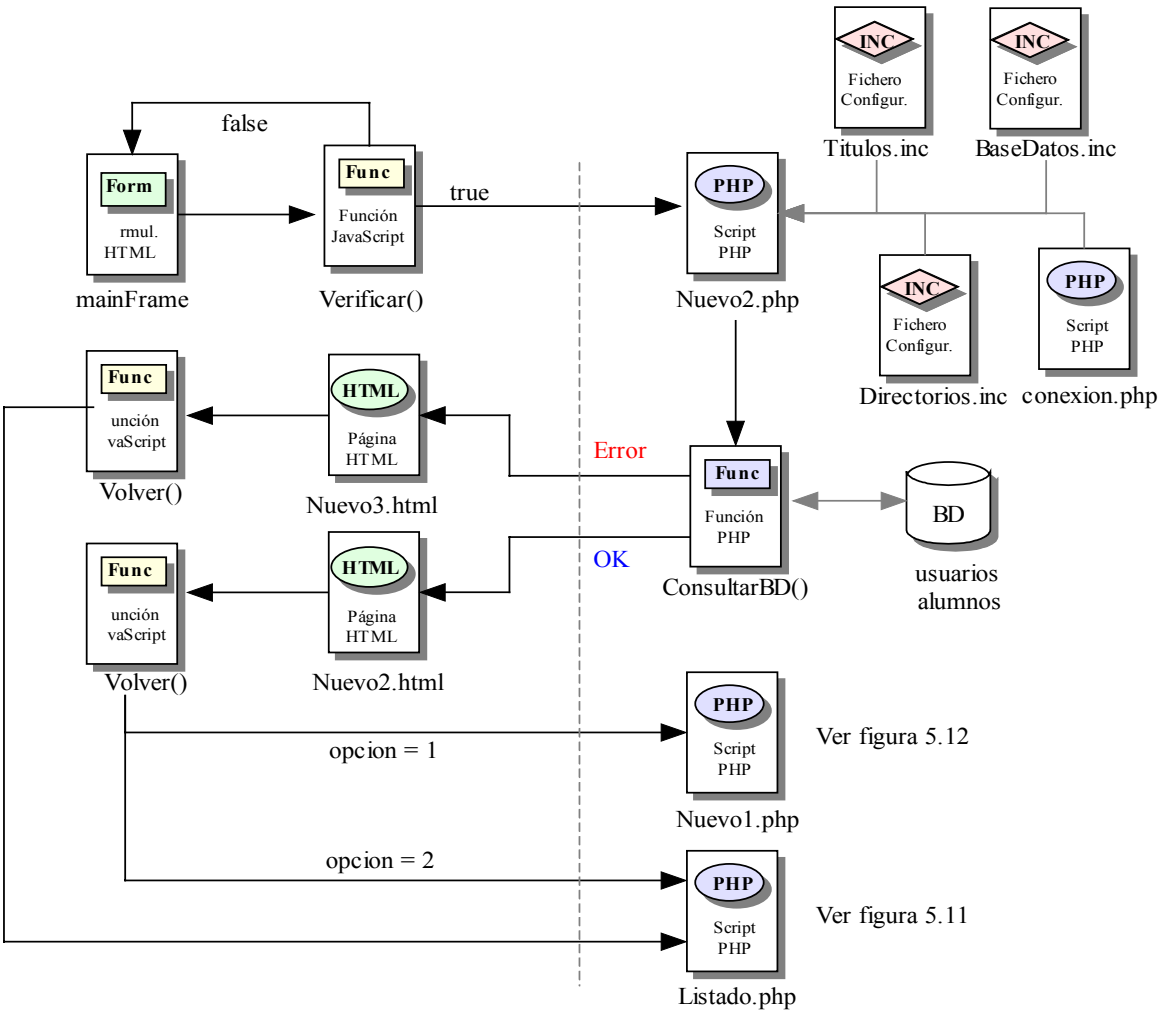


Figura 5.13 D.R. del Módulo de Crear Cuentas de Usuarios

5.3.3.3 Módulo de modificar cuentas de usuarios

En este módulo se realiza el proceso de modificación de la tabla *alumnos* de la Base de Datos MySQL de la aplicación. La modificación de los registros se realiza en cuatro pasos:

1.- Carga del formulario de edición con los datos del registro a modificar.

2.- Envío del formulario con los datos modificados.

3.- Comprobación de los datos introducidos en el formulario.

4.- Modificación del registro en la Base de Datos.

Para la carga del formulario, al igual que en el módulo anterior, el script *"EnlacesAdministracion.php"* presenta una página situada en el marco *linkFrame*, que contiene un vínculo que realiza la petición al servidor para que se ejecute el script *"Modificar1.php"*.

Este script incluye otro script, *"conexion.php"*, que permite realizar la llamada a la función *ConsultarBD()* para realizar una consulta en el servidor de Base de Datos MySQL. Con el resultado de esta consulta, el script *"Modificar1.php"* devuelve al cliente una página en el marco *mainFrame*, donde se muestra un listado con todos los alumnos de la aplicación. En ella, también se muestra una serie de vínculos los cuales vuelven a mandar a ejecutar al propio script *"Modificar1.php"*.

En esta página, el cliente debe seleccionar al alumno cuyos datos personales desea modificar. Una vez hecho esto, se manda a ejecutar en el servidor un nuevo script llamado *"Modificar2.php"*, que recibe por parámetro el identificador del alumno seleccionado.

"Modificar2.php" vuelve a incluir la ejecución del script *"conexion.php"* y, mediante la llamada a la función *ConsultarBD()*, realiza la búsqueda de todos los datos personales en la tabla *alumnos*, del alumno seleccionado, gracias al dato recibido del script *"Modificar1.php"*. Realizada la consulta, el script devuelve al cliente una página con un formulario de edición cuyos valores por defecto son los obtenidos de dicha consulta.

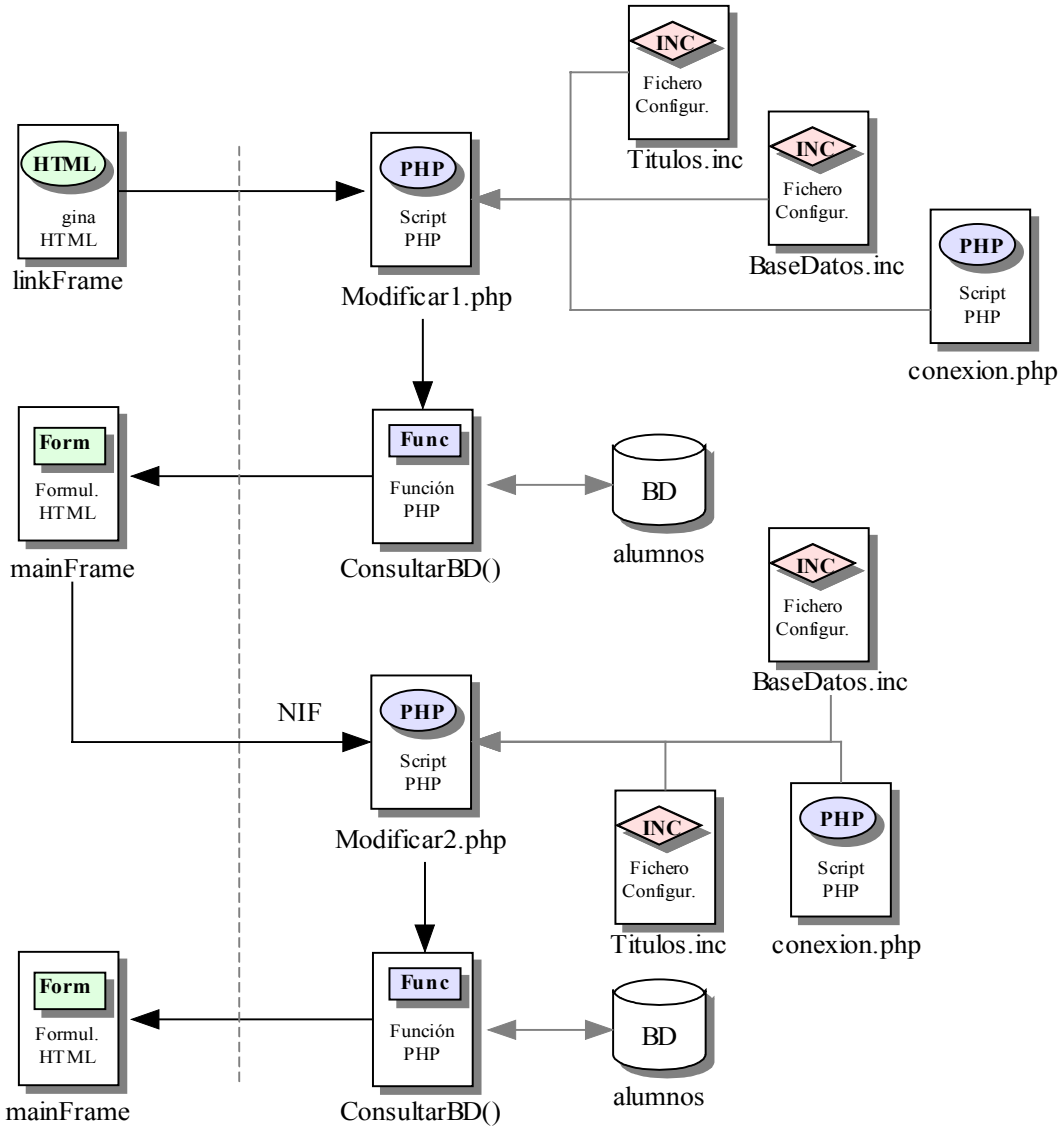


Figura 5.14 D.R. del Módulo de Modificar Cuentas. Creación del formulario

Este formulario de edición envía los datos modificados al script *"Modificar3.php"*, el cual vuelve a incluir la ejecución del script *"conexion.php"*, para esta vez modificar los datos en el registro de la tabla *alumnos*. Terminada la operación, el script manda a cargar en el cliente la página *"Modificar3.html"*, que contiene la función JavaScript *Volver()*, la cual manda a ejecutar en el servidor el script *"Listado.php"*.

En la figura 5.15 se muestra el diagrama de relaciones correspondiente al "Módulo de modificar cuentas de usuarios".

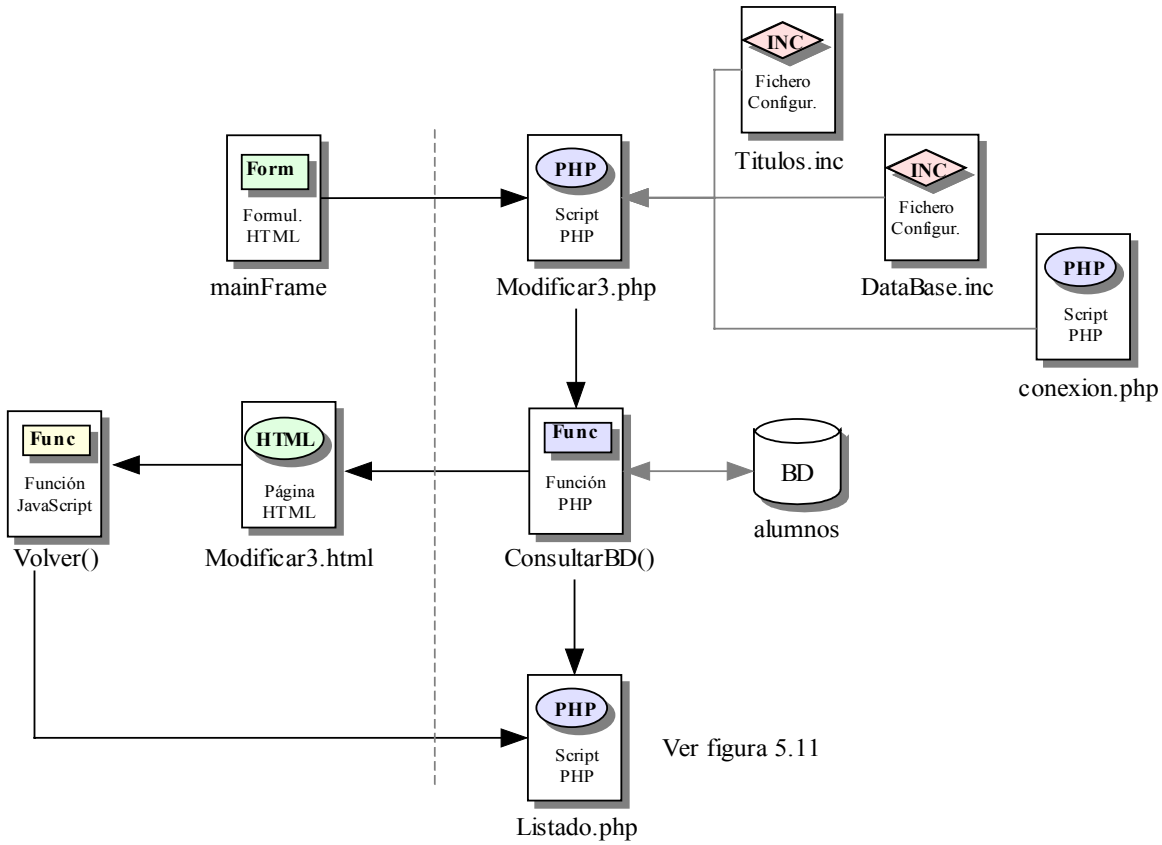


Figura 5.15 D.R. del Módulo de Modificar Cuentas de Usuarios

5.3.3.4 Módulo de eliminar cuentas de usuarios

Este módulo realiza la eliminación de las cuentas de los usuarios de la aplicación. En la figura 5.17 se muestra el diagrama de relaciones correspondiente a este módulo que se detalla a continuación.

En el cliente, el marco *linkFrame* presenta un vínculo que ordena la ejecución en el servidor del script *"Eliminar1.php"* y éste, a su vez, incluye la ejecución del script *"conexion.php"*. Se crea entonces una consulta, la cual es pasada por parámetro a la función *ConsultarBD()*. Esta función junto con la consulta recibida, lee todos los registros de la tabla *alumnos* y muestra dicho listado al cliente en una nueva página situada en el marco *mainFrame*. En ella, el usuario debe seleccionar el alumno, o los alumnos, que desea eliminar a través de un formulario. El diagrama de relaciones para la creación de dicho formulario se representa en la figura 5.16.

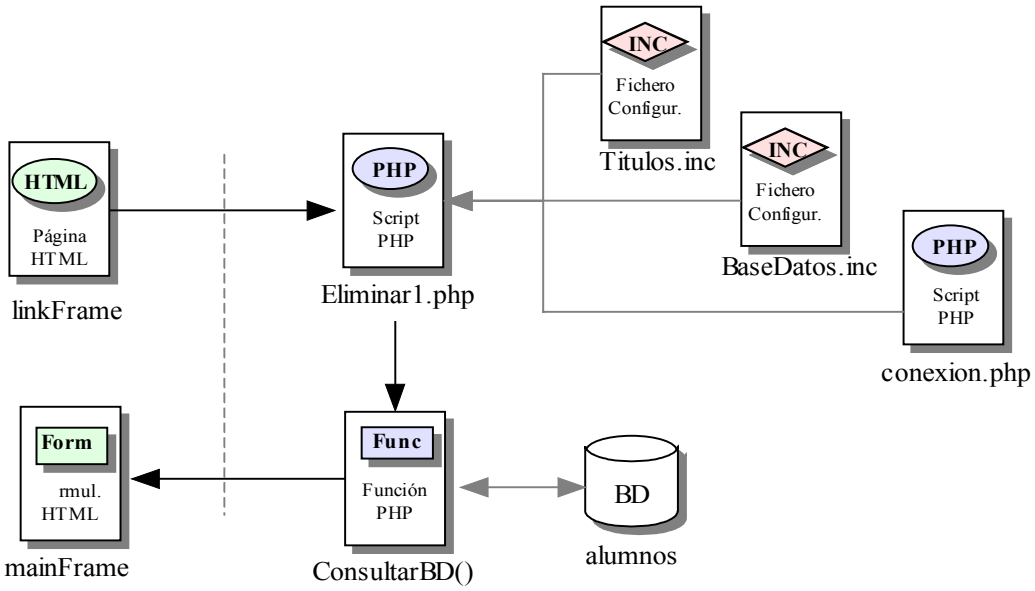


Figura 5.16 D.R. del Módulo de Eliminar Cuentas. Creación del formulario

Realizada la selección, se llama a la función JavaScript *Confirmación()*, la cual muestra un mensaje donde el cliente debe corroborar la eliminación, o por el contrario, cancelar la operación. En caso de abortarla, se vuelve a ordenar la ejecución del script *"Eliminar1.php"*. En caso contrario, se solicita la ejecución en el servidor del script *"Eliminar2.php"*, recibiendo como parámetro el identificador del alumno, o de los alumnos seleccionados en el script *"Eliminar1.php"*.

Si el usuario desea realizar una eliminación masiva de todos los alumnos dados de alta, puede seleccionarlos uno a uno, o hacer clic sobre el primer cuadro de selección. En este caso, el script llamaría a ejecutar en el cliente a la función JavaScript *Seleccion()*. Esta función selecciona a todos los alumnos del listado en caso de no estarlo. Si los alumnos están seleccionados, esta función los deselecciona.

"Eliminar2.php" vuelve a incluir la ejecución del script *"conexion.php"* para llamar a la función *ConsultarBD()*, pasándole esta vez como parámetro, una consulta que obliga a borrar todos los registros de las tablas *usuarios* y *alumnos* cuyos identificadores coincidan con los recibidos del script *"Eliminar1.php"*. También llama al fichero de inclusión *"FuncionesFicheros.inc"* permitiendo efectuar la llamada a la función *EliminarDirectorio()*. Esta función toma como parámetro los mismos identificadores

nombrados anteriormente y borra el directorio personal de cada uno de los alumnos seleccionados.

Realizada la eliminación, comienza la carga en el cliente de la página *"Eliminar2.html"*. En ella se muestra un mensaje, informando que la eliminación ha sido satisfactoria. También presenta un vínculo que llama a la función JavaScript *Volver()*, la cual ordena la ejecución en el servidor del script *"Listado.php"*.

Si el usuario no selecciona ningún alumno para su eliminación, el script *"Eliminar2.php"* manda a cargar en el cliente la página *"Eliminar3.html"*, indicándole al usuario que debe especificar algún alumno.

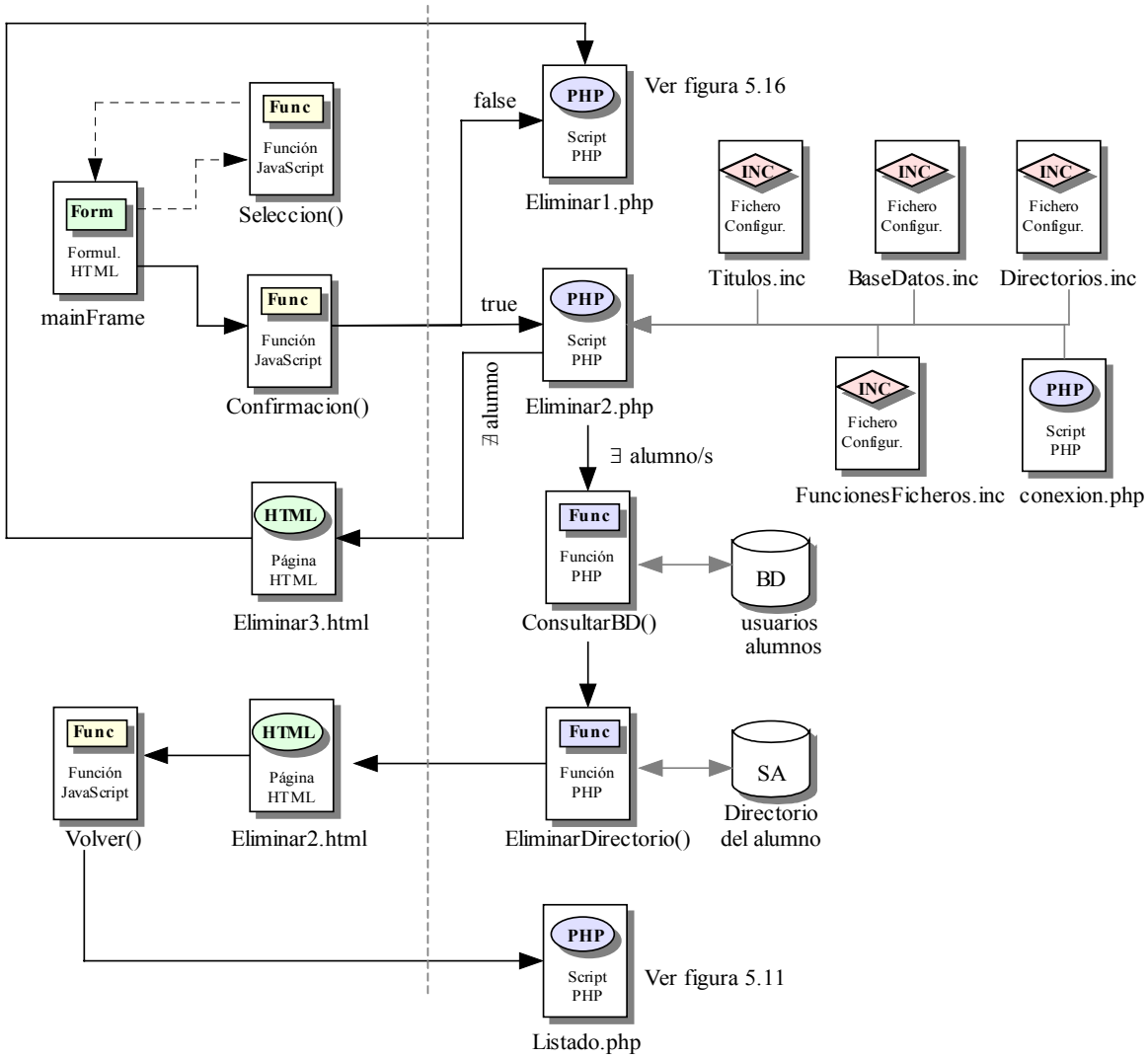


Figura 5.17 D.R. del Módulo de Eliminar Cuentas de Usuarios

5.3.3.5 Módulo de activar/desactivar cuentas de usuarios

Para que un usuario pueda tener acceso a la aplicación, no sólo basta con estar dado de alta en ella, sino que además, debe tener la cuenta de usuario en estado activo. En caso contrario, el acceso le será denegado.

Al igual que en el "Módulo de modificar cuentas de usuarios" del apartado 5.3.3.3, a través de un vínculo presentado en el cliente por el script *"EnlacesAdministracion.php"*, se ordena la ejecución en el servidor del script *"Modificar1.php"*, y éste incluye la ejecución de *"conexion.php"*, mediante el cual se efectúa la consulta de leer todos los registros de la tabla *alumnos* llamando a la función *ConsultarBD()*. El listado de los registros devueltos se muestra en el cliente, donde el usuario debe seleccionar a través de un formulario, un alumno para el cambio de estado de su cuenta. Es entonces cuando se ordena la ejecución del script *"Modificar2.php"*.

"Modificar2.php" recibe como parámetro de entrada, el identificador del alumno seleccionado procedente del script *"Modificar1.php"*. Con este dato e incluyendo la ejecución en el servidor del script *"conexion.php"*, se realiza en el servidor de Base de Datos, la consulta de seleccionar los campos pertenecientes al registro con el mismo identificador que el recibido. Estos valores, incluyendo el identificador del alumno, se muestran en el cliente, en un formulario de edición, donde el usuario podrá realizar la modificación del campo *estado*. Este campo sólo puede tomar dos valores, activo e inactivo. Dependiendo del valor que le adjudique, el alumno tendrá la cuenta activada o no, permitiendo su acceso o no a la aplicación.

La figura 5.18 representa el diagrama de relaciones correspondiente a la creación del formulario de edición.

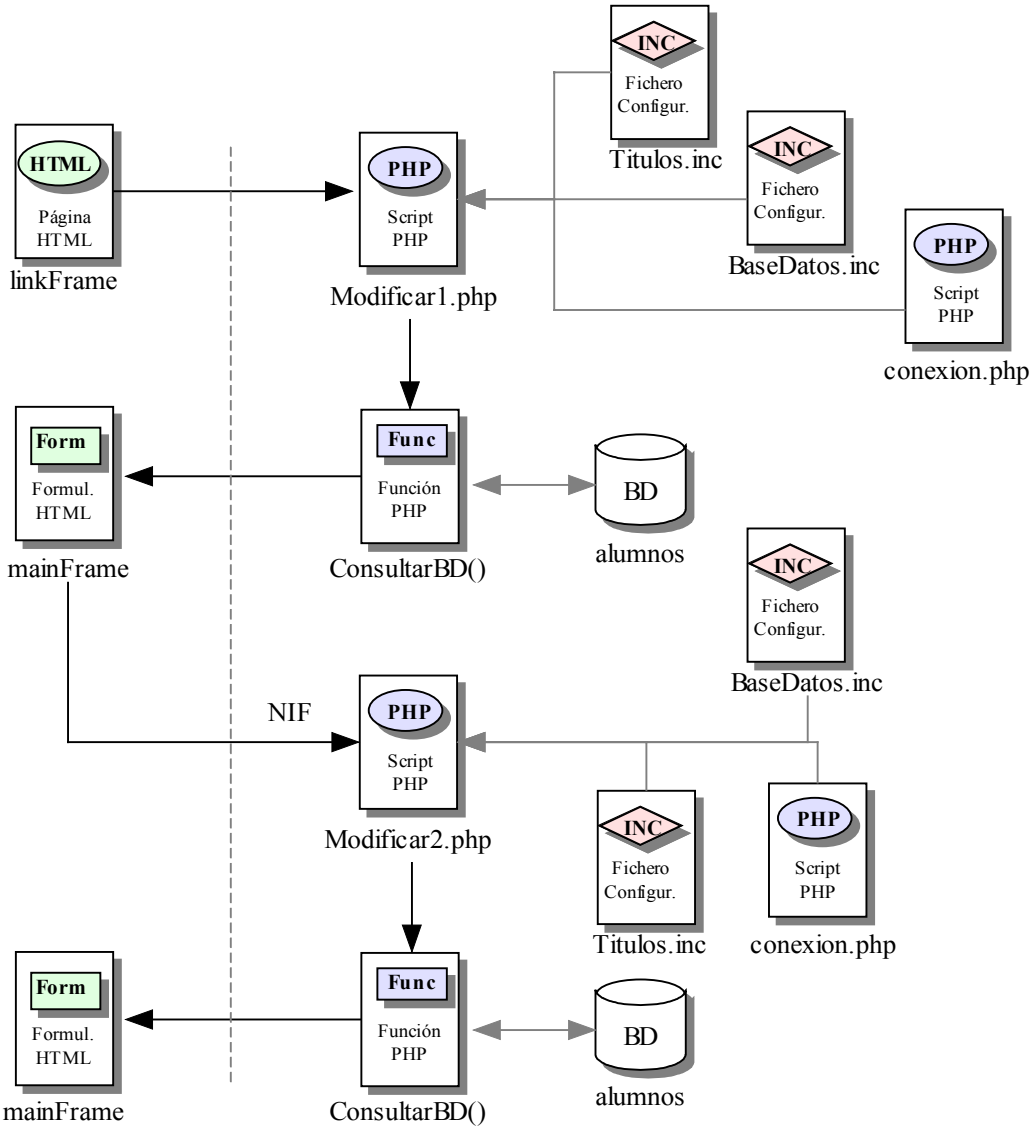


Figura 5.18 D.R. del Módulo de Activar/Desactivar Cuentas. Creación del formulario

Una vez se modifique el dato del estado de la cuenta, se ordena la ejecución en el servidor de un nuevo script, *"Modificar3.php"*, para el envío del formulario de edición.

Este script vuelve a incluir a *"conexion.php"* y llama a la función *ConsultarBD()*, recibiendo esta vez como parámetro, una consulta que modifica el campo *estado* de la tabla *alumnos*. Concluida la operación, el script manda a cargar en el cliente la página *"Modificar3.html"*, informando al usuario que la modificación del alumno ha sido

satisfactoria. A su vez, la página contiene la función JavaScript *Volver()*, que provoca la ejecución del script *"Listado.php"*.

La figura 5.19 muestra el diagrama de relaciones correspondiente al "Módulo de activar/desactivar cuentas de usuarios".

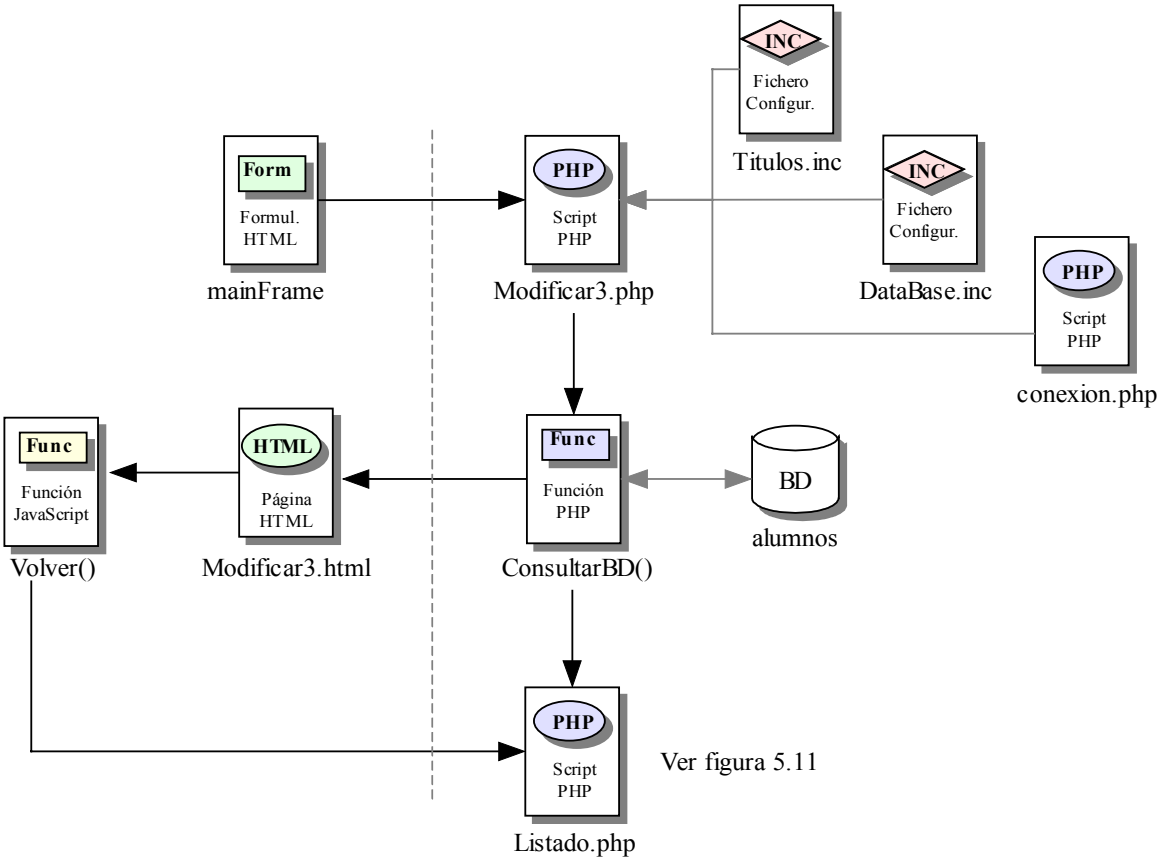


Figura 5.19 D.R. del Módulo de Activar/Desactivar Cuentas de Usuarios

5.3.3.6 Módulo de ordenar usuarios

Existen varios scripts que muestran en el cliente un listado completo de todos los registros de la tabla *alumnos*. El usuario puede ordenar este listado por cualquiera de los campos existentes en ella.

Los scripts *"Listado.php"*, *"Modificar1.php"* y *"Eliminar1.php"* devuelven al cliente el listado de los usuarios de la aplicación. El listado aparece representado en una

tabla cuyas columnas coinciden con los diferentes campos de la tabla *alumnos*. En la cabecera de esta tabla aparece el nombre de estos campos y un vínculo por cada uno de ellos. Estos enlaces invocan la ejecución en el servidor del script de origen.

El script *"Listado.php"* devuelve el listado con todos los campos existentes en la tabla *alumnos*: NIF, nombre, apellidos, teléfono, e-mail, curso y estado. Por tanto, presentará siete diferentes vínculos. Cada uno de estos vínculos volverá a mandar a ejecutar el script *"Listado.php"* pasando un parámetro llamado *orden* que recogerá el propio script para crear la consulta que será mandada al servidor de Base de Datos. La consulta se realizará por orden según este parámetro cuando sea ejecutada mediante la función *ConsultarBD()*. Esta función será ejecutada gracias a la inclusión del script *"conexion.php"*.

De la misma forma, el script *"Modificar1.php"* presenta en el cliente un listado mostrando sólo esta vez, seis de los siete campos de la tabla *alumnos*, por lo que sólo existirán seis vínculos Igualmente, estos vínculos vuelven a ordenar la ejecución en el servidor del script *"Modificar1.php"* recibiendo como parámetro la variable *orden* con el nombre del campo seleccionado en el enlace. El listado será mostrado según el campo recibido en la variable *orden* y mediante la función *ConsultarBD()* e incluyendo el script *"conexion.php"*.

"Eliminar1.php" devuelve el listado también con seis campos y funciona de la misma manera que *"Modificar1.php"*, con la excepción de que los vínculos que se presentan invocan la ejecución en el servidor del propio script *"Eliminar1.php"*.

Existe otro script, *"InterfazAlumno1.php"*, que se detalla en el apartado 5.3.4.2, el cual también devuelve al cliente un listado e incluye los vínculos correspondientes para ordenar la ejecución del propio script, mostrando el listado en orden según el parámetro recibido en la variable *orden*.

5.3.3.7 Módulo de cambio de administrador

Existen dos tipos diferentes de usuarios dentro de la aplicación, los usuarios restringidos y el administrador. La figura del administrador es única pero, no por ello, debe estar fijada a un usuario determinado durante el ciclo de vida de la aplicación. Existe una opción que permite traspasar el papel de administrador de un usuario a otro nuevo.

Uno de los vínculos presentados por el script *"EnlacesAdministracion.php"* en el cliente, ordena la ejecución en el servidor del script *"CambiarAdministrador1.php"*. Este script carga en el cliente un formulario de inserción en el marco *mainFrame*. En este formulario el administrador debe insertar su nombre de usuario y contraseña.

La figura 5.20 muestra el diagrama de relaciones perteneciente a la creación del formulario correspondiente al "Módulo de cambio de administrador".

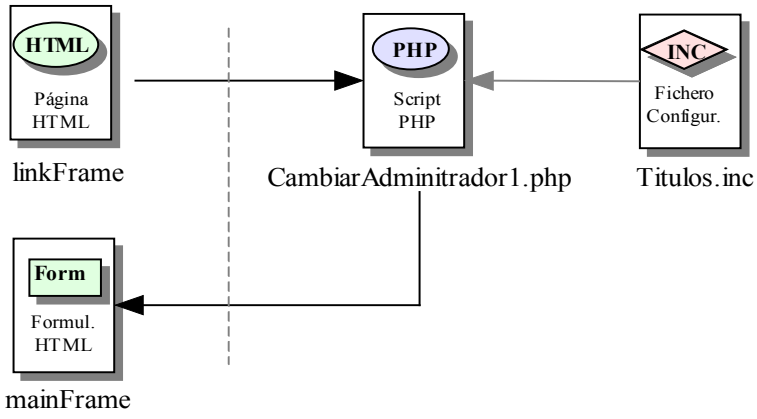


Figura 5.20 D.R. del Módulo de Cambio de Administrador. Creación del formulario

Después se realiza el envío de dicho formulario invocando la ejecución del script *"CambiarAdministrador2.php"*.

Este script incluye la ejecución del script *"conexion.php"* y crea una consulta que busca en la tabla *alumnos* si los datos recibidos en el formulario son correctos, es decir, si existe algún registro con dichos valores. En caso de no encontrarse, el script muestra en el cliente un mensaje informando al usuario de que los datos no son correctos junto a un

botón de conformidad que llama a una función propia del lenguaje JavaScript, la cual obliga a ejecutar el script anterior. En este caso, se volvería a ejecutar el script *"CambiarAdministrador1.php"*.

Si por el contrario, la consulta informa que los datos son correctos, el script carga en el cliente un nuevo formulario de inserción con tres campos, donde se deberán introducir los datos del nuevo usuario que será administrador de la aplicación. Después de esto, se llama a la función JavaScript *Verificar()*, la cual comprobará que todos los campos han sido rellenados. Inmediatamente después, se manda a ejecutar en el servidor el script *"CambiarAdministrador3.php"*.

"CambiarAdministrador3.php" incluye la ejecución del script *"conexion.php"* e incluye el fichero *"FuncionesFicheros.inc"*. Mediante el primero, llama a la función *ConsultarBD()*, pasándole como parámetro, una consulta que elimina el registro de la tabla *usuarios* que tenga en el campo *tipo* el valor administrador. Por medio del fichero *"FuncionesFicheros.inc"* llama a la función *EliminarDirectorio()* que recibe como parámetro el directorio personal del administrador. Esta función elimina dicho directorio y todo lo contenido en él.

Posteriormente, el script realiza otra consulta al servidor de Base de Datos insertando un nuevo registro en la tabla *usuarios* con los datos recibidos del formulario de inserción enviado por el script *"CambiarAdministrador2.php"* y con el campo *tipo* igual a administrador. Se crea entonces una carpeta personal para el nuevo usuario administrador y todos las subcarpetas necesarias para el correcto funcionamiento de la herramienta.

Por último, el script presenta en el cliente un mensaje informando que se ha producido el cambio de administrador y ofreciendo al antiguo administrador el cierre de la aplicación para no ocasionar futuros problemas. Para ello, se ejecuta en el cliente la función JavaScript *top.close()*. Esta función provoca el cierre de la sesión.

En la figura 5.21 se muestra el diagrama de relaciones correspondiente al "Módulo de cambio de administrador".

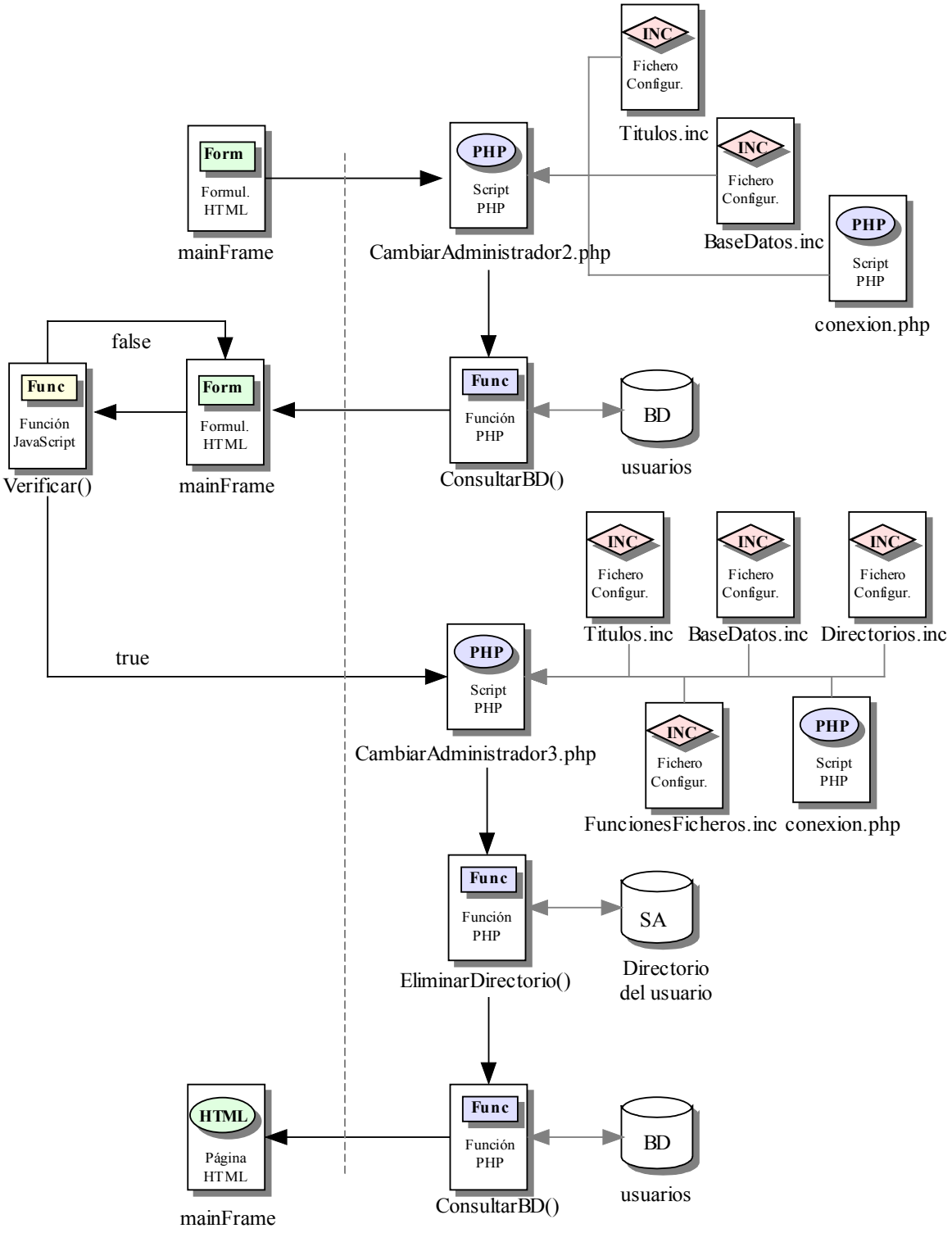


Figura 5.21 D.R. del Módulo de Cambio de Administrador

5.3.4 Otras opciones de administración

Además de la "Gestión de usuarios", el administrador de la aplicación puede realizar otras tareas únicas de su categoría. Estas operaciones son accedidas por el usuario desde el marco *linkFrame* y desde el marco *topFrame*.

5.3.4.1 Módulo de entorno de administrador

Existen dos bloques bien diferenciados en la aplicación, el bloque de trabajo y el bloque de administración. Sólo el usuario administrador tiene privilegios de acceso a ambos bloques y, por tanto, debe tener la posibilidad de pasar de uno a otro sin necesidad de cerrar la sesión y volver a abrir una nueva. Este módulo permite al administrador pasar del bloque de administración al bloque de trabajo.

Como se ha mencionado en apartados anteriores, *"EnlacesAdministracion.php"* presenta en el cliente una serie de vínculos. Uno de estos vínculos, ordena la ejecución del script *"InterfazAdministrador1.php"*. Lo primero que hace este script es desregistrar de la sesión al usuario para posteriormente registrarlo con el identificador del usuario administrador (el cual debe coincidir con el usuario actual). Este identificador fue registrado en el script *"index.php"*, una vez se comprueba que el usuario es el administrador.

Realizado este paso, el script *"InterfazAdministrador1.php"* incluye el script *"FramesAdministrador2.php"* que se ejecuta en el servidor y que realiza exactamente el mismo trabajo que el script *"FramesAdministrador.php"*, el cual se detalla en el apartado 5.3.5 del presente capítulo. La razón de que existan dos scripts que realicen las mismas instrucciones es puramente por motivos de rutas de directorios.

Existe otro vínculo en la aplicación que ordena la ejecución del script *"InterfazAdministrador1.php"* y que, por lo tanto, también realiza la operación del "Módulo de entorno de administrador". Este vínculo se sitúa en el marco *topFrame* y el script *"BanerAdministracion.php"* es el encargado de presentarlo en el cliente.

La figura 5.22 muestra el diagrama de relaciones correspondiente al "Módulo de entorno de administrador".

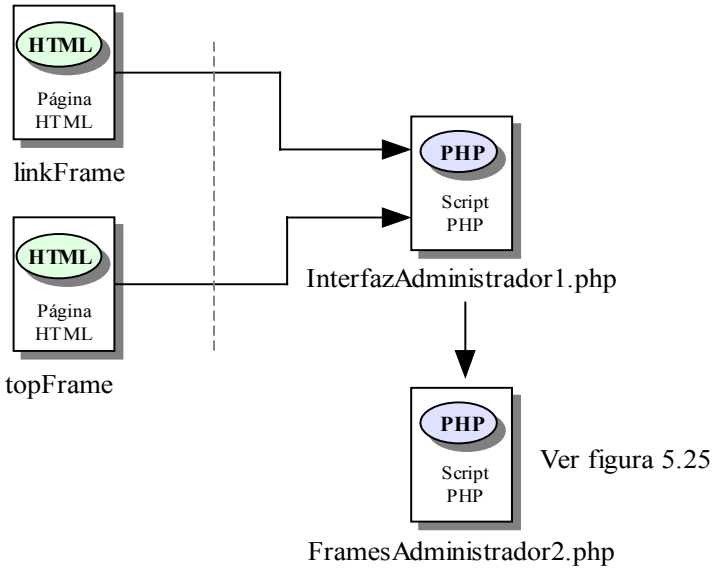


Figura 5.22 D.R. del Módulo de Entorno de Administrador

5.3.4.2 Módulo de entorno de alumno

El administrador de la aplicación tiene el privilegio de acceder tanto a su bloque de trabajo como al de todos los alumnos. Este acceso sólo tiene lugar desde el bloque de administración.

De todos los vínculos presentados por el script *"EnlacesAdministracion.php"* en el cliente, existe uno que manda a ejecutar en el servidor el script *"InterfazAlumno1.php"*. Este script incluye la ejecución de *"conexion.php"* y llama a la función *ConsultarBD()* pasando como parámetro una consulta que devuelve el listado de todos los alumnos. La consulta se envía al servidor de Base de Datos y éste devuelve todos los registros de la tabla *alumnos*. Presentado este listado en el cliente en el marco *mainFrame*, el administrador debe seleccionar a uno de ellos para acceder a su bloque de trabajo.

Inmediatamente después de aceptada la selección mediante un formulario, se solicita la ejecución en el servidor del script *"InterfazAlumno2.php"*. Este script recibe el

identificador del alumno mediante el formulario enviado en el script *"InterfazAlumno1.php"*. A partir de este momento, *"InrterfazAlumno2.php"* sólo realiza dos pasos. Primero, desregistra de la sesión al usuario actual, o sea, al administrador, para registrar como usuario de la sesión al usuario cuyo identificador sea el recibido por el formulario, es decir, el alumno seleccionado para acceder a su bloque de trabajo. Por último, el script incluye la ejecución de *"FramesAdministrador2.php"*, el cual realiza las mismas operaciones que el script *"FramesAdministrador.php"* (ver apartado 5.3.5), es decir, da paso al bloque de trabajo del usuario registrado en la sesión. Por tanto, se estaría accediendo al bloque de trabajo del alumno previamente seleccionado. La razón de que existan dos scripts que realicen las mismas instrucciones es puramente por motivos de rutas de directorios.

Existe otro vínculo en la aplicación que ordena la ejecución del script *"InterfazAlumno1.php"* y, que por lo tanto, también realiza la operación del "Módulo de entorno de alumno". Este vínculo se sitúa en el marco *topFrame* y el script *"BanerAdministracion.php"* es el encargado de presentarlo en el cliente.

La figura 5.23 muestra el diagrama de relaciones correspondiente al "Módulo de entorno de alumno".

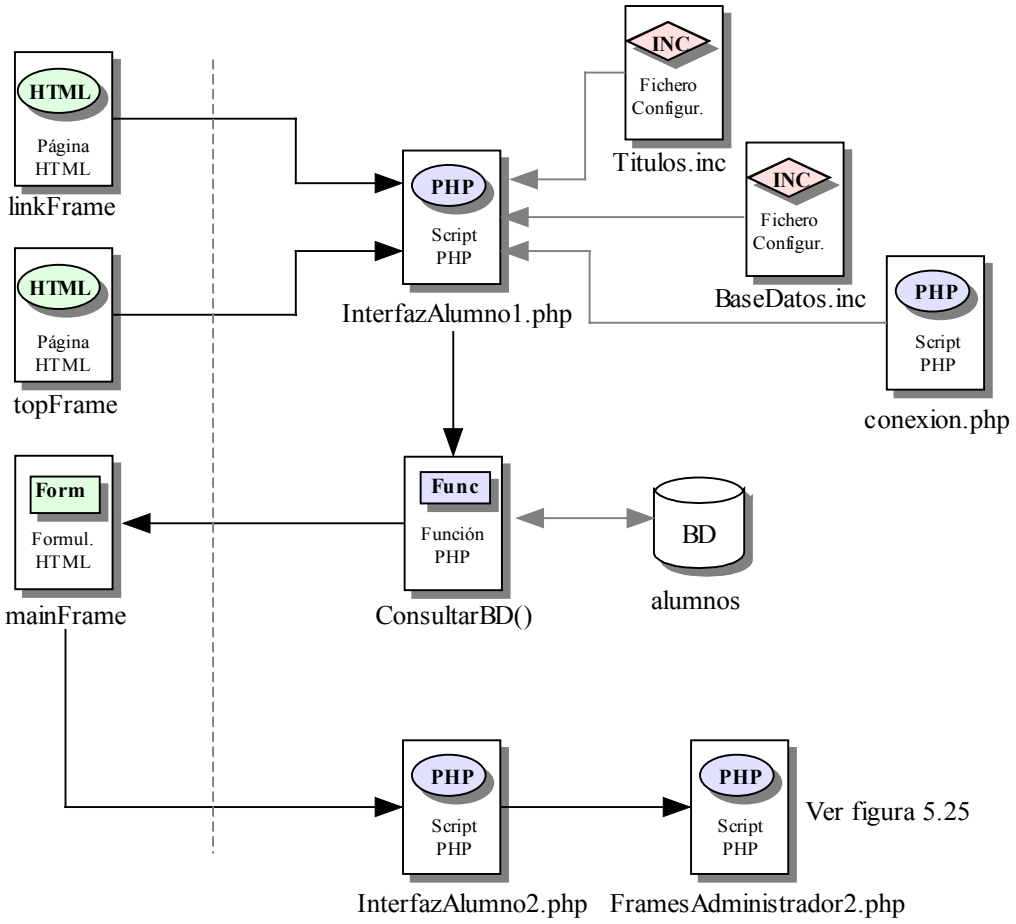


Figura 5.23 D.R. del Módulo de Entorno de Alumno

5.3.4.3 Módulo de manual de administración

Como en toda aplicación, es indispensable la ayuda de un manual de usuario para el manejo correcto de la herramienta. Dentro del bloque de administración se proporciona el acceso a un manual de administrador. Este manual sólo detalla las operaciones propias del bloque de administración, es decir, quedan excluidas todas aquellas opciones pertenecientes al bloque de trabajo, las cuales están habilitadas, no sólo por el administrador, sino también por todos los usuarios restringidos.

Para acceder a este manual, ejecutado el script *"BanerAdministracion.php"* en el servidor, en el marco *topFrame*, se presenta un vínculo que llama a la función JavaScript *Manual()*. Esta función abre una nueva ventana de navegación ordenando la carga en ella del archivo que contiene el manual de administrador.

La figura 5.24 muestra el diagrama de relaciones correspondiente al "Módulo de manual de administración".

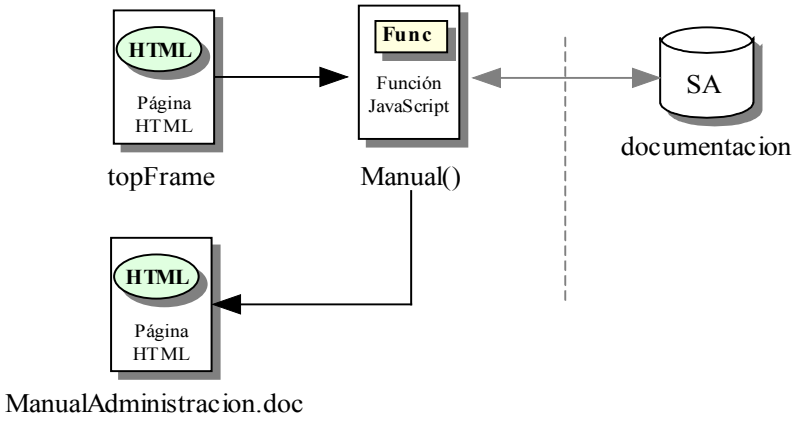


Figura 5.24 D.R. del Módulo de Manual de Administración

5.3.5 Bloque de trabajo

A este bloque se accede desde el script *"index.php"*. Corresponde con el script *"FramesAdministrador.php"* o *"FramesAlumno.php"*, según el tipo de usuario registrado en la sesión. Ambos devuelven al cliente una página con cuatro marcos, *topFrame*, *leftFrame*, *bottonFrame* y *mainFrame*, que se sitúan, respectivamente, en la parte superior, izquierda, derecha superior y derecha inferior de la página.

El diagrama de relaciones de petición de los marcos se muestra en la figura 5.25.

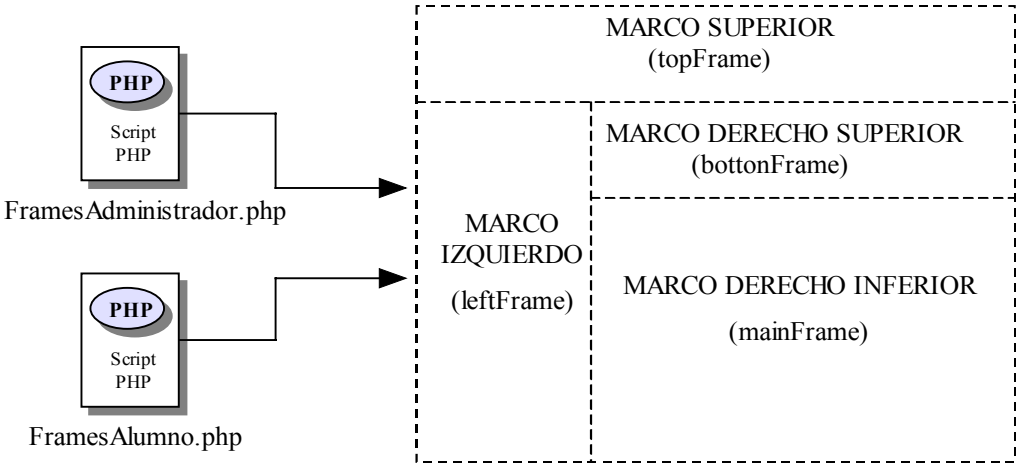


Figura 5.25 D.R. del Bloque de Trabajo. Petición de marcos

Cada marco en el cliente realiza una nueva petición al servidor de ejecución de scripts. En el caso de proceder del script *"FramesAdministrador.php"*, el marco *topFrame* pide la ejecución en el servidor del script *"BanerAdminsitrador.php"*. En caso de proceder de *"FramesAlumno.php"*, el script solicitado es *"BanerAlumno.php"*. Ambos devuelven al cliente una página web que se carga en el marco superior y que contiene funciones JavaScript.

Por otra parte, indiferentemente del script del que proceda, el marco *leftFrame* manda a ejecutar el script *"foldertree.php"*, el cual muestra en el cliente una estructura de árbol jerárquico mediante funciones encontradas en los ficheros *DefineMyTree.js*, *ftiens4.js* y *ua.js*.

El marco *bottonFrame* pide la ejecución de *"MenuAdministrador.php"* que incluye todas las funciones propias del bloque de trabajo de la aplicación, encontradas en el fichero *Funciones.js*, y devuelve una página con un icono distintivo por cada una de estas funciones.

El marco *mainFrame* llama a ejecutar el script *"PrincipalAdministrador.php"*, que devuelve en el cliente una página en la parte derecha inferior y cuyo contenido dependerá del script que solicite su ejecución.

En la figura 5.26 se muestra el diagrama de relaciones de construcción de marcos correspondiente al "Bloque de trabajo".

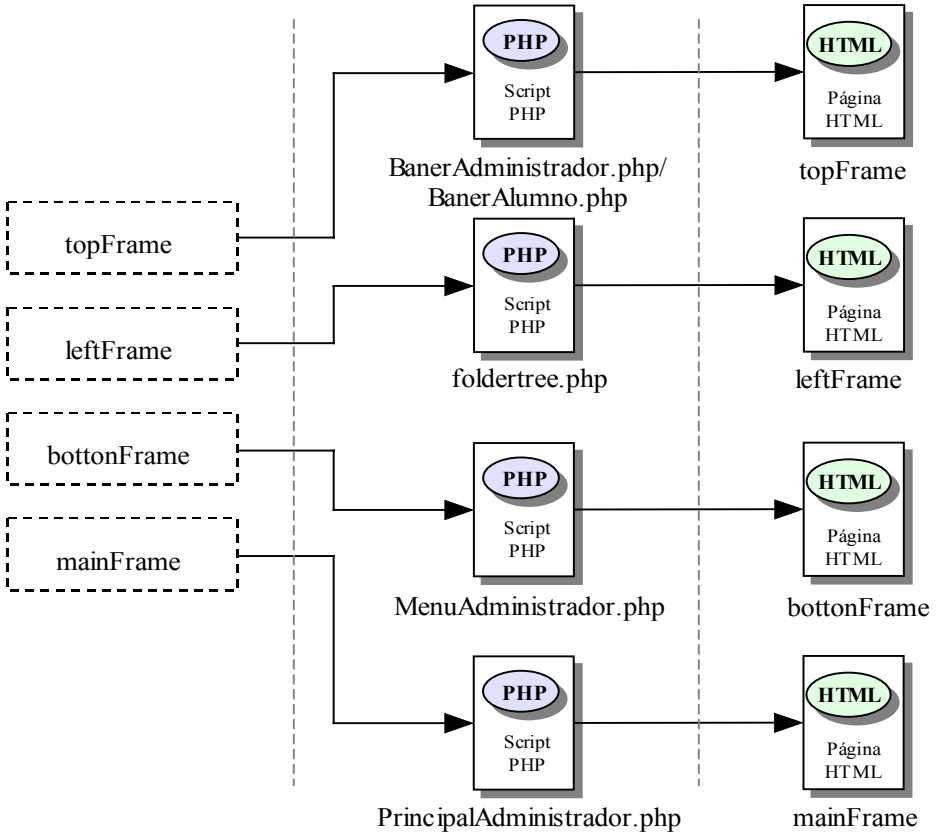


Figura 5.26 D.R. del Bloque de Trabajo. Construcción de marcos

5.3.6 Gestión de directorios

Todo usuario de la aplicación tiene asignado un directorio personal dentro del sistema de archivos del servidor. Dentro de este directorio personal, se crean cinco carpetas: *configuracion*, *ejecucion*, *practicas*, *procesos* y *temporal*. De las cinco carpetas, la aplicación asigna al directorio *practicas* como directorio de trabajo donde almacenar todos los ficheros que el usuario vaya desarrollando. Al mismo tiempo, dentro de este directorio, el usuario puede crear y eliminar tantas carpetas como desee. Las opciones de Crear directorio y Borrar directorio, son solicitadas por el usuario desde el marco *bottonFrame*.

5.3.6.1 Módulo de crear directorios

El sistema crea un directorio llamado *practicas* por cada usuario dado de alta en la aplicación. Este directorio corresponde con el que la aplicación muestra en el bloque de trabajo y todos los archivos que el usuario va creando, sólo pueden ser guardados dentro de esta carpeta. Por este motivo, la aplicación ofrece la posibilidad de crear nuevos directorios dentro de ésta, permitiendo al usuario estructurar todos sus archivos.

La página mostrada por la ejecución del script *"MenuAdministrador.php"* incluye el fichero *"Funciones.js"* y muestra en el marco *bottonFrame*, una serie de vínculos acompañados de iconos distintivos según la funcionalidad de cada uno de estos enlaces.

De los once vínculos mostrados, existe uno que llama a la función JavaScript *creardir()*, la cual se encuentra definida en "*Funciones.js"*. Esta función, a su vez, llama a otra función JavaScript *window.open()*, propia de este lenguaje, que ordena la apertura de una nueva ventana de navegación. Esta función se ejecuta en el cliente y tiene como parámetro, entre otros, el nombre del script que se ordena a ejecutar en el servidor con la apertura de la nueva ventana. Este script corresponde con *"FramesCrearDir.php"* y ordena la carga de una página dividida en tres marcos: *CrearDirectorio1*, *CrearDirectorio2* y *CrearDirectorio3*, los cuales se sitúan en la parte superior, media e inferior, respectivamente, de la página. Cada uno de estos marcos, en el cliente, realiza una nueva petición de ejecución de scripts en el servidor.

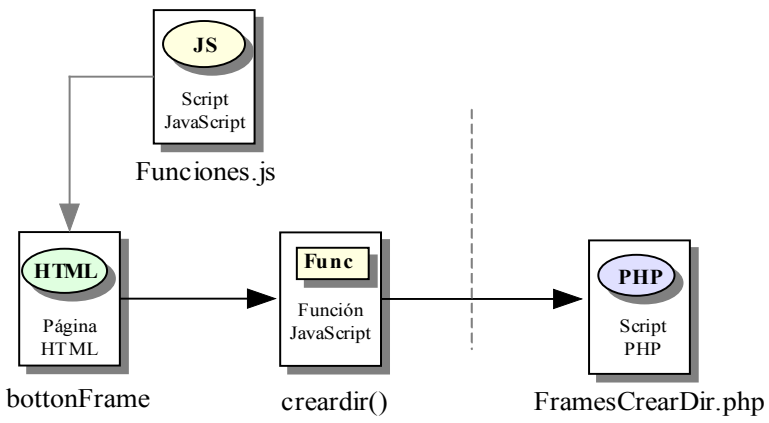


Figura 5.27 D.R. de la llamada al Módulo de Crear Directorios

La figura 5.28 muestra el diagrama de relaciones perteneciente a la petición de marcos del "Módulo de crear directorios".

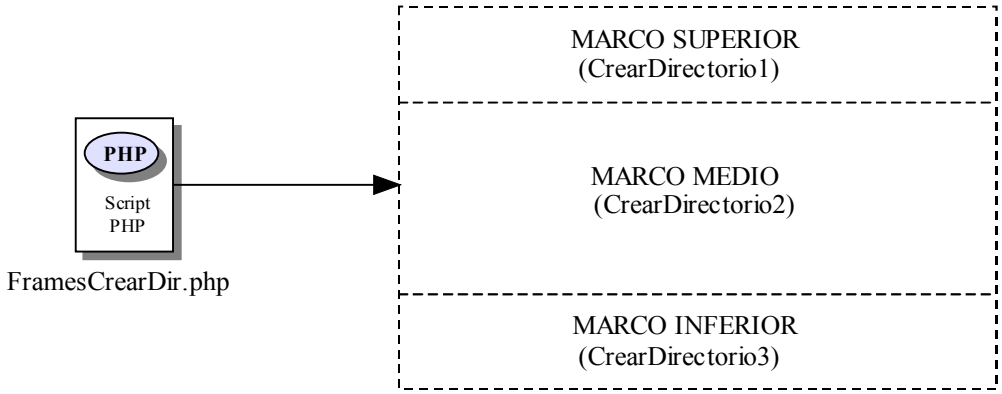


Figura 5.28 D.R. del Módulo de Crear Directorios. Petición de marcos

El marco *CrearDirectorio1* ordena la ejecución del script *"CrearDirectorio1.php"*. Este script mediante la inclusión del fichero *"Titulos.inc"*, carga en el cliente una página con el título propio del bloque de trabajo y la opción escogida de Crear Directorio. También incluye un formulario, *FormularioCrearDir1*, con un sólo campo llamado *ruta*. Este campo es de sólo lectura y no puede ser editado por el usuario.

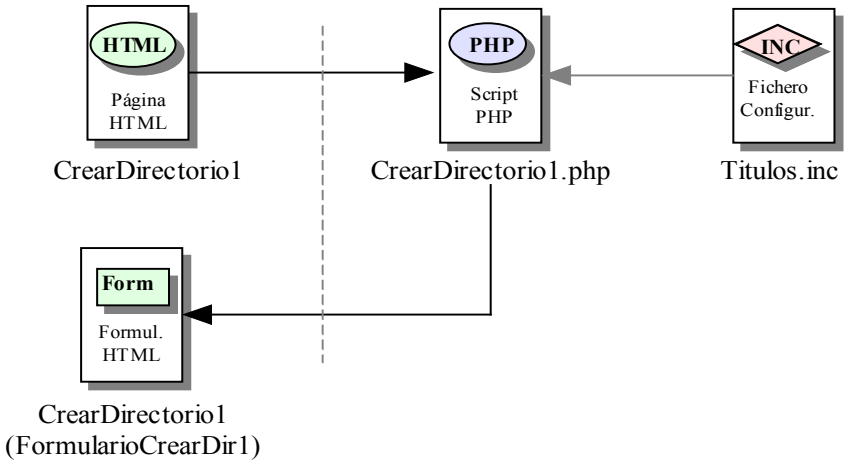


Figura 5.29 D.R. del Módulo de Crear Directorios. Marco CrearDirectorio1

El marco *CrearDirectorio2* ordena la ejecución del script *"CrearDirectorio2.php"* y le pasa un parámetro llamado *dir*, cuyo valor coincide con la ubicación del directorio *practicas* del usuario registrado en la sesión. Este valor es calculado gracias a la inclusión del fichero *"Directorios.inc"* en el script *"FramesCrearDir.php"*.

El script *"CrearDirectorio2.php"* llama a una función PHP llamada *Mostrar()*. Esta función recibe dos parámetros que coinciden con dos directorios, y efectúa, mediante un bucle, una lectura completa del directorio pasado como primer parámetro. Este primer parámetro corresponde siempre con el valor de la variable *dir* recibida por el script. A medida que realiza la lectura, examina cada uno de los elementos encontrados.

Si el elemento es un directorio, el script presenta un vínculo que ordena de nuevo la ejecución de *"CrearDirectorio2.php"* pero con el parámetro *dir* igual a la ruta del directorio encontrado.

Por el contrario, si el elemento es un archivo, calcula si dicho archivo es ejecutable o no, simplemente para asignarle una imagen u otra en la página a cargar en el cliente. En esta ocasión, el script no muestra vínculo ninguno.

"CrearDirectorio2.php" también divide la ruta recibida en la variable *dir* separando la raíz, indicada en el fichero *"Directorios.inc"*, del resto, y lo almacena en otra variable llamada *dir_visible*. Luego le pasa este valor a la función JavaScript *ActualizarDirectorio()*, la cual se encarga de cargar el dato en el formulario *FormularioCrearDir1* situado en el marco *CrearDirectorio1*.

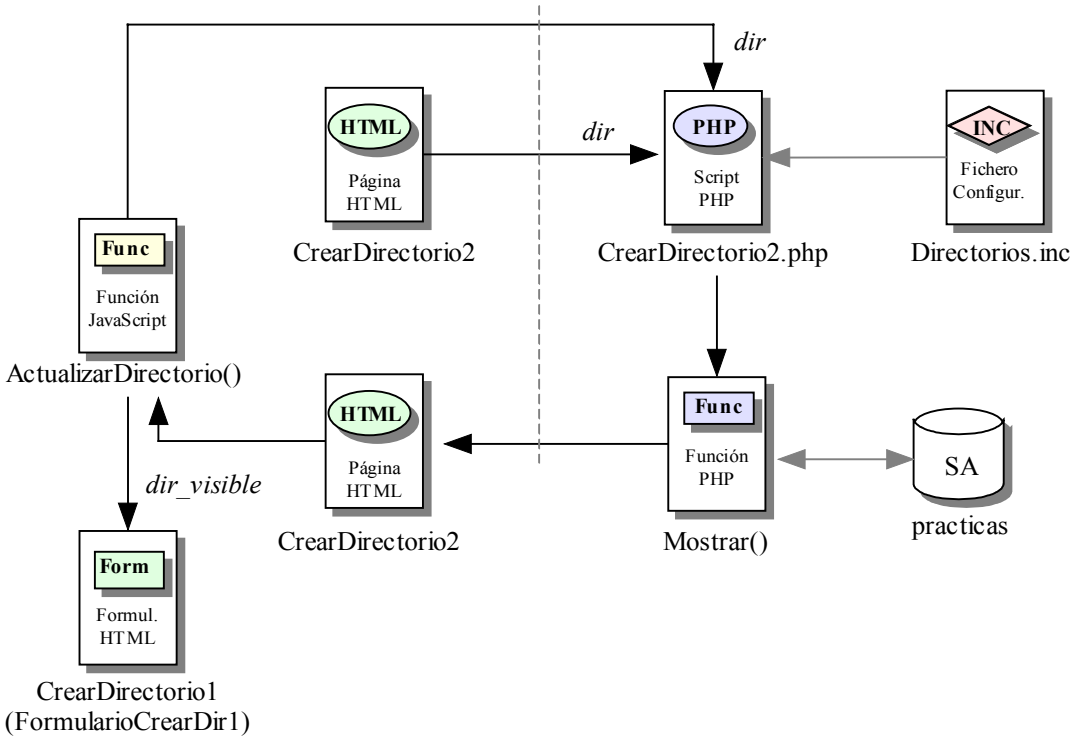


Figura 5.30 D.R. del Módulo de Crear Directorios. Marco CrearDirectorio2

El marco *CrearDirectorio3* ordena la ejecución del script *"CrearDirectorio3.php"* en el servidor. Este script llama a la función PHP *Formulario()*, la cual muestra en el cliente un formulario, *FormularioCrearDir3*, que contiene dos campos, *rutadir* y *dire*, y un botón de envío. También muestra un vínculo que lanza una llamada a la función JavaScript *top.close()*, que ordena el cierre de la ventana de navegación.

Si el usuario pulsa el botón del formulario, se ordena la ejecución en el cliente de la función JavaScript *Actualizar()*. Esta función carga el campo *rutadir* con el valor del campo *ruta* del formulario *FormularioCrearDir1*, situado en el marco *CrearDirectorio1*. Realizada esta operación, se vuelve a invocar la ejecución en el servidor del mismo script *"CrearDirectorio3.php"*. Es entonces cuando el script calcula que el valor del campo *dire*, que es rellenado por el usuario, no contiene ningún carácter especial. Si lo contiene, llama a la función JavaScript *Caracteres()*, la cual alerta al usuario de lo sucedido y el script vuelve a cargar el formulario mediante la llamada a la función *Formulario()*. Si, por el contrario, no existe ningún carácter especial en el campo *dire*, el script *"CrearDirectorio3.php"*, crea el nuevo directorio en la ruta indicada por el campo *rutadir*

y con el nombre indicado en el campo *dire*. Creado el nuevo directorio, la ventana es cerrada por la función JavaScript *top.close()*.

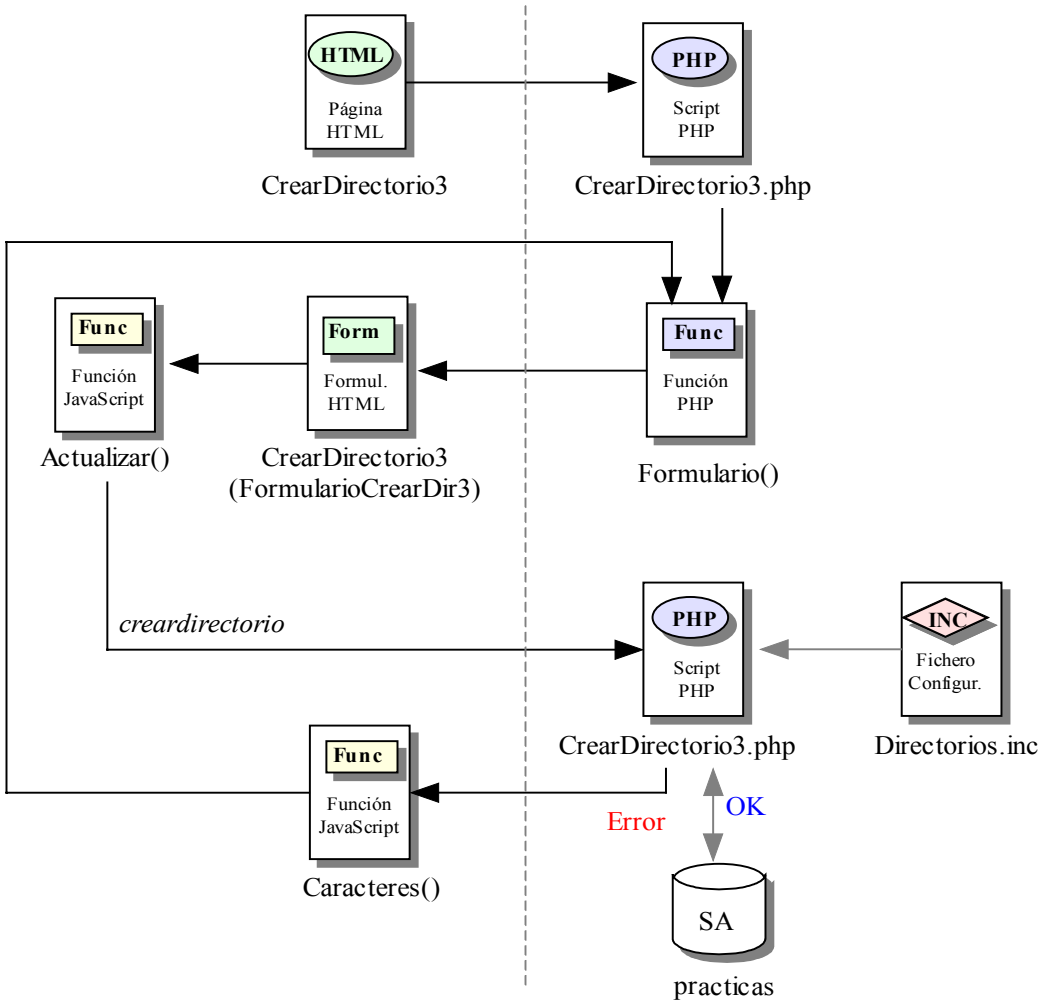


Figura 5.31 D.R. del Módulo de Crear Directorios. Marco CrearDirectorio3

5.3.6.2 Módulo de eliminar directorios

En el apartado anterior, se detalla cómo todo usuario puede crear, de forma ilimitada, tantos directorios como desee, dentro de su carpeta personal de trabajo. Esta opción de crear nuevos directorios, permite al usuario estructurar los archivos que va creando en la aplicación. Sin embargo, de la misma manera que se crean nuevas carpetas, éstas pueden ser eliminadas. El "Módulo de eliminar directorios" realiza la operación de borrar aquellos directorios que seleccione el usuario, junto a todo su contenido.

El script *"MenuAdministrador.php"* ejecutado en el servidor, muestra en el cliente una serie de vínculos que corresponden con las diferentes operaciones que ofrece el bloque de trabajo al usuario. Estos vínculos, mostrados en el marco *bottonFrame*, llaman a distintas funciones JavaScript definidas en un fichero de inclusión llamado *"Funciones.js"*.

La función que es llamada en este módulo es *borrardir()*. Esta función vuelve a ejecutar otra función en el cliente, propia del lenguaje JavaScript, *window.open()*, y ésta provoca la apertura de una nueva ventana de navegación. El script mandado a ejecutar en el servidor con la apertura de la nueva ventana es *"FramesBorrarDir.php"*.

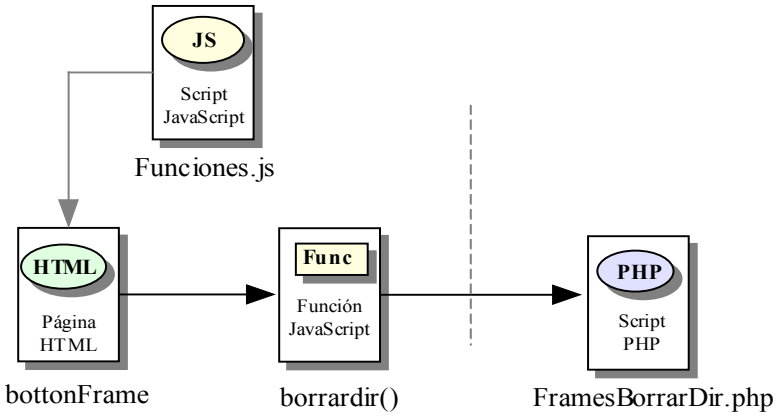


Figura 5.32 D.R. de la llamada al Módulo de Eliminar Directorios

"FramesBorrarDir.php" muestra en el cliente una página con tres marcos: *BorrarDirectorio1*, *BorrarDirectorio2* y *BorrarDirectorio3*, situados en la parte superior, media e inferior, respectivamente, de la página. Cada uno de estos marcos ordena la ejecución de nuevos scripts.

La figura 5.33 muestra el diagrama de relaciones perteneciente a la petición de marcos del "Módulo de crear directorios".

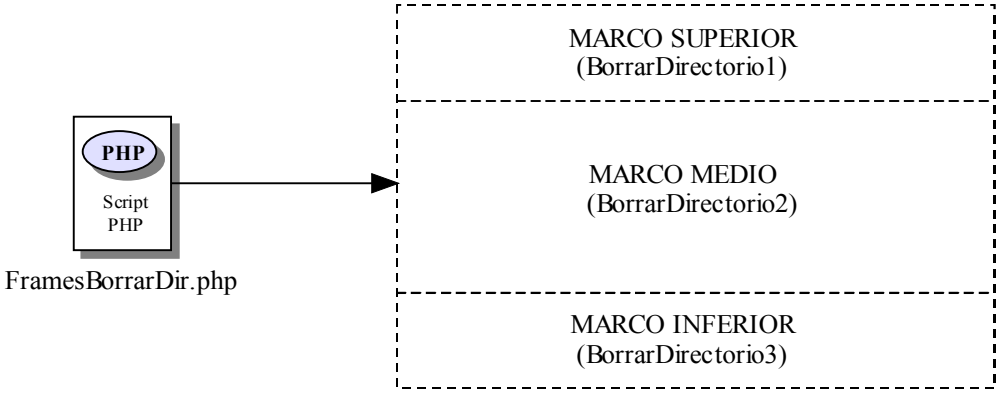


Figura 5.33 D.R. del Módulo de Eliminar Directorios. Petición de marcos

El marco *BorrarDirectorio1* ordena la ejecución en el servidor del script *"BorrarDirectorio1.php"*. Este script, mediante la inclusión del fichero *"Titulos.inc"*, carga una página que muestra en el cliente el título propio del bloque de trabajo. Además, el script carga un formulario, *FormularioBorrarDir1*, con un solo campo llamado *ruta*. Este campo es de sólo lectura y no modificable por el usuario.

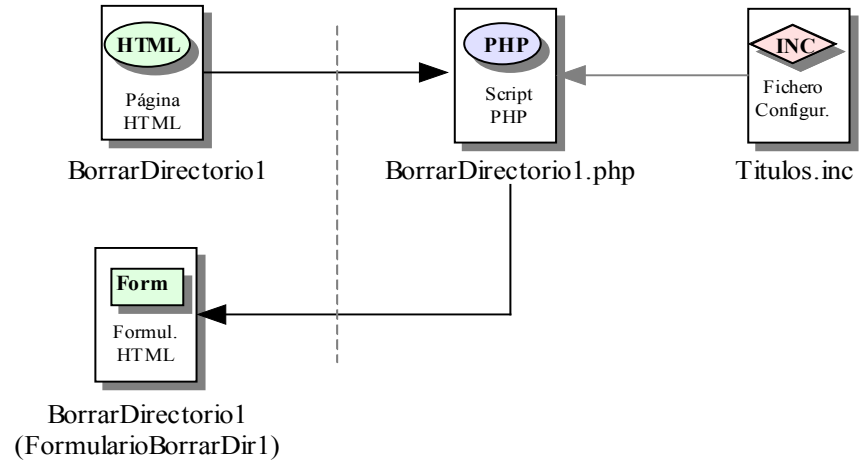


Figura 5.34 D.R. del Módulo de Eliminar Directorios. Marco BorrarDirectorio1

El marco *BorrarDirectorio2* manda la ejecución en el servidor del script *"BorrarDirectorio2.php"*. Este script recibe por parámetro una variable llamada *dir*. Esta variable, en principio toma el valor de la ruta del directorio *practicas* ubicado al usuario registrado en la sesión. Luego separa de esta variable la raíz, especificada en el fichero de

inclusión *"Directorios.inc"*, del resto, y lo almacena en otra variable llamada dir_visible. Esta variable es pasada como parámetro a la función JavaScript *Actualizar()* ejecutada en el cliente. La función carga el valor recibido, en el campo *ruta* perteneciente al formulario *FormularioBorrarDir1*, situado en el marco *BorrarDirectorio1*.

Pero previamente, *"BorrarDirectorio2.php"* llama a la función PHP *Mostrar()*. La función recibe dos parámetros que coinciden con dos directorios, y efectúa, mediante un bucle, una lectura completa del directorio pasado como primer parámetro, el cual corresponde con la variable *dir*. A medida que realiza esta lectura, examina cada uno de los elementos encontrados, distinguiendo si se trata de un directorio o un archivo. Dependiendo del caso, el script realiza unas u otras instrucciones.

En el caso de que el elemento sea un directorio, muestra un vínculo que ordena a ejecutar de nuevo al propio script *"BorrarDirectorio2.php"* pasando como parámetro, la variable *dir* con el valor del directorio encontrado.

Si por el contrario, el elemento es un archivo, el script muestra en el cliente el nombre del archivo sin vínculo ninguno.

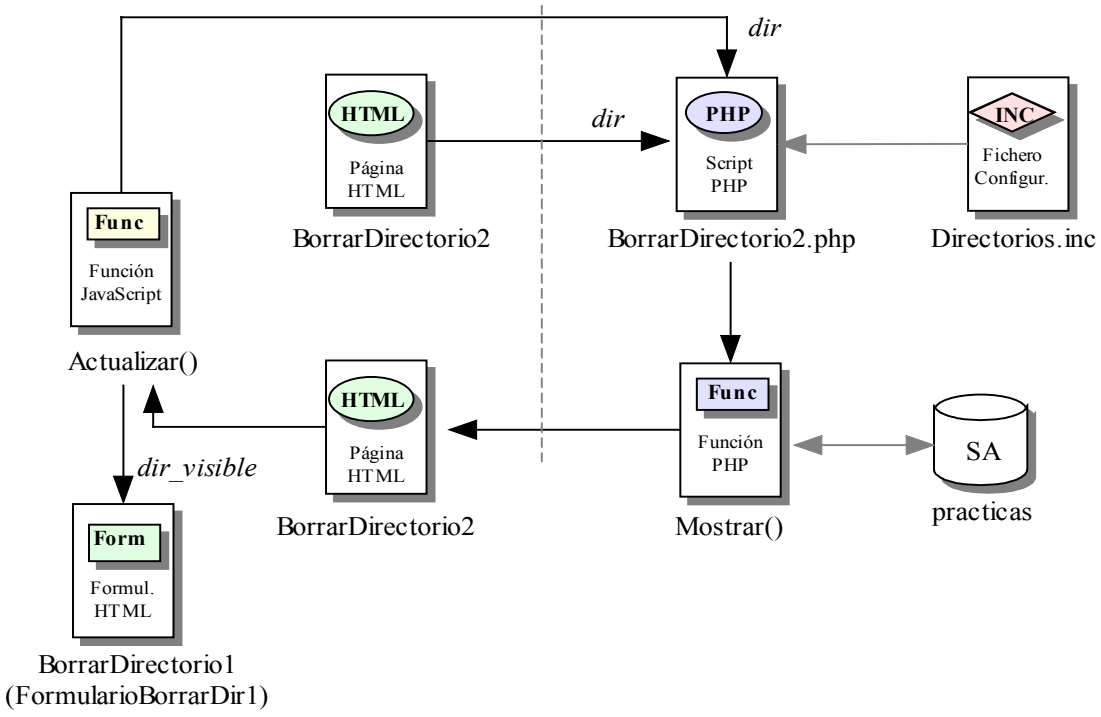


Figura 5.35 D.R. del Módulo de Eliminar Directorios. Marco BorrarDirectorio2

El marco *BorrarDirectorio3* ordena la ejecución en el servidor del script *"BorrarDirectorio3.php"*. Este script carga en el cliente un formulario, *FormularioBorrarDir3*, que contiene dos campos ocultos llamados *rutadir* y *opcion*. Si el usuario pulsa sobre el botón de envío del formulario, se ordena la ejecución en el cliente de la función JavaScript *CargarDatos()*.

Esta función lee el valor que tiene el campo *ruta* de *FormularioBorrarDir1*, presentado en el marco *BorrarDirectorio1*, y se asegura que no esté vacío. En caso de estarlo, muestra al usuario un mensaje de alerta. En caso de que el campo *ruta* tenga algún valor, la función espera la corroboración del usuario para cargar el campo *rutadir* del formulario *FormularioBorrarDir3* con el valor del campo *ruta* del formulario *FormularioBorrarDir1*. La función *CargarDatos()* también carga el campo *opcion* del formulario *FormularioBorrarDir3* con el valor 'Eliminar'. Cargados los campos del formulario, la función JavaScript lo envía y vuelve a ordenar la ejecución en el servidor del propio script *"BorrarDirectorio3.php"*. Esta vez, el script reconoce que el formulario *FormularioBorrarDir3* ha sido enviado y llama a la función PHP *Eliminar()*.

La función *Eliminar()* recibe por parámetro el valor del campo *rutadir* y elimina de forma recursiva el directorio recibido y todo su contenido. Por último, el script ordena la ejecución en el cliente de la función JavaScript *RestaurarArbol()*. Esta función manda la ejecución del script *"DirEliminado.php"* en el marco padre, es decir, sin los tres marcos anteriores.

El script carga en el cliente una página, informando al usuario que el directorio ha sido eliminado y mostrando un formulario sin campos y con un botón de conformidad. Pulsado este botón, se ejecuta la función JavaScript *top.close()*, la cual produce el cierre de la ventana abierta en el "Módulo de eliminar directorios" y, se ordena la ejecución en el servidor del script *"foldertree.php"* en el marco *leftFrame*, definido en el apartado del "Bloque de trabajo".

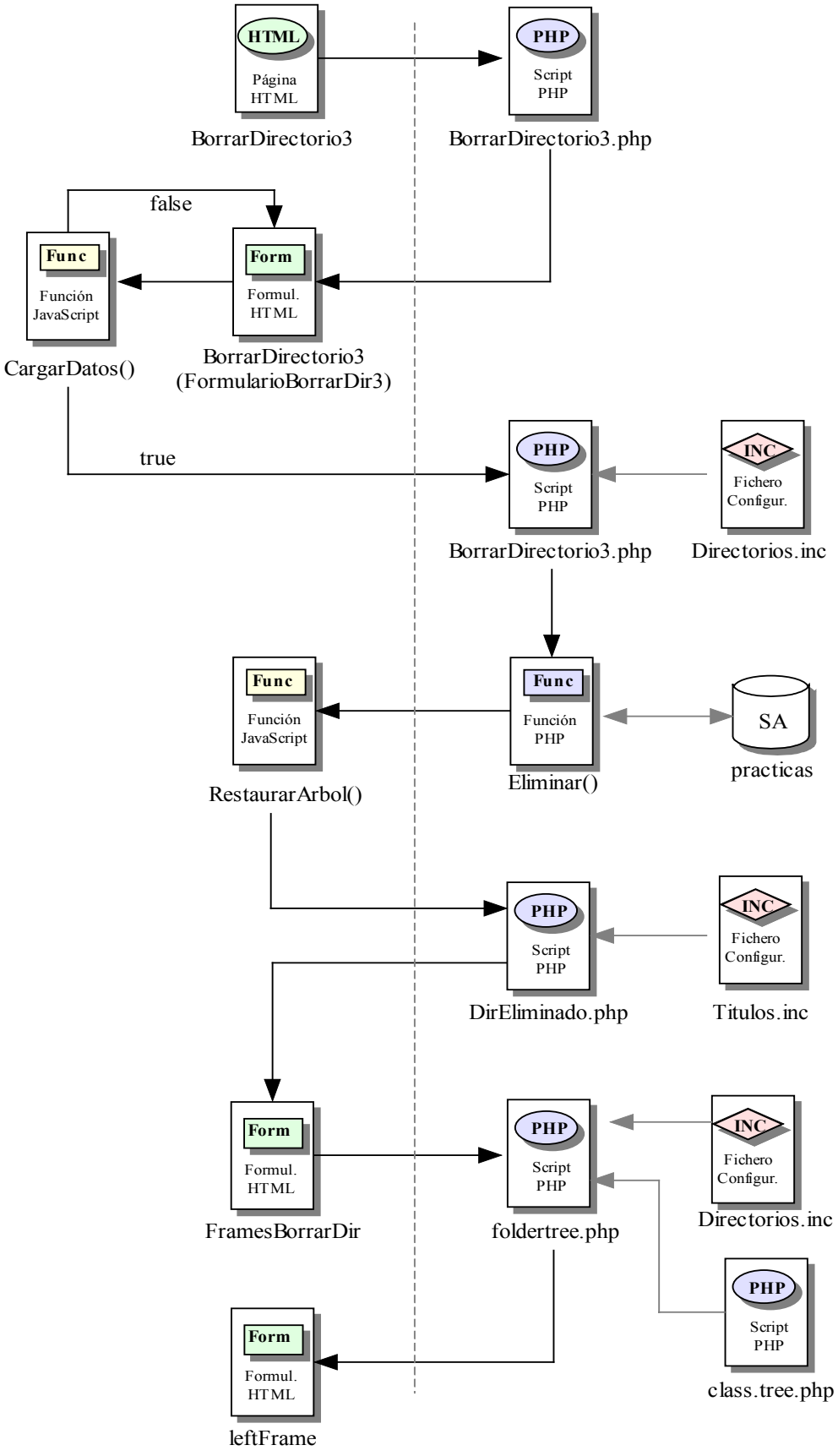


Figura 5.36 D.R. del Módulo de Eliminar Directorios. Marco BorrarDirectorio3

5.3.7 Gestión de archivos

La finalidad de este Proyecto es que el usuario de la aplicación pueda compilar y ejecutar los programas que vaya desarrollando desde la propia herramienta. Para ello, se hace necesario que la aplicación proporcione una serie de opciones donde el usuario, no sólo pueda ir creando sus propios archivos, sino también editarlos y eliminarlos. Todos estos archivos que se van desarrollando son almacenados dentro de un directorio personal de trabajo, asignado a cada uno de los usuarios a la hora de darlos de alta en la aplicación por parte del administrador.

Las opciones de "Gestión de archivos" son solicitadas por el usuario desde el marco *bottonFrame* y son detalladas a continuación.

5.3.7.1 Módulo de crear archivos

Este módulo permite al usuario crear nuevos archivos. La aplicación ofrece para ello, un editor de texto donde se pueden crear ficheros de cualquier tipo, sin importar ni su contenido, ni su extensión, ni su tamaño.

El script *"PrincipalAdministrador.php"* ejecutado en el servidor, es el encargado de cargar en el cliente este editor de texto. Esto se realiza mediante la presentación de un formulario en el marco *mainFrame*, llamado *FormularioPrincipal*, que contiene cuatro campos. Uno de los campos tiene un valor constante que sólo muestra la palabra 'Fichero:'. Otro de los campos se llama *fichero* y es de sólo lectura. El tercer campo se llama *area* y corresponde con el cuadro de texto donde el usuario introduce el contenido del archivo. El último campo se llama *directorio* y es un campo oculto. Los valores de los campos, *fichero*, *area* y *directorio* dependen de las variables recibidas por el script. De estos tres valores, el usuario sólo puede editar el valor del campo *area*.

En el marco *bottonFrame*, el script *"MenuAdministrador.php"* carga en el cliente once vínculos. El primero de ellos, lleva el título de Nuevo y es el único que ordena la ejecución en el cliente de la función JavaScript *nuevo()*.

Esta función es definida en el fichero de inclusión *"Funciones.js"* y lo único que hace es cargar los valores de los campos *area*, *fichero* y *directorio*, pertenecientes al formulario *FormularioPrincipal*, a vacío.

Una vez ejecutada la función, el editor de texto presentado en el marco *mainFrame* estará preparado para que el usuario introduzca el texto del nuevo archivo.

En la figura 5.37 se muestra el diagrama de relaciones correspondiente al "Módulo de crear archivos".

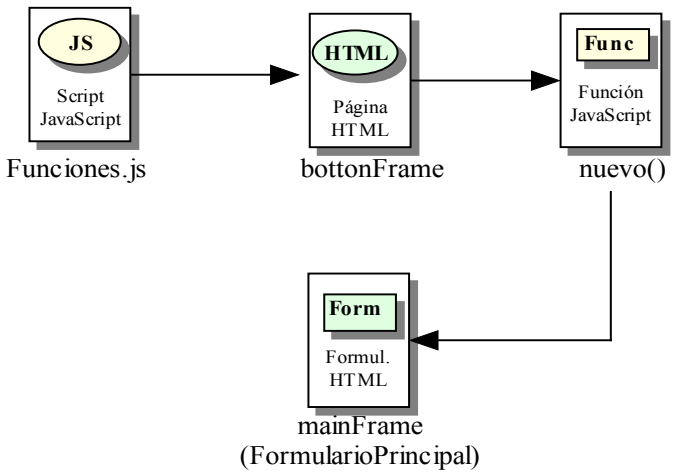


Figura 5.37 D.R. del Módulo de Crear Archivos

5.3.7.2 Módulo de leer archivos

Todos los archivos creados por el usuario pueden ser leídos por él mismo, en cualquier momento desde la aplicación. De esta forma, el usuario va teniendo siempre constancia de su trabajo realizado. Sólo el usuario administrador tiene privilegios, no sólo para leer los archivos realizados por él, sino también para leer los ficheros realizados por el resto de usuarios.

Existen dos marcos que presentan los vínculos que permiten la lectura de archivos, el marco *bottonFrame* y el marco *leftFrame*.

De los vínculos presentados por el script *"MenuAdministrador.php"*, sólo uno llama a la función JavaScript *abrir()*. Esta función se define en el fichero de inclusión *"Funciones.js"* y lo primero que hace es cargar, en una variable llamada *ifield*, el formulario *FormularioPrincipal* presentado en el marco *mainFrame*. Después abre una nueva ventana de navegación en el cliente mediante la función JavaScript *window.open()*. El script mandado a ejecutar en el servidor con la apertura de la nueva ventana es *"FramesAbrir.php"*. Por último, *abrir()* carga la variable *ifield* en la nueva ventana.

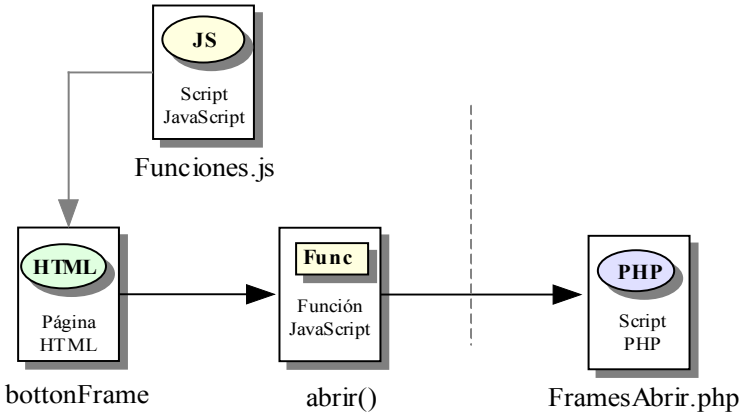


Figura 5.38 D.R. de la llamada al Módulo de Leer Archivos

"FramesAbrir.php" carga la nueva página con tres marcos, *marco01*, *marco02* y *marco03*, situados en la zona superior, media e inferior de la página respectivamente. A su vez, cada uno de los marcos ordena la ejecución de nuevos scripts.

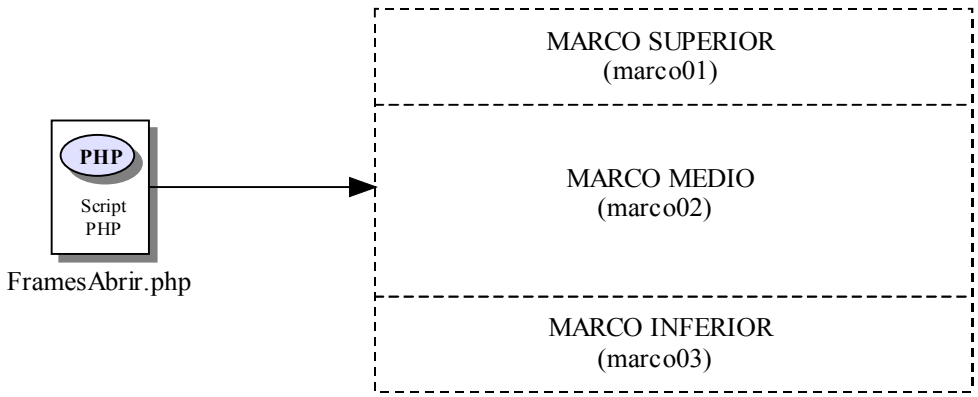


Figura 5.39 D.R. del Módulo de Leer Archivos. Petición de marcos

El marco *marco01* ordena la ejecución en el servidor del script *"Abrir1.php"*. Este script carga un formulario llamado *FormularioMarco01*. El formulario consta de un sólo campo llamado *ruta* y que es de sólo lectura.

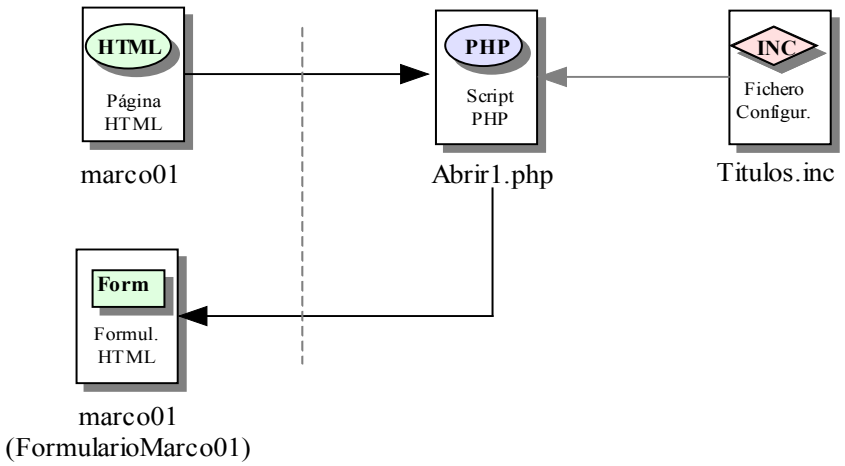


Figura 5.40 D.R. del Módulo de Leer Archivos. Marco marco01

El marco *marco03* únicamente presenta un vínculo que ejecuta la función propia del lenguaje JavaScript, *top.close()*, que obliga el cierre de la ventana.

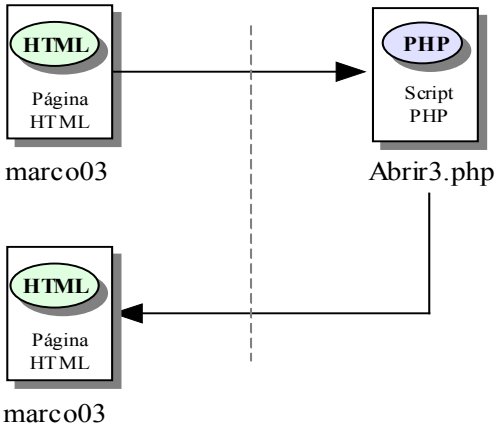


Figura 5.41 D.R. del Módulo de Leer Archivos. Marco marco03

El marco *marco02* ordena la ejecución en el servidor del script *"Abrir2.php"*, pasándole dos parámetros, *dir* y *d*. La variable *dir* toma el valor de la ruta del directorio de trabajo del usuario registrado en la sesión. La variable *d* toma el valor '1'.

"Abrir2.php" carga el formulario *FormularioPrincipal* y ordena la ejecución en el cliente de la función JavaScript *ActualizarDirectorio()*. Esta función carga el campo *ruta* del formulario *FormularioMarco01*, presentado en el *marco01*, con el valor que recibe el script *"Abrir2.php"* en la variable *dir*.

Después, el script pregunta por la variable *d*, que solo puede tomar dos valores, '1' en caso de directorio, y '0' en caso de archivo. Si la variable vale '1' el script manda a ejecutar la función PHP *Mostrar()*.

Mostrar() recibe dos parámetros de entrada. El primero es el valor de la variable *dir* y el segundo es la ruta completa del directorio de trabajo del usuario. Esta función, mediante un bucle, lee el contenido completo del directorio recibido a través de la variable *dir* y realiza, unas u otras instrucciones, dependiendo del tipo de elemento contenido en el directorio.

Si el elemento encontrado es un directorio, el script carga en el marco un vínculo que manda la ejecución del propio script *"Abrir2.php"* pasándole la variable *dir* con la ruta del directorio encontrado y la variable *d* con el valor '1'.

Si el elemento es un archivo, el script comprueba si se trata de un fichero ejecutable o no. Si el fichero es ejecutable, el script presenta el nombre del fichero sin vínculo ninguno. Si, por el contrario, es ejecutable, el script muestra un vínculo ordenando de nuevo la ejecución del script *"Abrir2.php"*, pero pasando esta vez cuatro variables. La variable *dir* ahora toma el valor del nombre del fichero leído, junto con su ruta completa, la variable *d* toma el valor '0', la variable *fich* toma el valor del nombre del fichero sin ruta, y la variable *di* toma el valor de la variable *dir* recibida en la función.

Sin embargo, si *"Abrir2.php"* recibe la variable *d* con valor '0' (caso de un archivo), el script llama a ejecutar a otra función PHP llamada *LeerFichero()*. Esta función se encuentra definida en el fichero de inclusión *"FuncionesFicheros.inc"*, y recibe por parámetro las variables *di* y *fich*. La función lee el contenido del archivo cuyo nombre sea el recibido en la variable *fich* y cuya ruta coincida con la recibida en la variable *di* y devuelve el resultado en una variable llamada *texto*. Por último, el script ordena la ejecución de una nueva función PHP, *CerrarVentana()*.

CerrarVentana() carga la variable *ifield*, la cual contiene el mismo formulario que el marco *mainFrame*, con los siguientes valores. El campo *fichero* lo carga con el valor de la variable *fich*, el campo *directorio* lo carga con el valor de la variable *di* y, por último, la variable *area* la carga con el valor de la variable *texto*. Luego, *CerrarVentana()* ejecuta la función propia de JavaScript *top.close()*, obligando el cierre de la ventana.

Cuando la nueva ventana de navegación, producida por la ejecución de la función JavaScript *abrir()*, es cerrada, el marco *mainFrame* carga en el cliente el formulario *FormularioPrincipal* con los nuevos valores cargados. Es decir, aparecen en el cliente los datos del fichero seleccionado en el vínculo presentado en el *marco02*.

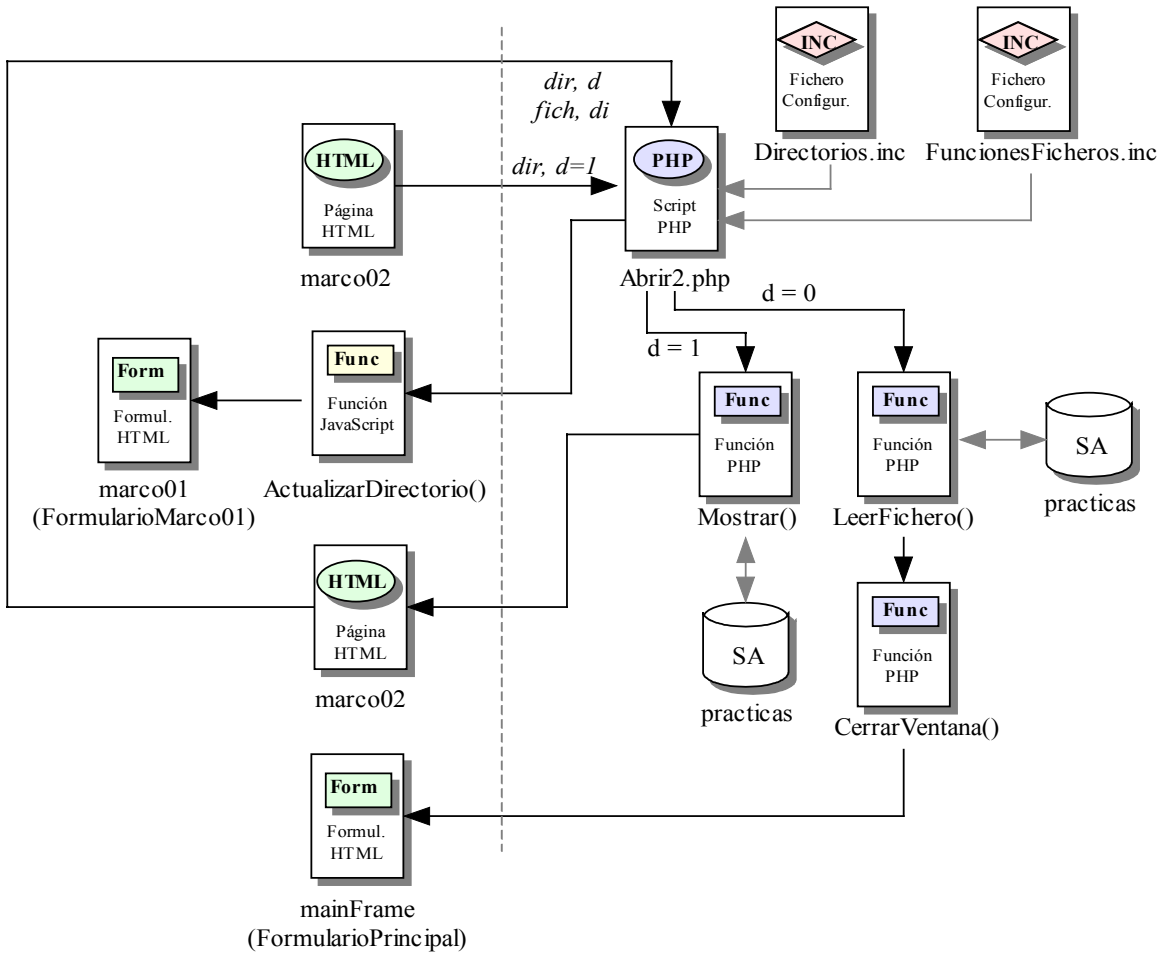


Figura 5.42 D.R. del Módulo de Leer Archivos. Marco marco02

El "Módulo de leer archivos" también es accesible por el usuario desde el marco *leftFrame*. El script *"foldertree.php"* carga en este marco, una página que muestra al usuario el contenido de su carpeta *practicas*, en forma de árbol jerárquico. El applet encargado de representar dicho árbol es detallado en el anexo de la presenta memoria. Por su parte, *"foldertree.php"* llama a la función *CrearArbol()*, la cual lee el contenido del directorio, consiguiendo de esta forma que la página presente todos los ficheros creados por el propio usuario. La función añade además, un vínculo por cada uno de estos archivos. Cada vínculo ordena la ejecución en el servidor del script *"PrincipalAdministrador.php"* pasándole tres parámetros, *opcion* con el valor 'arbol', *directorio* con el valor del directorio del archivo, y *fichero* con el nombre del fichero a abrir. Los ficheros ejecutables no presentan vínculo.

El script *"PrincipalAdministrador.php"* detecta que se proviene del script *"foldertree.php"* gracias a la variable *opcion* y llama a la función *LeerFichero()* definida en el archivo de inclusión *"FuncionesFicheros.inc"*. Esta función almacena el contenido del archivo, especificado en la variable *fichero* y ubicado en la ruta indicada por *directorio*, en la variable *area*. Después, el script carga el formulario *FormularioPrincipal* con los valores descritos, presentando en el editor de texto de la aplicación, el archivo seleccionado en el marco *leftFrame*.

La figura 5.43 muestra el diagrama de relaciones correspondiente al "Módulo de leer archivos" desde el marco *leftFrame*.

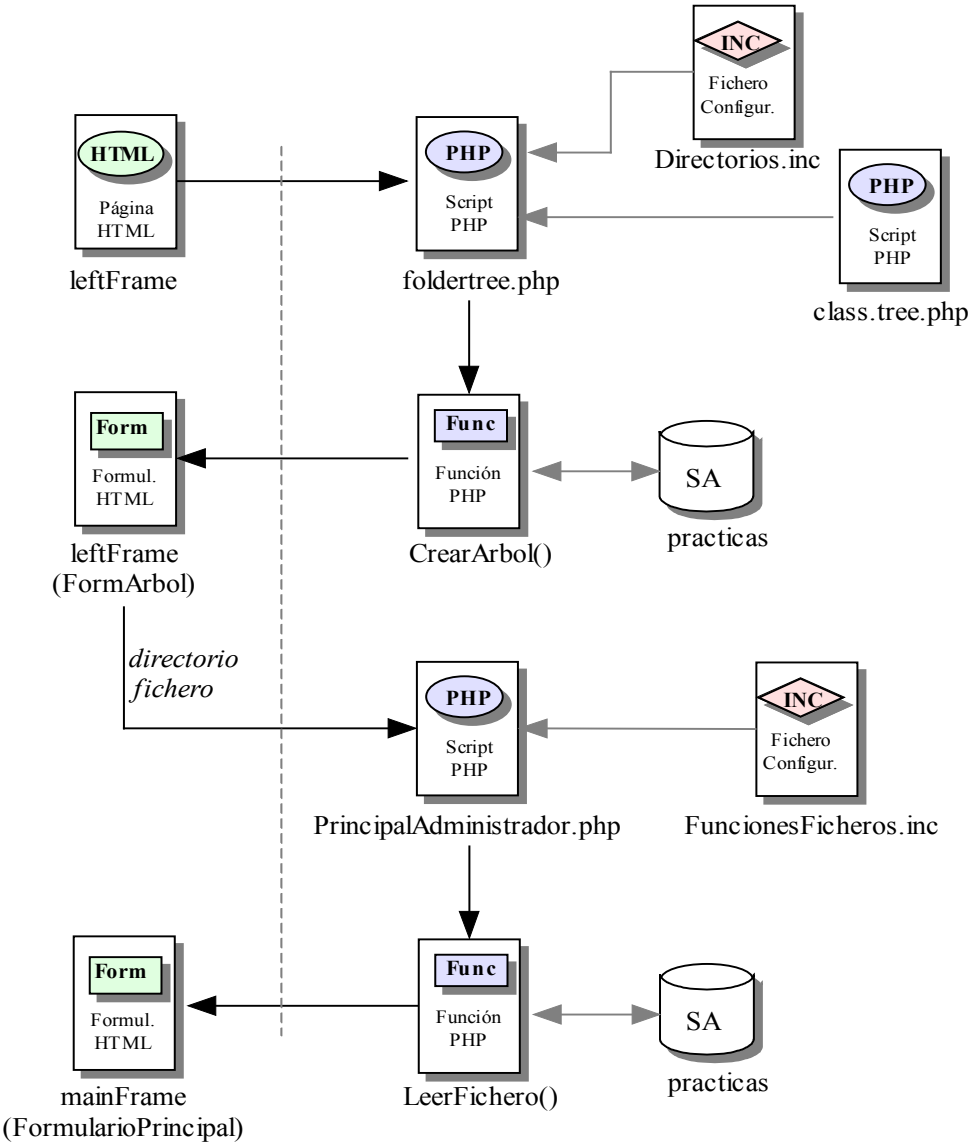


Figura 5.43 D.R. del Módulo Leer Archivos desde el marco leftFrame

5.3.7.3 Módulo de guardar

El "Módulo de guardar" se encarga de salvar todos los cambios producidos en un fichero, previamente abierto en el editor de texto ofrecido por la aplicación. Hay que resaltar que en este módulo no se crean archivos nuevos, sólo se modifican los ya existentes.

Aunque en apartados anteriores se ha nombrado con bastante frecuencia el script *"MenuAdministrador.php"*, nunca se había mencionado que este script carga un formulario llamado *FormularioMenu* que contiene cuatro campos, tres de ellos comunes con el formulario *FormularioPrincipal* presentado en el marco *mainFrame*. Estos campos comunes son *fichero*, *directorio* y *area*. El otro campo de *FormularioMenu* se llama *opcion*. Los cuatros campos son ocultos para el usuario.

Uno de los vínculos presentados por el script *"MenuAdministrador.php"* en el marco *bottonFrame*, realiza la llamada a la función JavaScript *guardar()*, definida en el fichero *"Funciones.js"*.

Guardar() carga los valores de los campos *fichero*, *directorio* y *area* del formulario *FormularioPrincipal* en los campos *fichero*, *directorio* y *area* del formulario *FormularioMenu*. Si el valor de *fichero* del formulario *FormularioPrincipal* esta vacío, la función llama a otra función JavaScript llamada *guardarcomo()*, la cual se detalla en el apartado siguiente de la presente memoria. Si no está vacío, entonces carga el campo *opcion* de *FormularioMenu* con el valor 'guardar' y ordena la ejecución en el servidor del script *"PrincipalAdministrador.php"* con el envío del formulario.

Este script llama a la función *GuardarFichero()* mediante la inclusión del archivo *"Funcionesficheros.inc"*. La función se encarga de guardar en un fichero, cuyo nombre coincide con el valor de la variable *fichero*, y ubicado en la ruta indicada en la variable *directorio*, el contenido indicado en la variable *area*, quedando de esta forma, el fichero abierto en el editor de texto de la aplicación salvado con el texto contenido en él.

Después de llamar a la función *GuardarFichero()*, *"PrincipalAdministrador.php"* carga el formulario *FormularioPrincipal*, presentándose en el marco *mainFrame* los datos del fichero guardado.

La figura 5.44 muestra el diagrama de relaciones correspondiente al "Módulo de guardar".

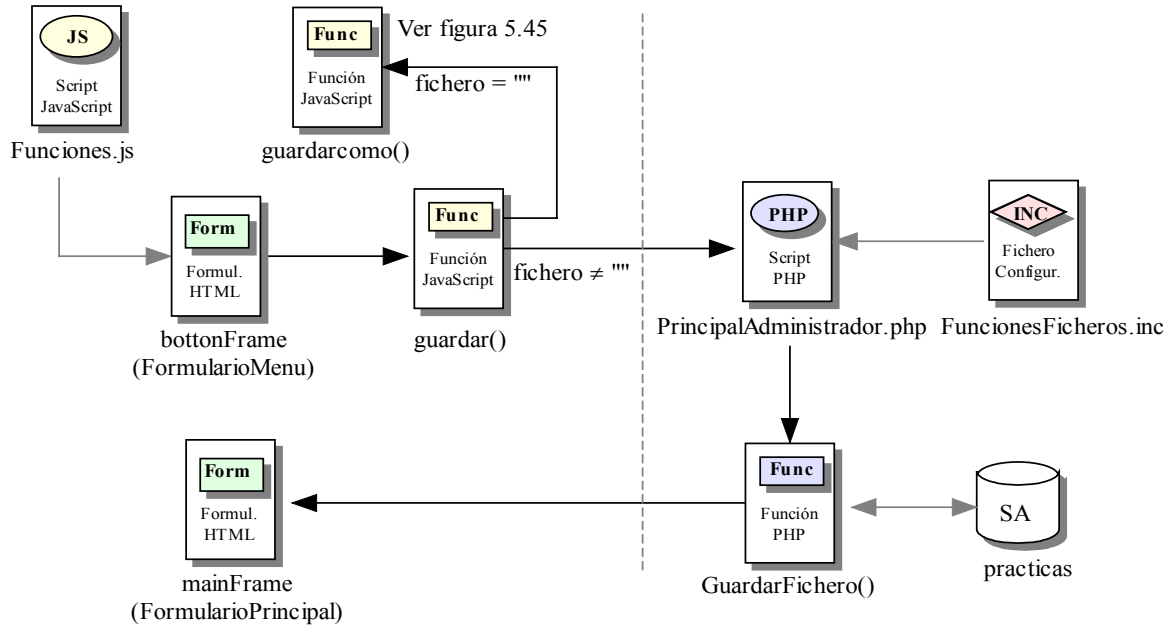


Figura 5.44 D.R. del Módulo de Guardar

5.3.7.4 Módulo de guardar como

A diferencia de la opción Guardar, cada vez que se ejecuta la opción de Guardar como, se crea un archivo nuevo dentro del directorio personal de trabajo del usuario que ha solicitado la operación. Una vez creado el archivo, el usuario puede leer su contenido dentro del editor de texto que presenta la aplicación.

Existe un vínculo, mostrado por el script *"MenuAdministrador.php"* en el marco *bottonFrame*, que ordena la ejecución en el cliente de la función JavaScript *guardarcomo()* definida en el fichero *"Funciones.js"*.

Al igual que la función *guardar()*, *guardarcomo()* carga con los valores de los campos *fichero*, *directorio* y *area* del formulario *FormularioPrincipal*, los campos *fichero*, *directorio* y *area* del formulario *FormularioMenu*. La función finaliza cargando el campo *opcion* con el valor 'guardarcomo' y enviando el formulario con la ejecución en el servidor del script *"PrincipalAdministrador.php"*.

Este script, mediante la inclusión del archivo *"FuncionesFicheros.inc"* llama a la función PHP *GuardarFichero()*. La función guarda el texto contenido en la variable *area*, recibida por el envío del formulario *FormularioMenu*, en un archivo temporal. Este archivo viene impuesto por el sistema y se llama *"fichero_temporal.txt"*. Se crea en el directorio *temporal*, ubicado en la carpeta personal del usuario registrado en la sesión.

Después de *GuardarFichero()*, el script *"PrincipalAdministrador.php"* llama a la función JavaScript *AbrirVentanaGuardarComo()*. Esta función carga en una variable llamada *ifield*, el formulario *FormularioPrincipal*. Luego, llama a la función JavaScript *window.open()* para la apertura de una nueva ventana de navegación. Después carga la variable *ifield* a la nueva ventana, provocando que dicha ventana conste del formulario. El script solicitado en la apertura de la nueva ventana es *"FramesGuardarComo.php"*.

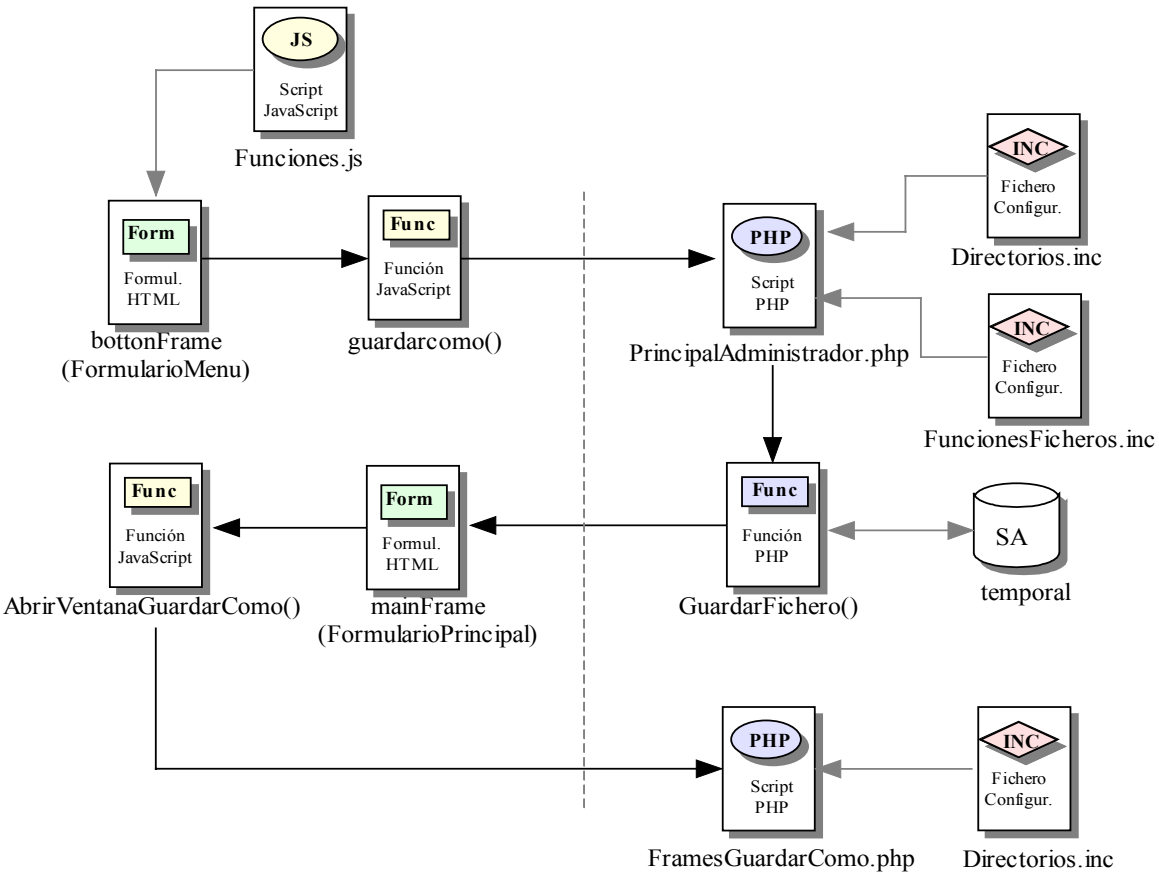


Figura 5.45 D.R. del Módulo de Guardar Como. Solicitud de nueva ventana

"FramesGuardarComo.php" carga en el cliente una página con tres marcos, *GuardarComo1*, *GuardarComo2* y *GuardarComo3*, situados en la parte superior, media e inferior de la página respectivamente. Cada uno de estos marcos ordena la ejecución de nuevos scripts.

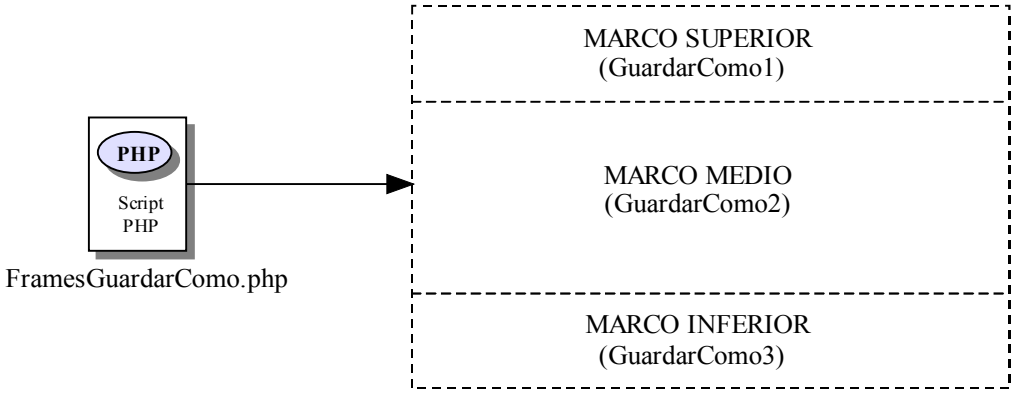


Figura 5.46 D.R. del Módulo de Guardar Como. Petición de marcos

El marco *GuardarComo1* ordena la ejecución del script *"GuardarComo1.php"*, pasándole un parámetro llamado *dir* con la ruta del directorio *practicas* perteneciente al usuario registrado en la sesión. El script carga el formulario *FormularioGC1* con un campo de sólo lectura llamado *ruta* y que toma el valor de la variable *dir* recibida por el script.

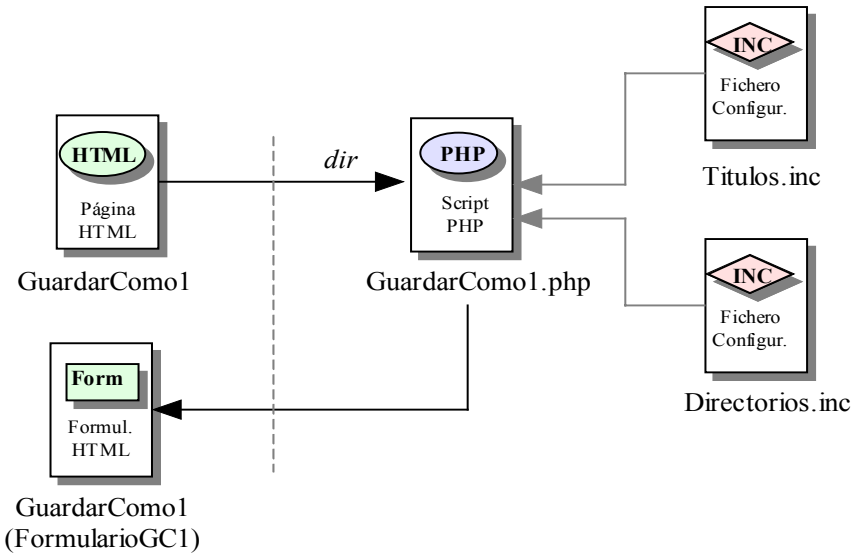


Figura 5.47 D.R. del Módulo de Guardar Como. Marco GuardarComo1

El marco *GuardarComo2* ordena la ejecución del script *"GuardarComo2.php"*. Este marco, al igual que *GuardarComo1*, le pasa al script una variable llamada *dir* con la ruta del directorio de trabajo del usuario y otra variable llamada *d* con el valor '1'. Después, el script llama a la función JavaScript *ActualizarDirectorio()* que a su vez, recibe por parámetro la misma variable *dir*. La función carga la variable en el campo *ruta* del formulario *FormularioGC1*. Seguidamente, el script llama a la función PHP *Mostrar()*.

Mostrar() recibe dos parámetros de entrada. El primero es el valor de la variable *dir* y el segundo es la ruta completa del directorio de trabajo del usuario. Esta función, mediante un bucle, lee el contenido completo del directorio recibido a través de la variable *dir* y realiza, unas u otras instrucciones, dependiendo del tipo de elemento contenido en el directorio.

En caso de ser un directorio el elemento, el script muestra en el cliente un vínculo que vuelve a ordenar la ejecución de *"GuardarComo2.php"* con los dos parámetros. La variable *dir* toma el valor del directorio encontrado y la variable llamada *d* que toma el valor '1'. Si el elemento es un archivo, el script carga en el marco el nombre del fichero sin ningún vínculo relacionado.

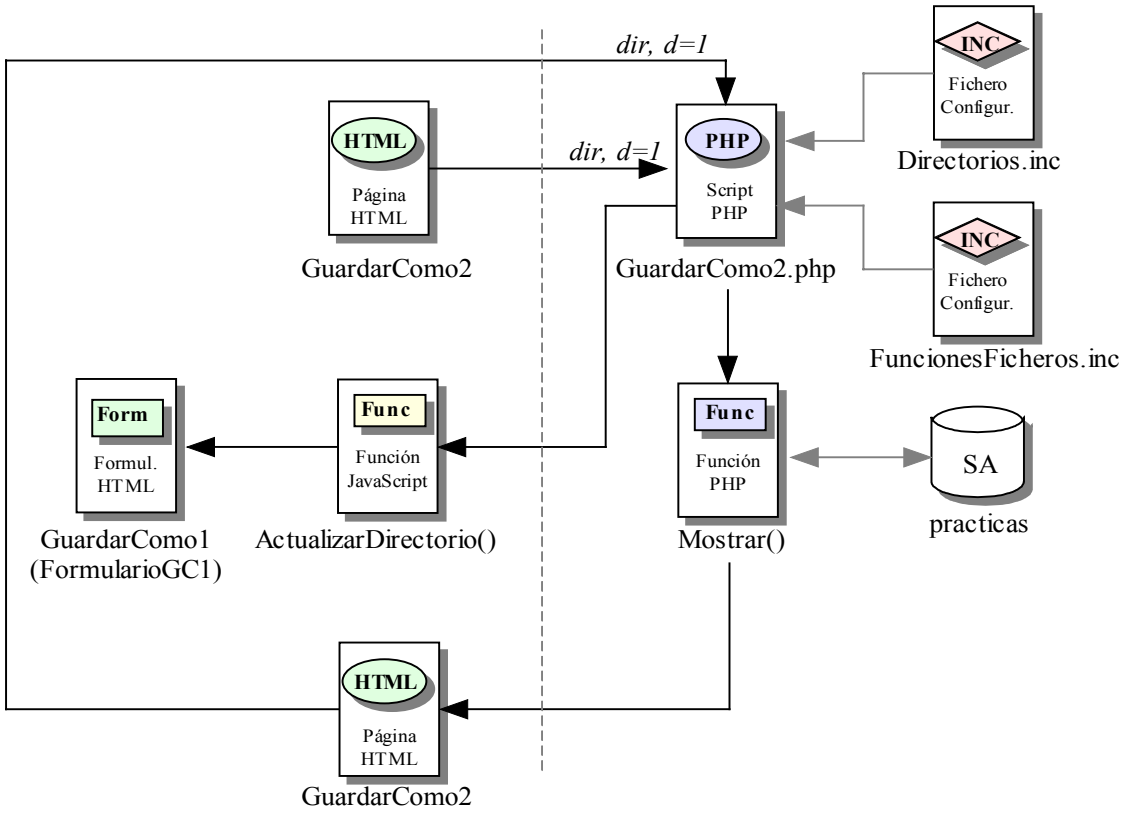


Figura 5.48 D.R. del Módulo de Guardar Como. Marco GuardarComo2

El marco *GuardarComo3* ordena la ejecución del script *"GuardarComo3.php"* en el servidor. El script llama a la función PHP *Formulario()* y ésta carga un nuevo formulario llamado *FormularioGC3*. Este formulario consta de tres campos, *rutadir*, *rutapadre* y *fiche*. Los dos primeros son campos ocultos y el segundo de ellos toma el valor de la ruta del directorio de trabajo del usuario. Además, el formulario presenta otro campo llamado *guardarcomo* que representa el botón de envío.

El envío del formulario ordena la ejecución en el cliente de la función JavaScript *ActualizarDirectorio()*. Ésta carga el campo *guardarcomo* con el valor 'guadarcomo' y el campo *rutadir* con el valor que tiene el campo *ruta* del formulario *FormularioGC1* en el marco *GuardarComo1*. El otro campo, *rutapadre*, almacena la ruta raíz del directorio de trabajo del usuario, para que, junto a *ruta*, tener la ubicación completa del directorio especificado.

Después de la función, se vuelve a ordenar la ejecución de *"GuardarComo3.php"*. Esta vez el script reconoce que se ha realizado el envío del formulario y pasa a comprobar que el valor recibido en el campo *fiche* no contenga ningún carácter especial. Si contiene algún carácter especial, el script ordena llamar a la función JavaScript *Caracteres()*. Esta función alerta al usuario del error producido. Posteriormente, el script llama a la función PHP *Formulario()*.

Si no existe ningún carácter especial en la variable *fiche*, el script comprueba que no existe ningún archivo llamado con el valor de *fiche* y ubicado en el directorio especificado en *rutadir*. En caso de existir, *"GuardarComo3.php"* pasa a llamar a la función JavaScript *FicheroExiste()*. Esta función alerta al usuario de que el fichero existe y el script vuelve a ordenar la ejecución de *Formulario()*. Si, por el contrario, el fichero no existe, el script, que incluye el archivo *"FuncionesFicheros.inc"*, realiza la llamada a dos funciones, *LeerFichero()* y *GuardarFichero()*. La primera de ellas, devuelve en una variable llamada *texto* el contenido del archivo temporal *"fichero_temporal.txt"* ubicado en la carpeta *temporal* dentro del directorio personal del usuario. *GuardarFichero()* guarda el texto, contenido en la variable *texto*, en el archivo *fiche* ubicado en *rutadir*, quedando el nuevo archivo creado. Por último, el script llama a al función JavaScript *CerrarVentana()*.

CerrarVentana() une las variables *rutapadre* y *ruta* para calcular la ubicación completa del directorio especificado. Luego, carga esta unión en el campo *directorio* y la variable *fiche* en el campo *fichero*, pertenecientes a *ifield*. Por último, obliga el cierre de la ventana, quedando en el cliente, los cuatro marcos del bloque de trabajo. Teniendo en cuenta que *ifield* corresponde con *FormularioPrincipal*, el marco *mainFrame* muestra en este formulario los nuevos datos almacenados correspondientes al nuevo archivo creado.

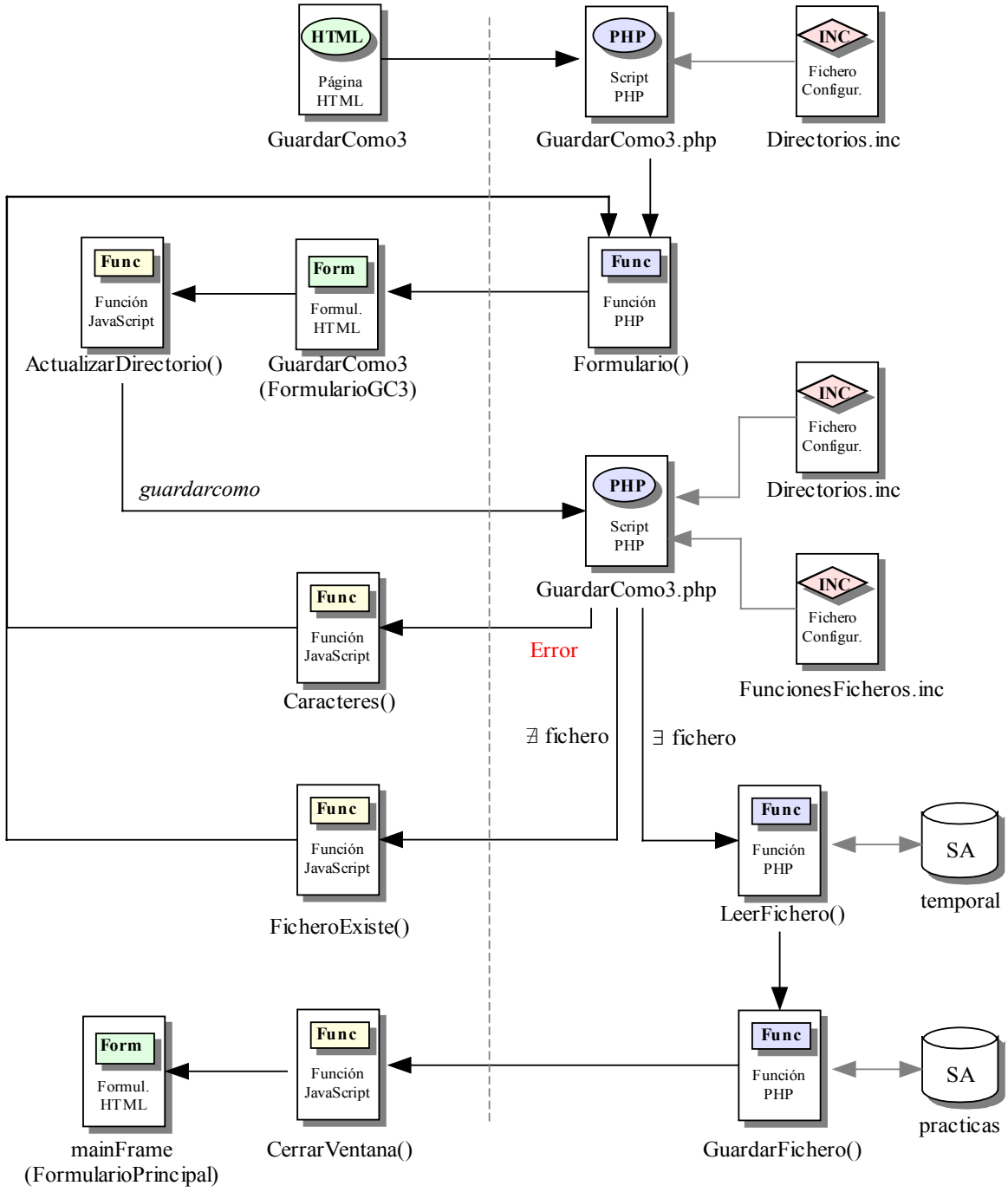


Figura 5.49 D.R. del Módulo de Guardar Como. Marco GuardarComo3

5.3.7.5 Módulo de eliminar archivos

Todos los archivos creados en el módulo anterior, pueden ser eliminados por el usuario. Para poder eliminarlo, el archivo debe estar previamente abierto en el editor de texto de la aplicación. Existe un detalle a tener en cuenta a la hora de borrar archivos y es

que cada vez que se elimine uno, si éste tiene un fichero ejecutable asociado a él, ambos son eliminados. En el "Módulo de compilación" se explica cómo se crean los ficheros ejecutables.

El quinto vínculo presentado en el marco *bottonFrame* por el script *"MenuAdministrador.php"*, llama a la función JavaScript *eliminar()* que se define en el fichero *"Funciones.js"*. Esta función carga en dos variables, *file* y *dire*, los valores de los campos *fichero* y *directorio*, respectivamente, del formulario *FormularioPrincipal* contenido en el marco *mainFrame*. Si *file* está vacío, la función muestra un mensaje de alerta indicando que no existe ningún archivo abierto en el editor de texto de la aplicación.

Si *file* es distinto de vacío, la función espera una confirmación, por parte del usuario, para continuar ejecutándose. Después de recibir esta confirmación, la función abre una nueva ventana de navegación ordenando la ejecución del script *"Eliminar.php"* y pasando por parámetro las variables *file* y *dire*. A continuación, *eliminar()* llama a la función JavaScript *nuevo()*, descrita en el "Módulo de crear archivos". Por último, la función vuelve a ordenar la ejecución del script *"foldertree.php"* en el marco *leftFrame*.

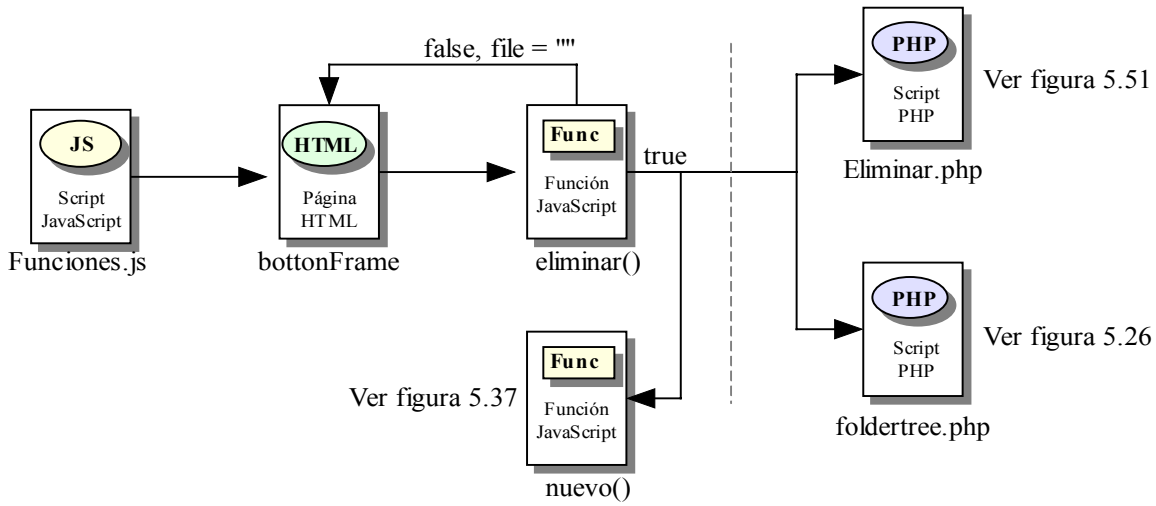


Figura 5.50 D.R. del Módulo de Eliminar Archivos

Mientras tanto, el script *"Eliminar.php"* incluye *"FuncionesFicheros.inc"* para poder ejecutar la función PHP *EliminarFichero()*. Esta función elimina el archivo con el nombre de *file* y ubicado en el directorio *dire*. Eliminado el fichero, el script comprueba si

existe el ejecutable, en cuyo caso, vuelve a ordenar la función *EliminarFichero()*, eliminando esta vez al fichero compilado. Después, y mediante la función *top.close()*, el script obliga el cierre de la ventana.

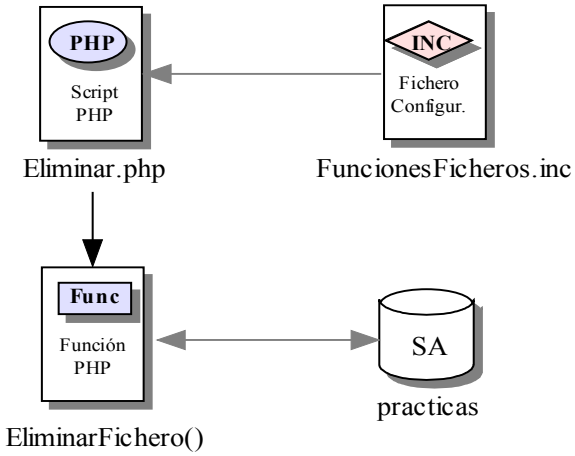


Figura 5.51 D.R. del Módulo de Eliminar Archivos. Eliminar.php

5.3.8 Compilación

En diferentes ocasiones, se menciona en la presente memoria, que el Proyecto tiene como objetivo ofrecer un entorno de trabajo donde el usuario de la aplicación pueda crear sus propios programas para posteriormente realizar sus ejecuciones. Pero para poder ejecutar un programa, éste debe compilarse, puesto que es el fichero ejecutable el que verdaderamente se manda a ejecutar.

5.3.8.1 Módulo de compilación

Otro de los vínculos presentado por el script *"MenuAdministrador.php"* en el marco *bottonFrame*, realiza una llamada a la función JavaScript *compilar()* definida en el fichero *"Funciones.js"*, que incluye el script.

Esta función, como otras anteriormente descritas, también carga en dos variables, *file* y *dire*, los valores de los campos *fichero* y *directorio* respectivamente, del formulario *FormularioPrincipal* presentado en el marco *mainFrame*. Hay que resaltar que estos dos

valores corresponden con el fichero abierto por el usuario en el editor de texto de la aplicación, y que por tanto, corresponden con los datos del fichero a compilar. Si *file* esta vacío, la función muestra al usuario un mensaje de alerta indicando que no tiene ningún archivo abierto en el editor.

Si *file* no esta vacío, la función ordena la apertura de una nueva ventana de navegación mediante la función JavaScript *window.open()*. El script mandado a ejecutar en el servidor con la apertura de la nueva ventana es *"FramesCompilar.php"* y recibe por parámetro dos variables, *fichero* y *directorio* que corresponden con las variables *file* y *dire* respectivamente.

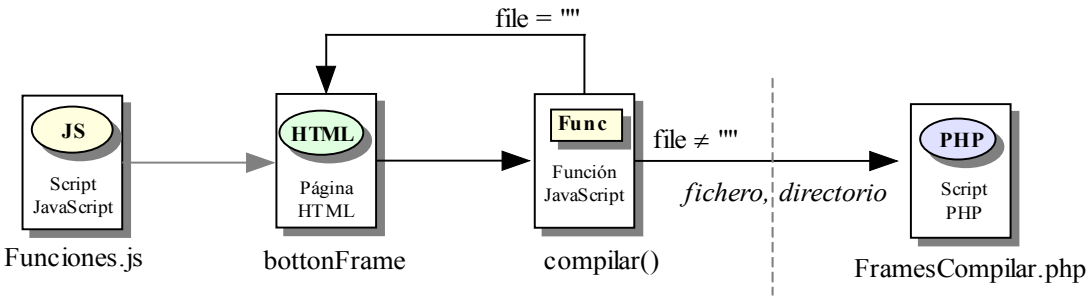


Figura 5.52 D.R. de la llamada al Módulo de Compilación

El script carga en la nueva ventana una página con tres marcos, *Compilar1*, *Compilar2* y *Compilar3*, situados en la parte superior, media e inferior respectivamente de la página. Cada uno de estos marcos, por su parte, ordena la ejecución de nuevos scripts.

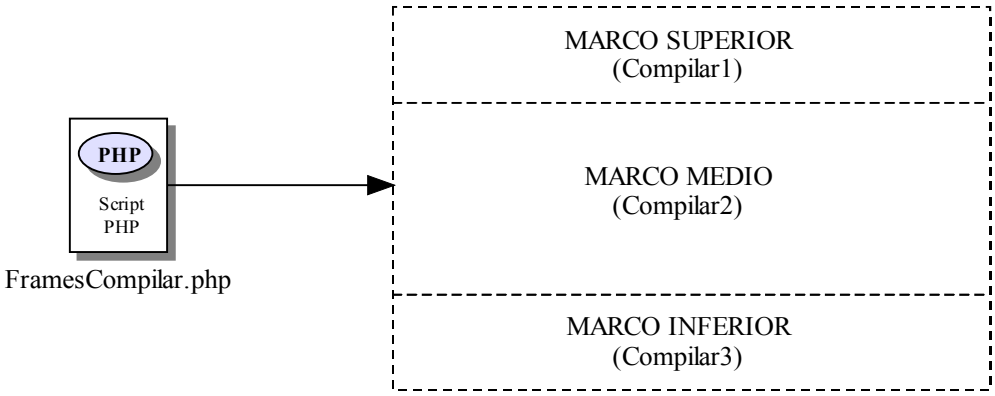


Figura 5.53 D.R del Módulo de Compilación. Petición de marcos

El marco *Compilar1* ordena la ejecución en el servidor del script *"Compilar1.php"*. Este script simplemente carga en el cliente la página con el título de compilación.

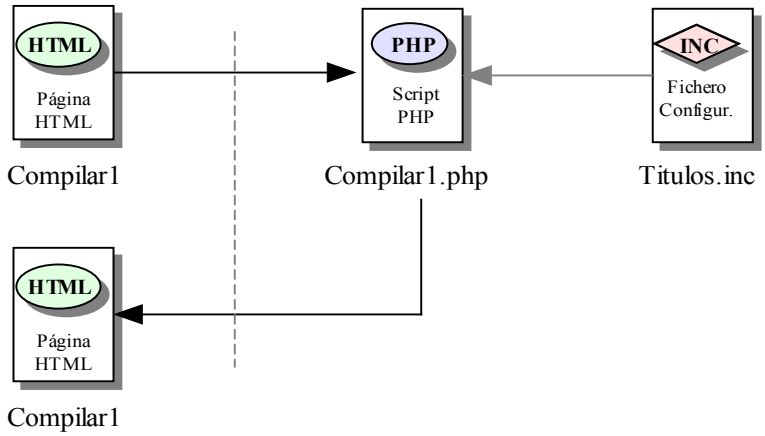


Figura 5.54 D.R. del Módulo de Compilación. Marco Compilar1

El marco *Compilar2* ordena la ejecución en el servidor del script *"Compilar2.php"*. Este script recibe dos parámetros de entrada, *fichero* y *directorio*. Estos dos valores corresponden con las variables *file* y *dire*, definidas en la función *compilar()*, es decir, corresponden con el nombre y ruta del fichero a compilar.

Lo primero que hace el script es separar la extensión del fichero. Según esta extensión, ordena un comando u otro para realizar la compilación del archivo en el propio servidor de la aplicación, y almacenar el resultado de la misma, en el archivo *"stderr.txt"* situado en la carpeta *procesos* dentro del directorio personal del usuario registrado en la sesión.

A continuación, el script lee este fichero y carga el resultado en la página, mostrando el resultado de la compilación en el cliente.

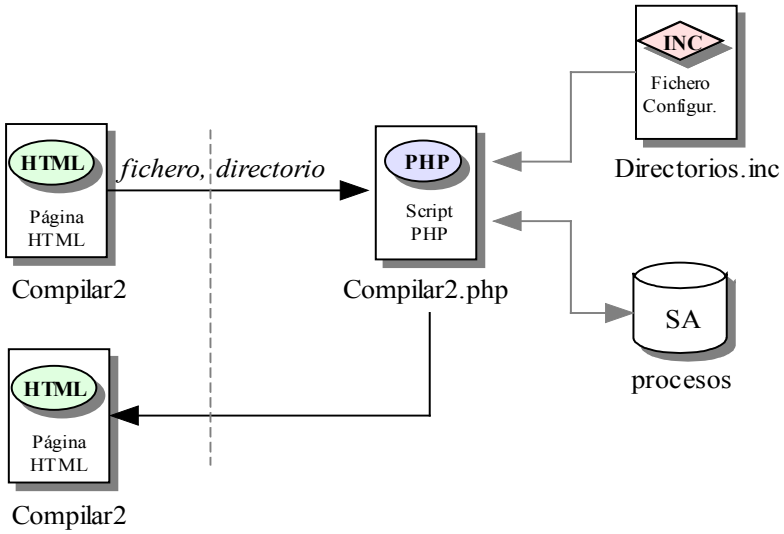


Figura 5.55 D.R. del Módulo de Compilación. Marco Compilar2

El marco *Compilar3* ordena la ejecución en el servidor del script *"Compilar3.php"*. Este script simplemente carga la página con un vínculo que efectúa una llamada a la función JavaScript *top.close()* y que ordena el cierre de la ventana.

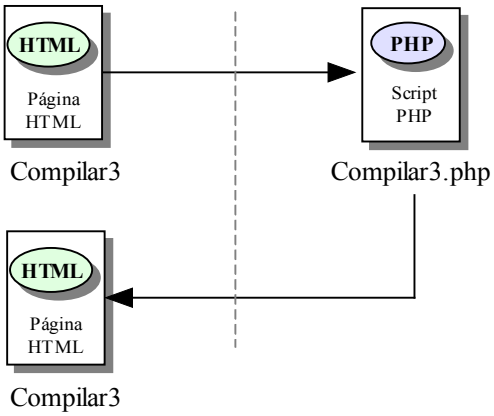


Figura 5.56 D.R. del Módulo de Compilación. Marco Compilar3

5.3.9 Ejecución

Después de compilado un programa, éste puede ser mandado a ejecutar. En la aplicación se hace necesario especificar un fichero de configuración antes de ejecutar el programa. En este fichero, el usuario puede especificar los datos necesarios para la correcta

ejecución del programa en el servidor, desde los parámetros necesarios, hasta los ficheros de entrada y salida, en caso de así requerirlo.

5.3.9.1 Módulo de ejecución

Quizás, de entre todos los vínculos mostrados en el marco *bottonFrame* por el script *"MenuAdministrador.php"*, el que ordena la ejecución de la función JavaScript *ejecutar()* sea el más relevante en la aplicación.

Esta función se define en el fichero *"Funciones.js"*, fichero que incluye el script. La función *ejecutar()*, al igual que otras funciones definidas en el mismo archivo, toma los valores de los campos *fichero* y *directorio* del formulario *FormularioPrincipal* mostrado en el marco *mainFrame*, y los almacena en dos variables, *file* y *dire* respectivamente. Después, comprueba si *file* está vacío, en cuyo caso, manda un mensaje de alerta al usuario informando que no tiene ningún fichero abierto en el editor de texto de la aplicación.

En caso contrario, la función ordena la apertura en el cliente de una nueva ventana de navegación mientras manda la ejecución del script *"FramesEjecutar.php"* en el servidor. La función le pasa a este script dos parámetros llamados *fichero* y *directorio*, con los valores de *file* y *dire* respectivamente.

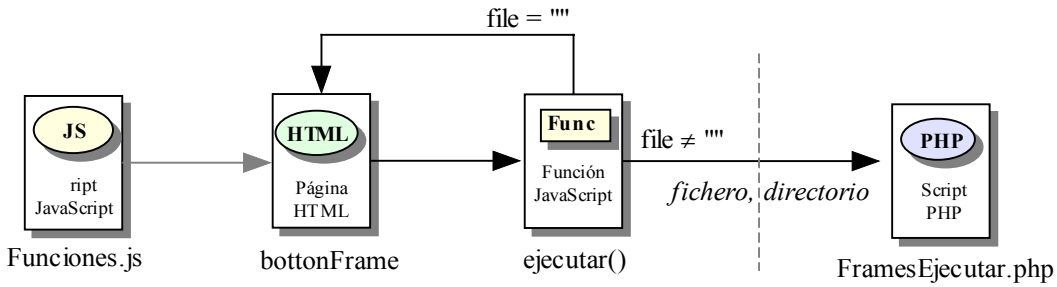


Figura 5.57 D.R. de la llamada al Módulo de Ejecución

Por su parte, el script carga una página en el cliente con tres marcos, *Ejecutar0*, *Ejecutar1* y *Ejecutar6*, situados en la parte superior, media e inferior de dicha página. Cada uno de estos marcos ordena la ejecución de nuevos scripts.

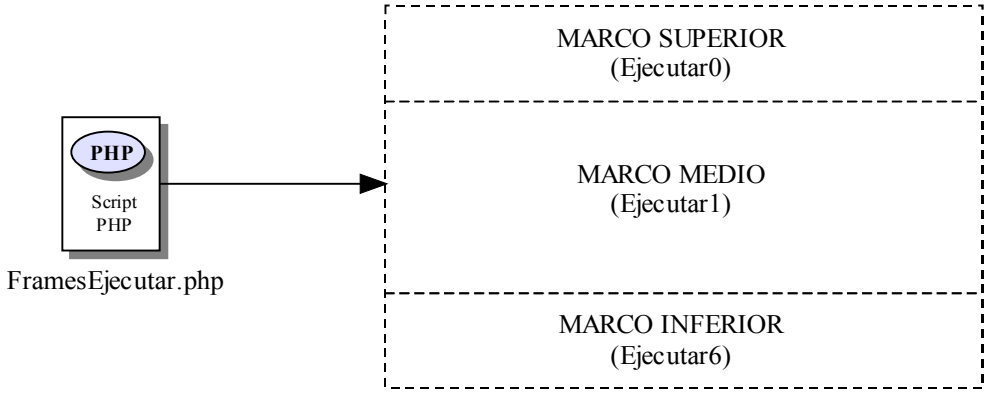


Figura 5.58 D.R. del Módulo de Ejecución. Petición de marcos

Ejecutar0 ordena la ejecución en el servidor del script *"Ejecutar0.php"*. El script carga una página mostrando el fichero a ejecutar mediante la variable *fichero* recibida. Además, carga un vínculo en el cliente mediante el cual, el usuario tiene la posibilidad de crear nuevos ficheros de configuración. Para ello, este enlace ordena la ejecución del script *"Ejecutar2.php"* pasando tres parámetros. El parámetro *fichero* que corresponde con la variable *fichero*, el parámetro *directorio* que corresponde con la variable *directorio* y un parámetro llamado *nuevo* con valor 'true'.

"Ejecutar2.php" carga una página en el marco *Ejecutar1* y comprueba, mediante las variables recibidas de *fichero* y *directorio*, si la extensión del fichero abierto en el editor de texto corresponde con la de un programa desarrollado bajo el lenguaje C o C++. En caso de no corresponder con ninguno de los lenguajes, el script almacena en una variable llamada *tipo* el valor '2'. En caso de que la extensión corresponda con '.c' o '.cpp', el script pasa a comprobar que el fichero ha sido compilado correctamente, corroborando que existe un fichero ejecutable asociado al fichero en cuestión. Si no existe el ejecutable, el script almacena en la variable *tipo* el valor '1'. Si el fichero puede ser mandado a ejecutar, el script almacena en *tipo* el valor '0'.

Lo segundo que hace *"Ejecutar2.php"* es comprobar el valor de *nuevo* y en caso de ser 'true' carga un formulario llamado *FormularioFichConf* con siete campos (*nombre*, *parámetros*, *entrada*, *salida*, *timeout*, *directorio* y *fichero*) y dos botones de envío (*guardar* y *ejecutar*). Los campos *directorio* y *fichero* son los únicos campos ocultos, no

modificables por el usuario, y que toman los valores de las variables *directorio* y *fichero* recibidas por el script. Por otro lado, ambos botones efectúan una llamada a la función JavaScript *Validar()* antes del envío del formulario.

Validar() toma dos parámetros de entrada, uno corresponde con los propios valores del formulario y el segundo corresponde con la variable *tipo*, siempre y cuando el botón seleccionado sea *ejecutar*, ya que, si es *guardar*, el segundo parámetro toma el valor '0' siempre. *Validar()* comienza estudiando este segundo parámetro. Si vale '1', la función muestra al usuario un mensaje de alerta indicándole que el fichero no ha sido compilado. Si vale '2', muestra otro mensaje de alerta indicando que el fichero no es ejecutable. Si por el contrario, el parámetro no toma ninguno de estos valores, la función comprueba si el campo *nombre* de *FormularioFichConf* esta vacío, en cuyo caso el mensaje de alerta mostrado indicaría al usuario que debe rellenar dicho campo. Si *nombre* no está vacío, *Validar()* comprueba que el valor del campo *timeout* es de tipo numérico, o por el contrario muestra un nuevo mensaje de alerta. Si por fin, la función comprueba que no existe ningún error y no muestra ninguno de los mensajes de alerta descritos, se realiza el envío del formulario y se ordena la ejecución en el servidor de un nuevo script, *"Ejecutar3.php"*.

"Ejecutar3.php" crea, en caso de no existir, y sobrescribe, en caso de existir, un archivo, con el nombre especificado en el campo *nombre* de *FormularioFichConf*, y en el directorio *configuración* perteneciente al directorio personal del usuario registrado en la sesión. El contenido de este fichero corresponde con los valores de los campos *parámetros*, *entrada*, *salida* y *timeout*. Después, el script distingue si el botón que fue seleccionado en *"Ejecutar2.php"* fue *guardar* o *ejecutar*. Si fue *guardar*, el script ordena el cierre de la ventana mediante la llamada a la función característica de JavaScript *top.close()*. Si el botón seleccionado fue *ejecutar*, el script, gracias al fichero de inclusión *"Directorios.inc"*, va creando el comando que se mandará a ejecutar en el servidor Linux de la aplicación.

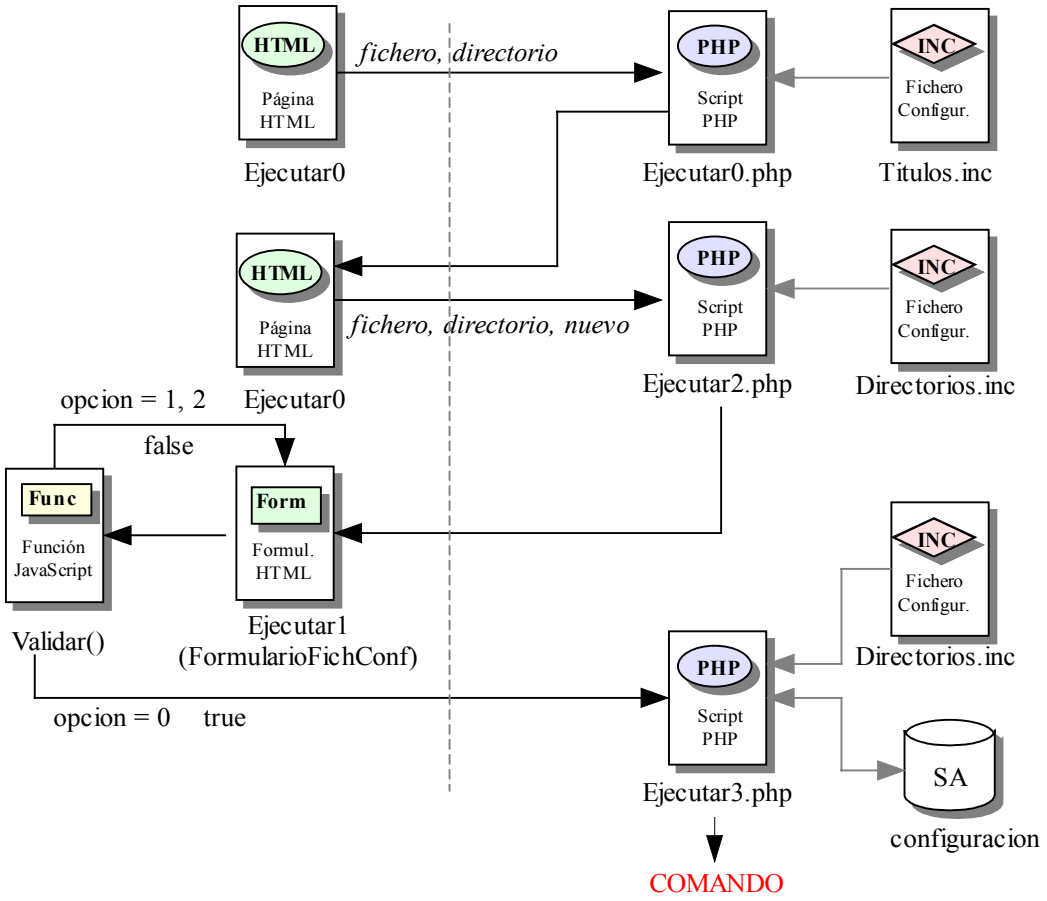


Figura 5.59 D.R. del Módulo de Ejecución. Marco Ejecutar0. Parte I

Para la creación del comando, el script tiene que estudiar, todos y cada uno, de los casos posibles. Estos casos son que no exista timeout y no exista fichero de entrada, que no exista timeout y sí fichero de entrada, que exista timeout y no exista fichero de entrada y que exista timeout y fichero de entrada. Los valores de timeout y fichero de entrada se corresponden con las variables recibidas por el script de *timeout* y *entrada*.

En cualquier caso, el comando resulta ser un conjunto de parámetros bajo un orden controlado, que el servidor debe interpretar de forma correcta para producir el efecto deseado. Este orden corresponde con el siguiente:

1º.- Script tipo bash.

2º.- Fichero que tomará el sistema como fichero estándar de salida.

3º.- Proceso padre.

4º.- Proceso hijo.

5º.- Fichero donde almacenar información sobre los procesos padre e hijo.

6º.- Directorio de trabajo del usuario.

7º.- Directorio de publicación de la aplicación.

8º.- Timeout (en caso de existir).

9º.- Fichero que tomará el sistema como fichero estándar de entrada (en caso de existir).

10º.- Listado de resto de parámetros.

A continuación se detallan cada uno de los puntos anteriores.

- ***Script tipo bash:*** Corresponde siempre con el script tipo bash *"miscript.sh"*, incluido en la instalación de la herramienta. Su función es muy básica, ya que lo que ordena es un cambio de directorio, pasando del actual al directorio *practicas* ubicado dentro de la carpeta personal del usuario registrado en la sesión. Después continua con la ejecución de todos los parámetros que recibe, y por último, el script vuelve a hacer un cambio de directorio volviendo a donde se ubica el script.

- ***Fichero estándar de salida:*** Corresponde con la variable *salida* recibida del script *"Ejecutar2.php"* mediante el formulario *FormularioFichConf*. Este parámetro va precedido por "&>". Por este motivo, el resultado de *"miscript.sh"* es almacenado en este fichero. El carácter "&" indica que el resultado sea almacenado en el archivo, incluso en el supuesto de cometerse error.

- ***Proceso padre:*** El proceso padre llamado, depende de los diferentes casos descritos anteriormente, pudiendo ser uno de estos cuatro: *padre*, *padre2*, *padre_timeout* y *padre_timeout2*. El caso más completo es el de *padre_timeout2*, que es el llamado en el caso de existir timeout y fichero de entrada. Lo primero que hace este proceso, es cerrar el fichero estándar de entrada del sistema para asignárselo al archivo especificado en la variable

entrada. El siguiente paso es crear un proceso hijo y escribir en el fichero de información de procesos una serie de datos acerca de los procesos padre e hijo. Este fichero se describe con detalle en dos puntos más adelante. El proceso hijo invoca la ejecución del programa mandado a ejecutar por el usuario, mediante la llamada exec. Después, el proceso padre crea un nuevo proceso hijo, el cual únicamente se queda "durmiendo" tanto tiempo como el especificado en la variable *timeout*. Por último, el proceso padre espera a que cualquiera de sus dos procesos hijos termine e inmediatamente después identifica cuál de los dos fue el que finalizó. Por último, el proceso padre vuelve a escribir en el fichero de información de procesos y mata al proceso hijo no finalizado. En caso de no existir timeout, el proceso padre sólo crea un proceso hijo y queda esperando únicamente por éste.

- ***Proceso hijo:*** Cualquiera de los cuatro procesos padre crea, al menos un proceso hijo. Este proceso hijo, invoca al programa mandado a ejecutar por el usuario, recibido por el script *"Ejecutar3.php"* mediante las variables *directorio* y *fichero* del formulario *FormularioFichConf* y con todos los parámetros especificados en la variable *parametros*. La ejecución del programa del usuario se realiza mediante la llamada que hace el proceso hijo a una de las funciones de la familia exec.

- ***Fichero de información de procesos:*** Corresponde con el archivo *"ejecucion.txt"* situado en el directorio *procesos*, dentro del directorio personal del usuario registrado en la sesión. En él se almacena, primeramente, el nombre del programa ordenado a ejecutar por el usuario junto a su ruta completa, el identificador del proceso que lanza este programa y que corresponde con el proceso hijo, el identificador del proceso padre que crea al proceso hijo descrito, la palabra "ON" indicando que el proceso hijo se está ejecutando, y la palabra "OK" indicando que no ha habido ningún error en la operación. Después de finalizar el proceso hijo, si éste coincide con el que ordenó la ejecución del programa del usuario, el proceso padre vuelve a escribir en el fichero, el programa del usuario ejecutado, el identificador del proceso hijo, el identificador del proceso padre, la palabra "OFF" que indica que el proceso hijo ha finalizado, y por último, el estado de finalización del proceso. En el caso de

ser el segundo proceso hijo el que primero finalice, el proceso padre escribe la misma información pero en vez del estado, escribe la palabra "timeout".

- ***Directorio de trabajo del usuario:*** Este directorio coincide con la carpeta *practicas* del directorio personal del usuario. Se incluye en la línea de comando porque es utilizado por *"miscript.sh"* para el cambio de directorios. La razón de que el script realice el cambio del directorio actual al directorio *practicas*, es por si el usuario trabaja en su programa con rutas, ya que el usuario de la aplicación nunca verá más allá de esa carpeta.

- ***Directorio de publicación de la aplicación:*** El último paso que realiza *"miscript.sh"* es pasar, del directorio *practicas* del usuario registrado, al directorio de publicación de las páginas pertenecientes a la aplicación, quedando la ubicación, tal cual estaba antes de la llamada a *"miscript.sh"*.

- ***Timeout:*** Tiempo en segundos, especificado por el usuario de la aplicación en el formulario *FormularioFichConf*. Este parámetro indica el tiempo que el proceso padre debe esperar por la finalización del programa del usuario. En caso de superarlo, el proceso padre obliga matar al proceso hijo, evitando que éste se quede colgado en el sistema.

- ***Fichero estándar de entrada:*** Este fichero corresponde con el indicado en la variable *entrada* recibida por *"Ejecutar3.php"*. En caso de no ser nula, el proceso padre obliga a que éste sea el fichero estándar de entrada del sistema de donde el programa del usuario leerá los datos de entrada.

- ***Parámetros:*** Corresponde con la variable *parámetros*, recibida en *"Ejecutar3.php"* por el envío de *FormularioFichConf*. Consiste en uno, o varios parámetros separados por un espacio en blanco entre sí y son los utilizados por el programa del usuario. Para ello, son enviados en la llamada a la función exec, dentro del proceso hijo descrito anteriormente.

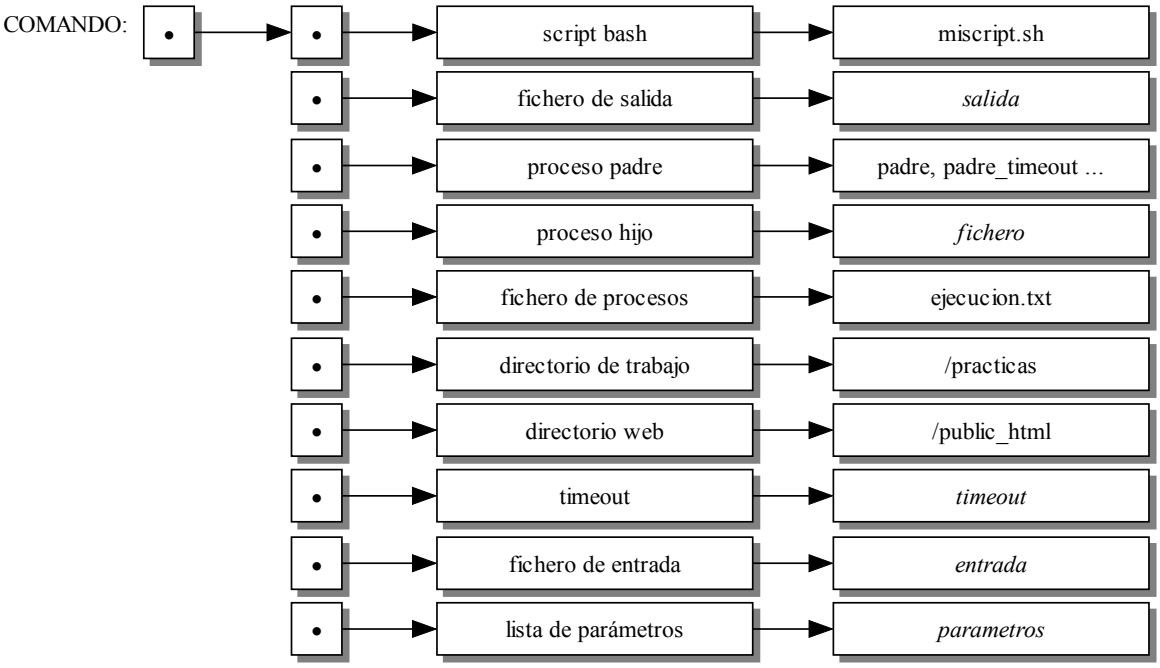


Figura 5.60 Construcción del comando

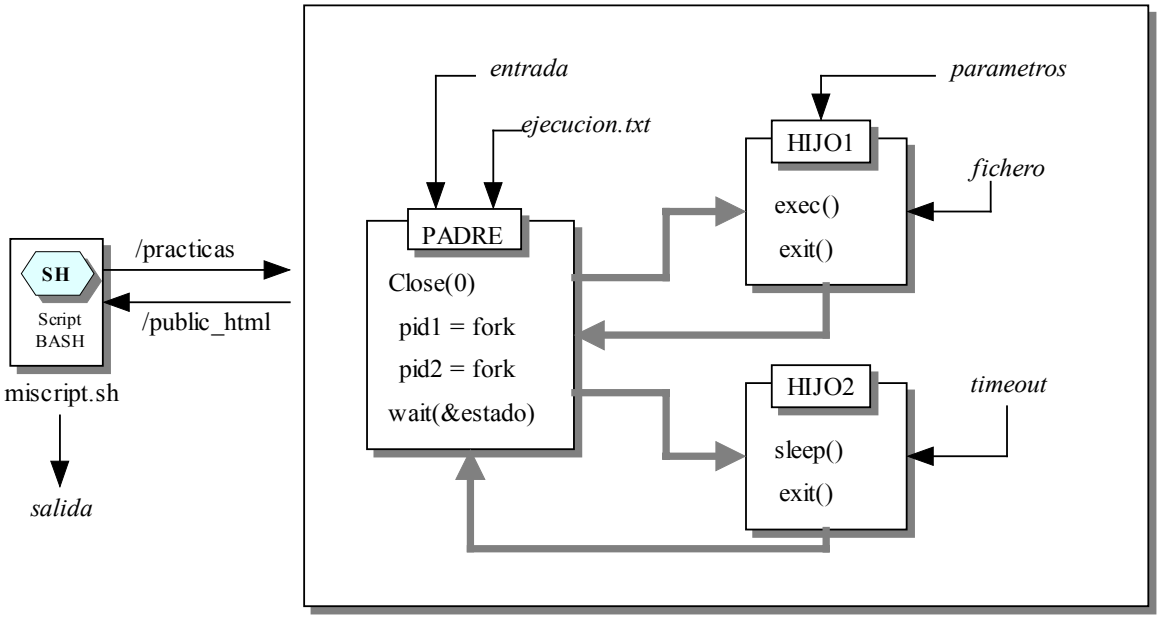


Figura 5.61 Ejecución del comando con padre_timeout2

"Ejecutar3.php" lanza el comando descrito, con la llamada a la función PHP *system*. Esta misma función devuelve una variable llamada *resultado* en caso de producirse algún error, de esta forma el script comprueba si se ha producido fallo. Si existe *resultado*, *"Ejecutar3.php"* carga en el cliente una página donde se informa al usuario que se ha producido error en la ejecución de su programa. En la misma página se muestran dos vínculos, uno provoca la apertura de una nueva página de navegación mostrando la ayuda de la aplicación, el otro vínculo vuelve a cargar la página anterior, *"Ejecutar2.php"*.

Si la función *system* no devuelve la variable *resultado*, indicaría que no ha habido ningún error y el programa del usuario es ejecutado en el servidor de la aplicación. Mientras esta ejecución dure, la ventana continuará abierta. Una vez finalice el proceso, la ventana se cerrará en el cliente mediante la función JavaScript *top.close()*.

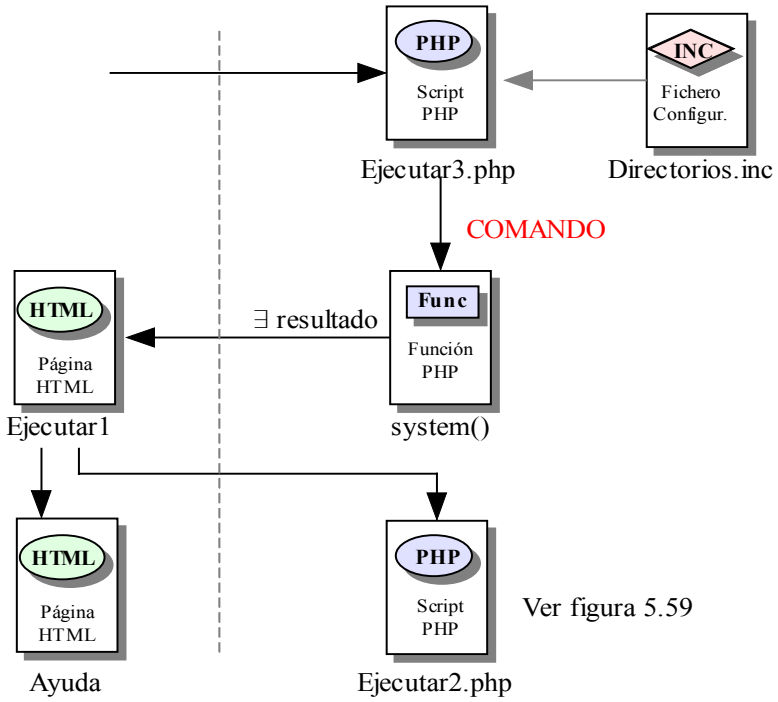


Figura 5.62 D.R. del Módulo de Ejecución. Marco Ejecutar0. Parte II

Por su parte, el marco *Ejecutar1* ordena la ejecución del script *"Ejecutar1.php"* en el servidor. El script recibe dos variables, *fichero* y *directorio*, que contienen el nombre y la ruta del archivo abierto en el editor de texto de la aplicación. De la misma forma que el script *"Ejecutar2.php"*, se comprueba si el fichero existe, pertenece al lenguaje C o C++ y

está compilado. Cuando el script estudia el caso, asigna a una variable llamada *tipo* el valor '2' si el fichero no es .c o .cpp, el valor '1' si el fichero no está compilado y el valor '0' si el fichero puede ser mandado a ejecutar. Después, *"Ejecutar1.php"* llama a la función PHP *MostrarDirConfiguracion()*. La función lee el contenido del directorio *configuracion* dentro de la carpeta personal del usuario. Este directorio sólo contiene los ficheros de configuración que va creando el propio usuario y la función muestra, por cada uno de estos ficheros, tres enlaces, *editar*, *ejecutar* y *eliminar*.

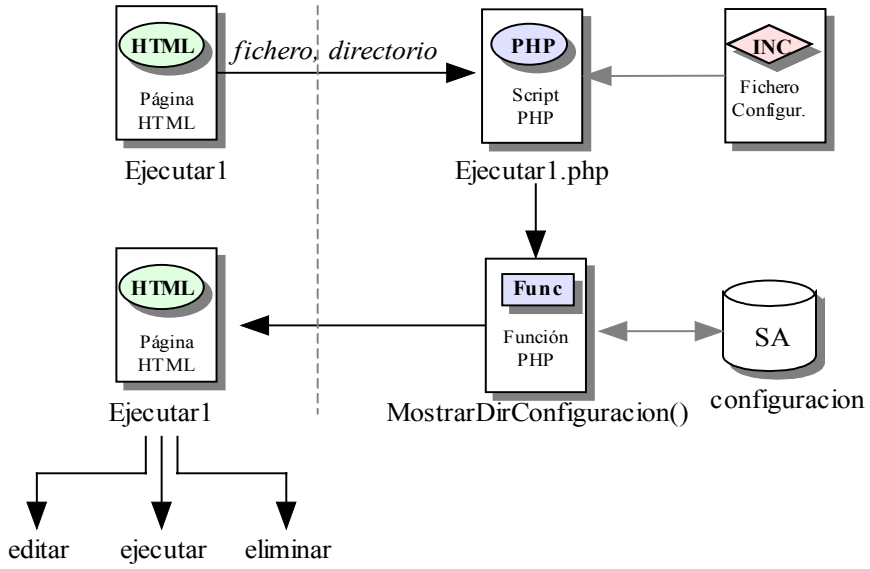


Figura 5.63 D.R. del Módulo de Ejecución. Marco Ejecutar1. Parte I

El primer enlace ordena la ejecución en el servidor del script *"Ejecutar2.php"* que ha sido nombrado anteriormente en este módulo. Pero en esta ocasión, *"Ejecutar2.php"* recibe tres parámetros, *fichero*, *directorio* y *fich_conf*. Los dos primeros corresponden con el nombre y ruta del fichero abierto en el editor de texto, y el tercero corresponde con el archivo de configuración seleccionado, correspondiente al directorio *configuracion* del usuario. Por tanto, *"Ejecutar2.php"* no recibe la variable *nuevo* con valor 'true' y realiza instrucciones diferentes a las descritas con anterioridad. En esta ocasión, el script lee el contenido del fichero de configuración sacando los valores de las variables *parametros*, *entrada*, *salida* y *timeout*. Luego, carga en el cliente un formulario, *FormularioFichConf*, con siete campos (*parametros*, *entrada*, *salida*, *timeout*, *fichero*, *directorio* y *nombre*). Los tres últimos campos son ocultos y los demás cargan, por defecto, el valor de las variables

anteriores. Estos campos sí son modificables por el usuario. Además, el formulario presenta dos botones de envío, *guardar* y *ejecutar*. Ambos botones llaman a la función JavaScript *Validar()* descrita en párrafos anteriores.

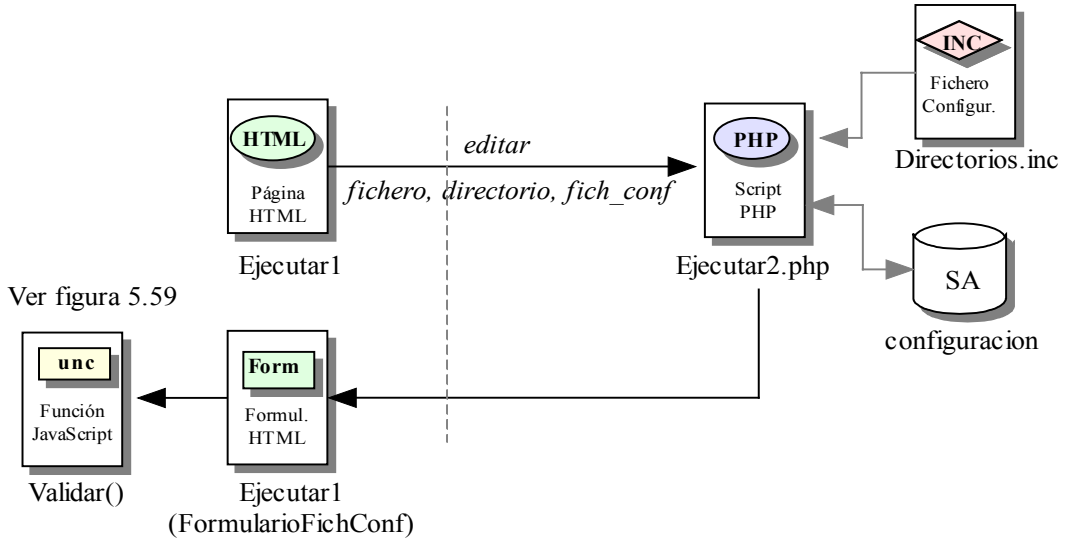


Figura 5.64 D.R. del Módulo de Ejecución. Marco Ejecutar1, opción Editar

El segundo de los enlaces, *ejecutar*, ordena la ejecución del script *"Ejecutar4.php"* en el servidor, recibiendo exactamente los mismos parámetros que *"Ejecutar2.php"*. Sin embargo, antes de la ejecución, el enlace realiza la llamada a la función JavaScript *VerificarDoc()* con la variable *tipo* perteneciente al script. La función estudia el valor del parámetro recibido alertando al usuario de que el archivo no está compilado o no pertenece a un lenguaje tratado en la aplicación, o por el contrario, permite la ejecución del script *"Ejecutar4.php"*.

Este script realiza las mismas operaciones que *"Ejecutar3.php"* a diferencia que en vez de guardar primero los datos del formulario *FormularioFichConf* en el fichero de configuración del usuario, realiza la operación inversa, es decir, lee el archivo de configuración recibido en la variable *fich_conf* y saca los valores de las variables *parametros*, *entrada*, *salida* y *timeout* para la creación del comando. El modelo seguido de construcción del comando y su funcionamiento es exactamente igual al descrito en el script *"Ejecutar3.php"*.

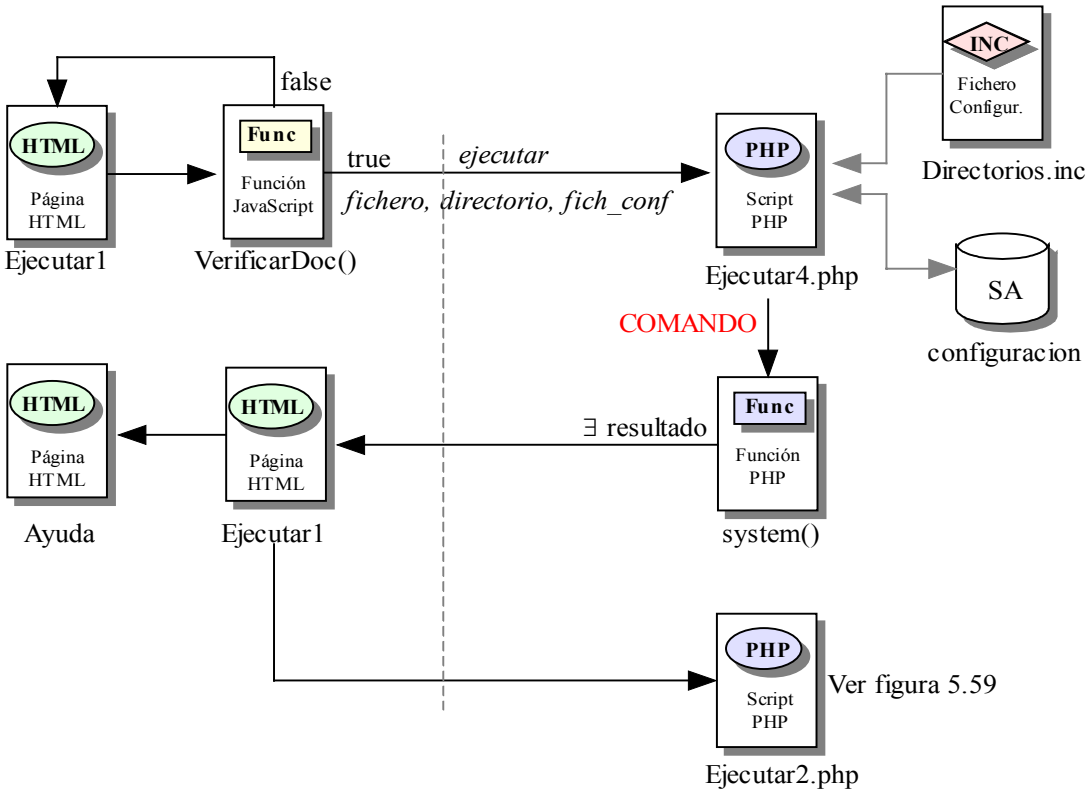


Figura 5.65 D.R. del Módulo de Ejecución. Marco Ejecutar1, opción Ejecutar

El último de los enlaces, *eliminar*, también recibe los mismos parámetros que los anteriores. Este vínculo ordena la ejecución en el servidor del script *"Ejecutar5.php"* pero antes, llama a ejecutar en el cliente a la función JavaScript *Confirmación()*. La función muestra un mensaje de confirmación donde el usuario corrobora que desea eliminar el fichero de configuración especificado. Si no es así, sigue cargado en el cliente la página del script. Si el usuario confirma la eliminación, el nuevo script es ejecutado.

"Ejecutar5.php" realiza la llamada a la función PHP *EliminarFichero()*. Gracias a la inclusión en el script del archivo *"FuncionesFicheros.inc"*, esta función, que recibe por parámetro la variable *fich_conf*, elimina el fichero especificado en ella. Después de eliminado el archivo, el script ordena de nuevo la ejecución de *"Ejecutar1.php"*.

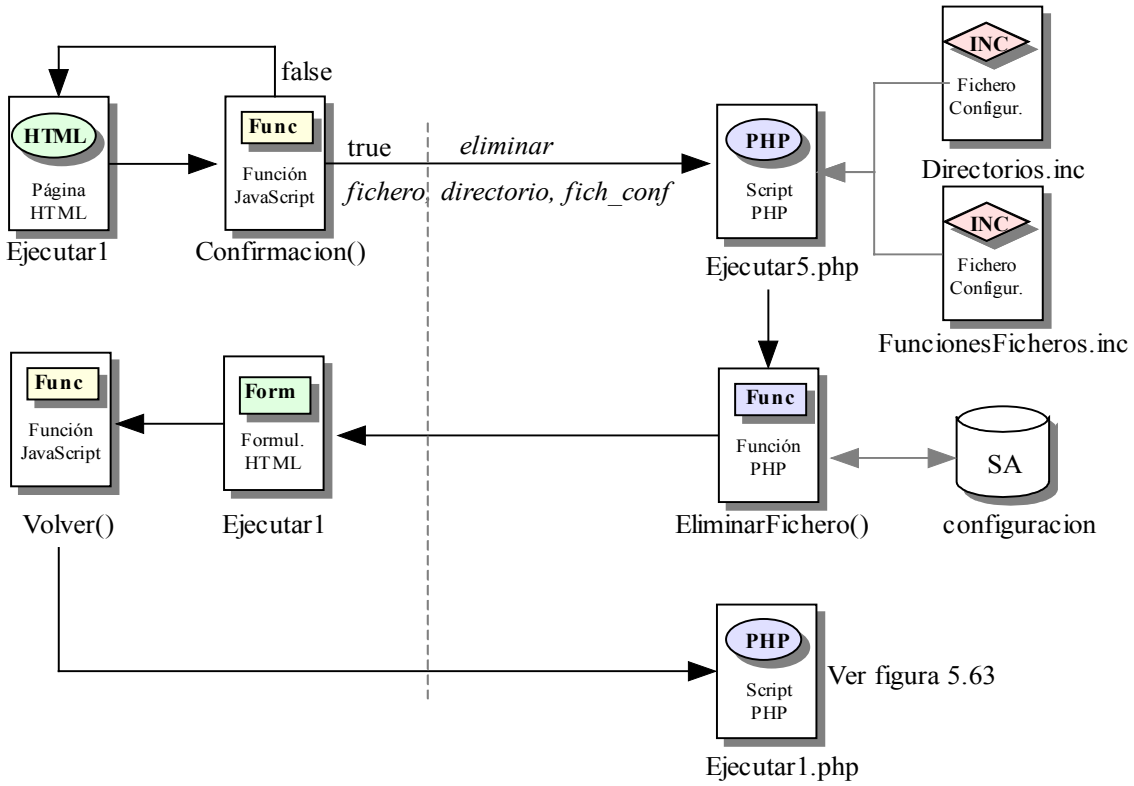


Figura 5.66 D.R. del Módulo de Ejecución. Marco Ejecutar1, opción Eliminar

El último marco presentado por la ejecución de *"FramesEjecutar.php"*, *Ejecutar6*, ordena la ejecución en el servidor del script *"Ejecutar6.php"*. Este script únicamente muestra en el cliente un vínculo que solicita el cierre de la ventana.

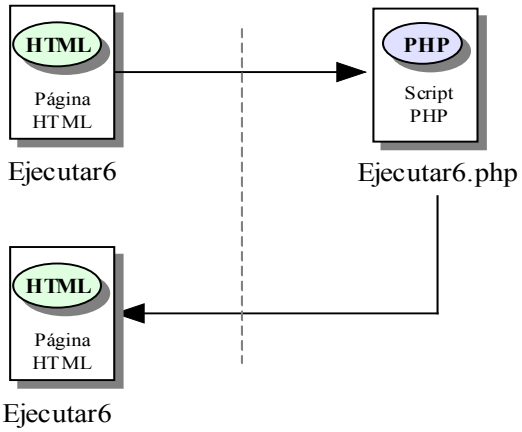


Figura 5.67 D.R. del Módulo de Ejecución. Marco Ejecutar6

5.3.10 Gestión de procesos

Todos los programas realizados por el usuario y lanzados a ejecutar en el servidor, se convierten en procesos. El tiempo de vida de un proceso se puede dividir en un conjunto de estados, cada uno con unas características determinadas. En la aplicación, no se quiere dar por hecho que el usuario tenga conocimientos a cerca de estos estados y por eso, se incorporan opciones donde el usuario gestiona sus propios procesos de una forma sencilla y sin necesidad de previo estudio.

5.3.10.1 Módulo de visualización y eliminación de procesos

La aplicación proporciona al usuario, un listado con todos los procesos lanzados. Además, muestra cierta información a cerca de cada uno de ellos, notificando al propio usuario, si el proceso ha finalizado o, por el contrario, continua ejecutándose en el servidor. De esta forma, el usuario de la aplicación sabe, en todo momento, si tiene algún proceso saturando al servidor. Además, por cada proceso vivo, la aplicación muestra un enlace que permite su finalización.

En el marco *bottonFrame*, el script *"MenuAdministrador.php"* carga en el cliente un vínculo que realiza una llamada a la función *en_ejecucion()*. La función se encuentra definida en el fichero *"Funciones.js"*, fichero que se incluye en el script. Esta función simplemente provoca la apertura de una nueva ventana de navegación mediante la función JavaScript *window.open()*. El script ordenado a ejecutar en el servidor con la apertura de la ventana es *"FramesProcesos.php"*.

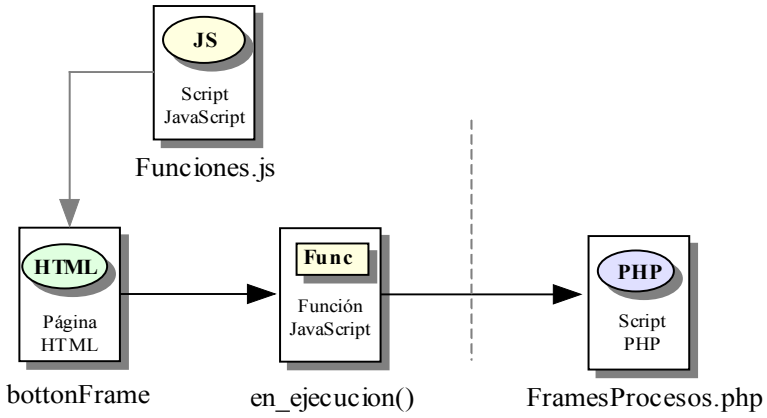


Figura 5.68 D.R. de la llamada al Módulo de Procesos

"FramesProcesos.php" carga en el cliente una página dividida en tres marcos, *Procesos1*, *Procesos2* y *Procesos3*, situados en la parte superior, media e inferior de la página respectivamente. Al igual que en ocasiones anteriores, estos tres marcos ordenan, a su vez, la ejecución de nuevos scripts.

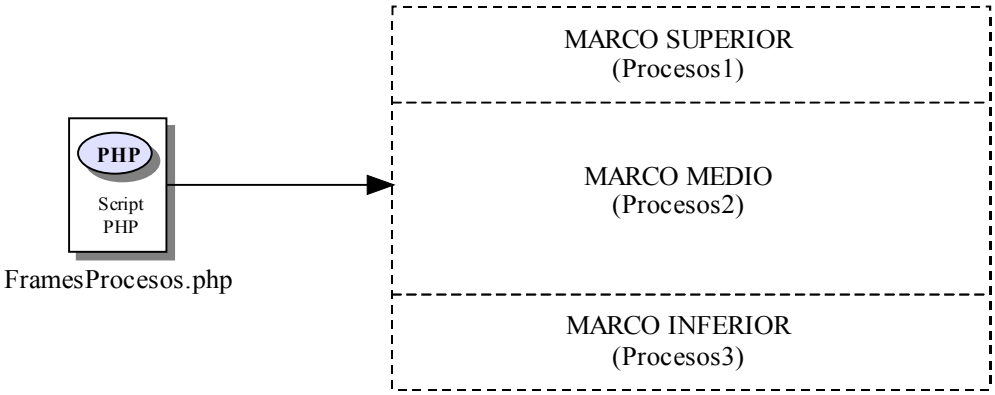


Figura 5.69 D.R. del Módulo de Procesos. Petición de marcos

El marco *Procesos1* ordena la ejecución en el servidor del script *"Procesos1.php"*. Este script carga una página en el cliente que muestra, a modo de título, la opción escogida por el usuario de Procesos. Además, el script presenta un vínculo que ordena la eliminación del listado de procesos pertenecientes al usuario registrado en la sesión. Para

ello, el vínculo solicita la ejecución del script *"EliminarExpediente.php"* en el marco *Procesos2*.

Hay que resaltar que *"EliminarExpediente.php"* elimina el listado de procesos y no los procesos en sí. Para evitar esta confusión, la aplicación no permite al usuario que realice esta operación si tiene algún proceso vivo en el servidor.

"EliminarExpediente.php" realiza una lectura del archivo *"ejecucion.txt"* ubicado en la carpeta *procesos* dentro del directorio personal del usuario registrado en la sesión. En este fichero, no sólo existe un listado de todos los procesos lanzados, sino que, además almacena cierta información sobre cada uno de ellos. Si en la lectura, el script detecta que existe al menos, un proceso vivo, ordena la llamada a la función JavaScript *ProcesosVivos()*. Esta función muestra en el cliente, un mensaje de alerta indicando al usuario que la eliminación no ha sido posible por existir procesos vivos. Si por el contrario, el script no encuentra información en el fichero *"ejecucion.txt"*, de que existe proceso vivo, entonces ordena la eliminación del archivo mediante la función PHP *exec*. Por último, el script realiza la inclusión de otro script, *"Procesos2.php"*.

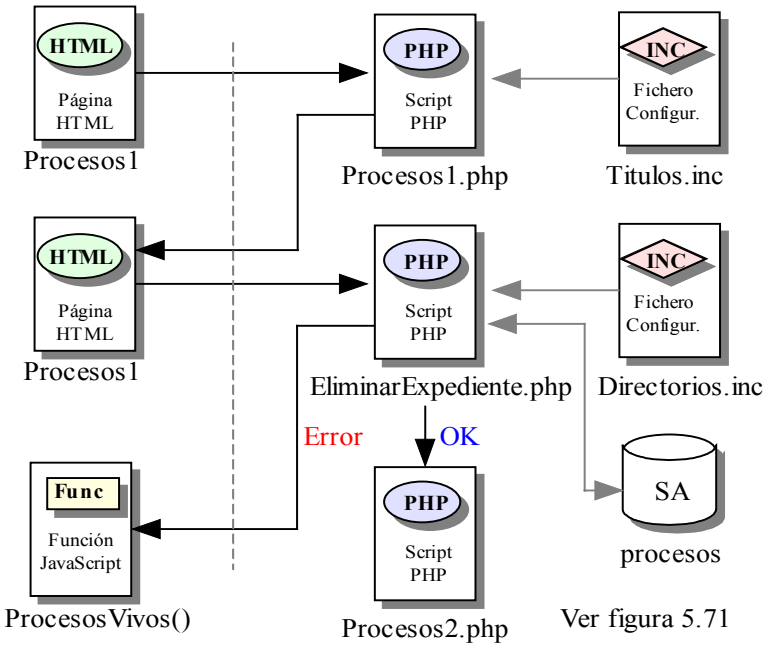


Figura 5.70 D.R. del Módulo de Procesos. Marco Procesos1

El marco *Procesos2* ordena la ejecución en el servidor del script *"Procesos2.php"*. Lo primero que hace este script es comprobar que existe el fichero *"ejecucion.txt"* perteneciente al usuario registrado. Si no existe, el script carga en el cliente una página informando que no hay ningún proceso en el historial. Si el fichero existe, el script realiza una lectura sacando toda la información de los procesos lanzados. El script carga entonces, en el cliente, un listado mostrando en cada fila el nombre y ruta del proceso lanzado por el usuario. Al lado de cada proceso, el script carga el estado en el que se encuentra el proceso y si se puede, o no, eliminar. Toda esta información es sacada por el script del propio archivo *"ejecucion.txt"*. Si el proceso ha finalizado y no a causa del parámetro *timeout*, el script carga al lado del proceso, las palabras "Finalizado" y "Normal". Si es el parámetro *timeout*, el motivo por el que el proceso ha finalizado, el script muestra las palabras "Finalizado" y "Timeout". Y por último, si el script comprueba que el proceso no ha finalizado, muestra la palabra "En ejecución" y un vínculo que ordena la ejecución de un nuevo script, *"MatarProceso.php"*.

"MatarProceso.php" recibe un parámetro de *"Procesos2.php"*, llamado *id* que corresponde con el identificador del proceso a eliminar. El script mata al proceso mediante el paso del correspondiente comando a la función PHP *exec*, provocando su finalización. Después, el script incluye de nuevo al script *"Proceso2.php"*.

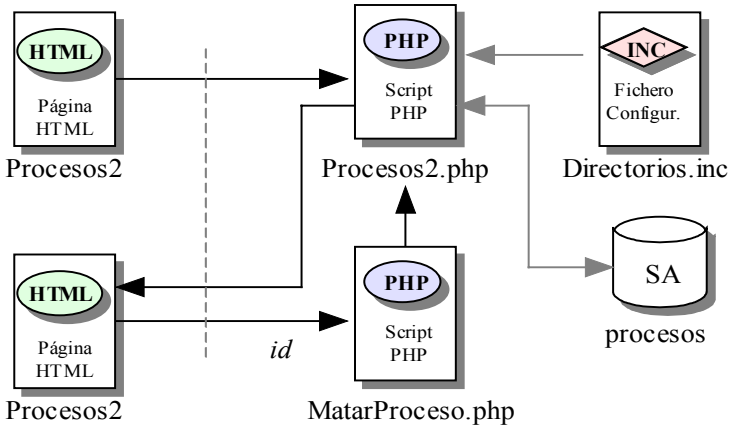


Figura 5.71 D.R. del Módulo de Procesos. Marco Procesos2

El marco *Procesos3* ordena la ejecución en el servidor del script *"Procesos3.php"*. Este script simplemente, carga una página en el cliente donde se muestra al usuario un vínculo que ordena el cierre de la ventana mediante la llamada a la función característica de JavaScript, *top.close()*.

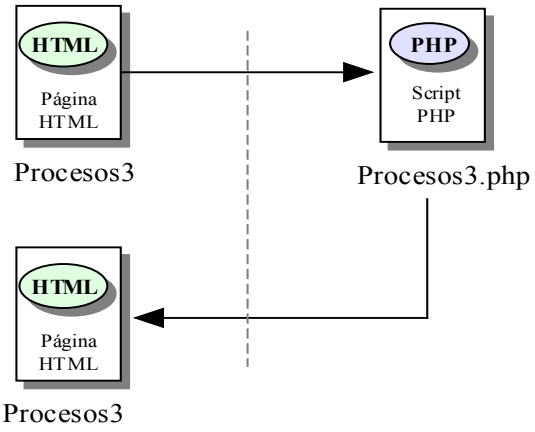


Figura 5.72 D.R. del Módulo de Procesos. Marco Procesos3

5.3.11 Otras opciones de trabajo

5.3.11.1 Módulo de salidas

Todos los vínculos cargados por el script *"MenuAdministrador.php"* en el marco *bottonFrame*, han sido descritos en apartados anteriores, a excepción de uno. Este vínculo, aún no detallado, es el encargado de mostrar los ficheros de salida provocados por la ejecución de los programas del usuario.

Este vínculo realiza la llamada a la función *salidas()*, que es definida en el fichero *"Funciones.js"*, y la cual se incluye en *"MenuAdministrador.php"*. La función provoca la apertura en el cliente de una nueva ventana de navegación mediante la llamada a la función JavaScript *window.open()*. Con ella, se ordena la ejecución en el servidor de un script llamado *"FramesEjecutados.php"*.

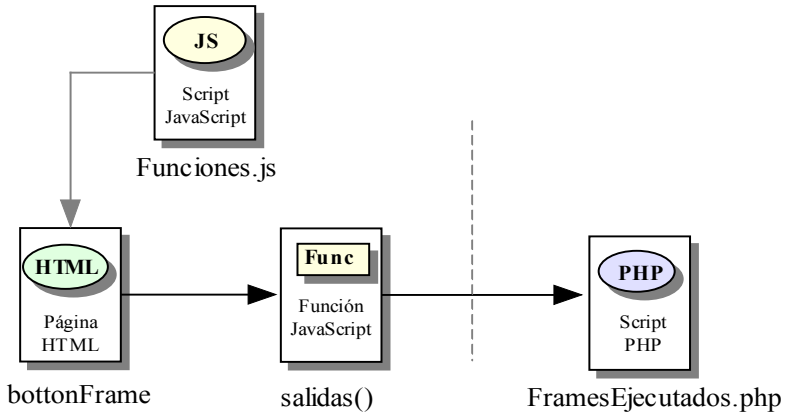


Figura 5.73 D.R. de la llamada al Módulo de Salidas

"FramesEjecutados.php" carga en el cliente una página dividida en tres marcos, *Ejecutados0*, *Ejecutados1* y *Ejecutados6*, situados en la parte superior, media e inferior de la página respectivamente. Cada uno de estos marcos ordena la ejecución de nuevos scripts en el servidor.

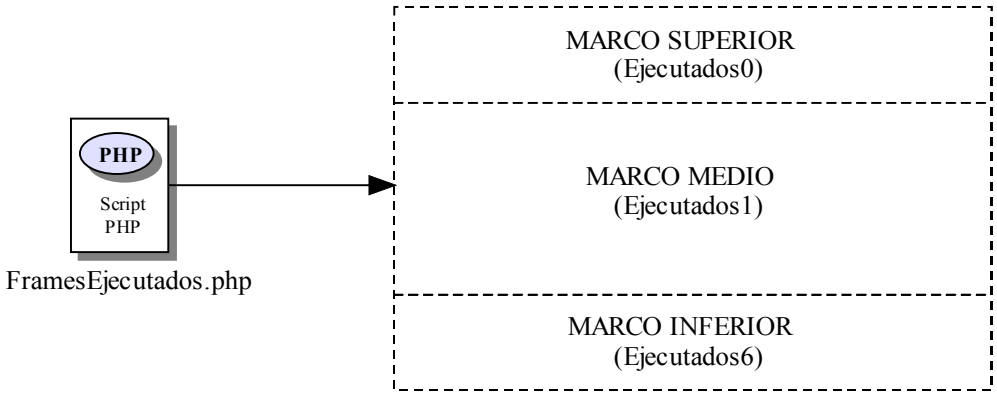


Figura 5.74 D.R. del Módulo de Salidas. Petición de marcos

El marco *Ejecutados0* ordena la ejecución del script *"Ejecutados0.php"* en el servidor. Este script sólo carga en el cliente una página con información, a modo de título, de la opción seleccionada de Salidas.

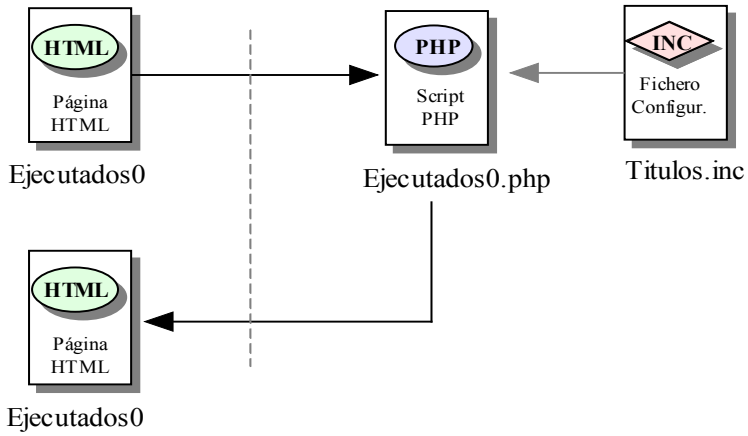


Figura 5.75 D.R. del Módulo de Salidas. Marco Ejecutados0

El marco *Ejecutados1* ordena la ejecución del script *"Ejecutar1.php"* en el servidor. Este script llama a la función *MostrarEjecutados()* pasando por parámetro la ruta del directorio *ejecucion*, ubicado dentro del directorio personal del usuario registrado en la sesión. Dentro de esta carpeta se encuentran todos los ficheros de salida creados por la ejecución de los programas lanzados por el usuario. La función *MostrarEjecutados()* lee el contenido del directorio y carga en el cliente el listado de archivos contenidos en él. Junto a cada fichero, el script carga dos enlaces, *consultar* y *eliminar*. Cada uno de estos enlaces ordena la ejecución de nuevos scripts.

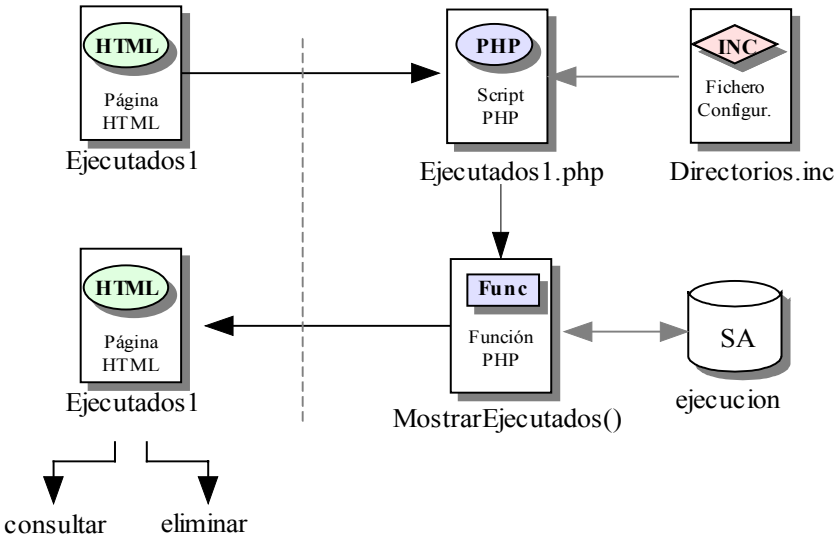


Figura 5.76 D.R. del Módulo de Salidas. Marco Ejecutados1

El enlace *consultar*, ordena la ejecución en el servidor del script *"Ejecutados2.php"*. Este script recibe el parámetro *fichero* con el archivo de salida seleccionado. *"Ejecutados2.php"* abre dicho archivo para lectura y carga en el cliente su contenido. Junto a esto, el script carga un nuevo vínculo que ordena la ejecución del último script ejecutado, es decir, *"Ejecutados1.php"*.

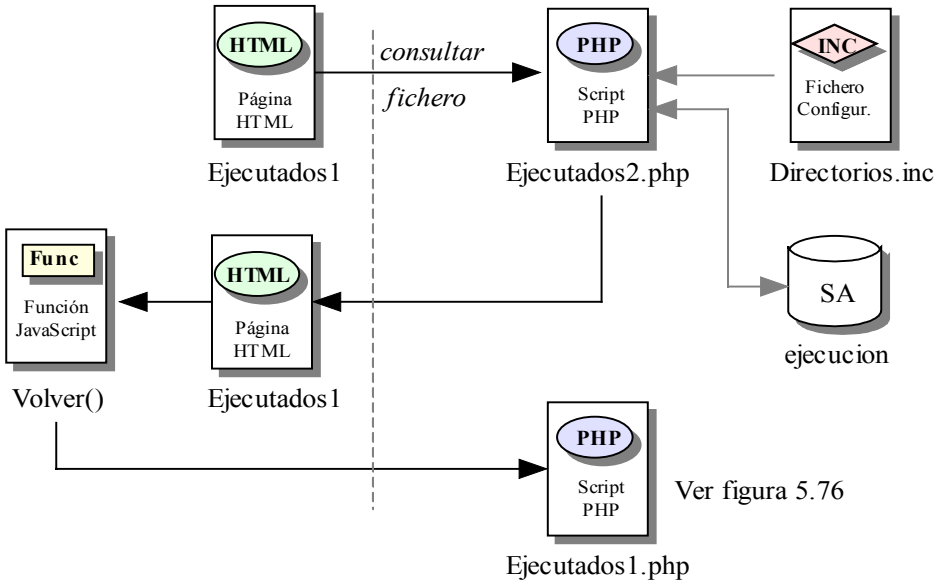


Figura 5.77 D.R. del Módulo de Salidas. Marco Ejecutados1, opción Consultar

El enlace *eliminar*, ordena la ejecución del script *"Ejecutados3.php"* en el servidor, pero antes, realiza la llamada a la función JavaScript *Confirmacion()*. Esta función muestra un mensaje de confirmación en el cliente, donde el usuario debe corroborar que desea eliminar el fichero seleccionado, o por el contrario, cancelar la operación, permaneciendo la página cargada por *"Ejecutados1.php"*, abierta. Si embargo, si el usuario reafirma que desea eliminar el fichero, se lanza la ejecución del script *"Ejecutados3.php"* en el servidor. *"Ejecutados3.php"* llama a la función *EliminarFichero()*, la cual, gracias a la inclusión del fichero *"FuncionesFicheros.inc"* en el script, elimina el archivo de salida seleccionado. Después, el script incluye a otro script, *"Ejecutados1.php"*, que vuelve a cargarse en el marco *Ejecutados1*.

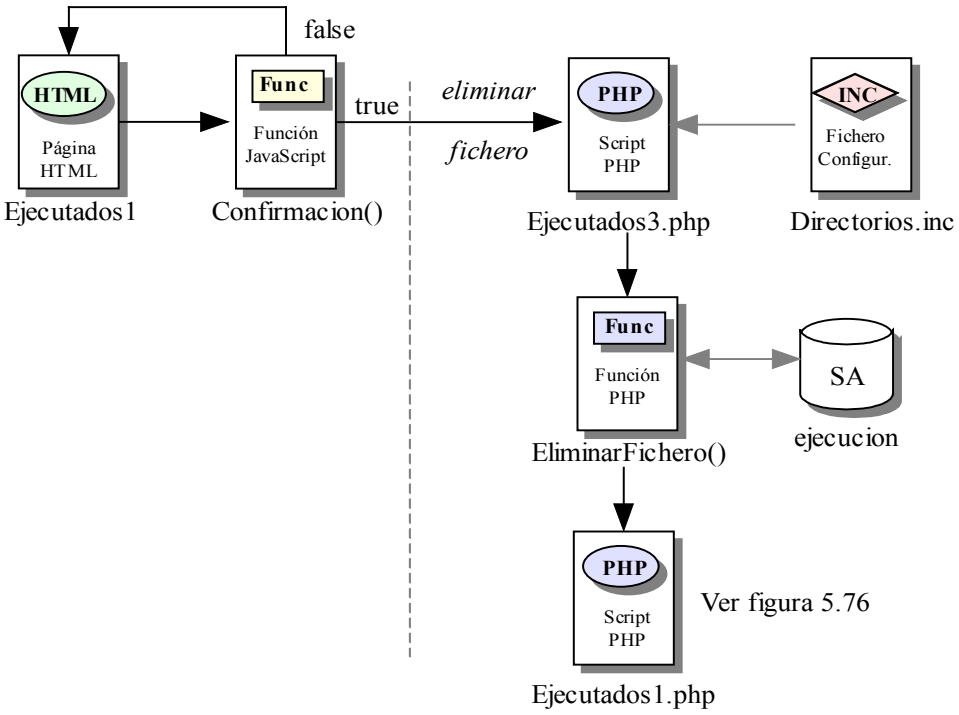


Figura 5.78 D.R. del Módulo de Salidas. Marco Ejecutados1, opción Eliminar

Por último, el marco *Ejecutados6* ordena la ejecución del script *"Ejecutados6.php"* en el servidor. Este script carga una página en el cliente, que presenta un vínculo, el cual provoca el cierre de la ventana de navegación abierta en el "Módulo de salidas".

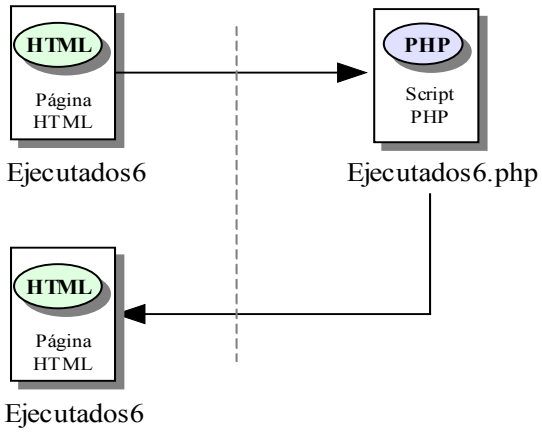


Figura 5.79 D.R. del Módulo de Salidas. Marco Ejecutados6

5.3.11.2 Módulo de acceso al bloque de administración

En el apartado 5.3.4.1 se especificó que en la aplicación existen dos bloques bien diferenciados, el bloque de administración y el bloque de trabajo. También se detalló, que el administrador era el único usuario con privilegios suficientes para pasar de uno a otro. Cómo se realiza el paso del bloque de administración al bloque de trabajo queda aclarado en dicho aparatado. Sin embargo, el paso del bloque de trabajo al bloque de administración se aclara en este módulo.

En el marco *topFrame*, correspondiente al bloque de trabajo, el script *"BanerAdministrador.php"* carga un vínculo que ordena la ejecución en el servidor de otro script, *"FramesAdministracion.php"*. Este script carga en el cliente el bloque de administración y es explicado con exhausto detalle en el apartado 5.3.2.

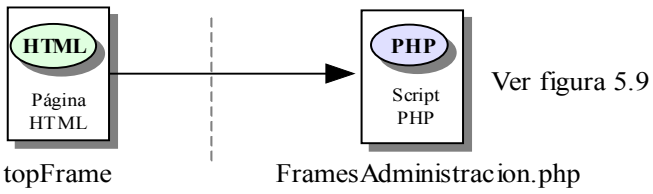


Figura 5.80 D.R. del Módulo de Acceso al Bloque de Administración

5.3.11.3 Módulo de manual de C y C++

El Proyecto consiste en proporcionar al usuario un entorno de trabajo donde pueda desarrollar programas bajo los lenguajes de C y C++ para potenciar los conocimientos sobre éstos. Si bien, el usuario puede mandar a ejecutar cualquier programa desarrollado bajo alguno de estos lenguajes, es razonable pensar que siempre puede aparecer alguna duda por su parte a la hora de crearlos. Por este motivo, la aplicación ofrece la posibilidad de acceder directamente a manuales de C y C++ facilitando el trabajo al usuario.

Existen dos vínculos en el marco *topFrame*, cargados en el cliente por la ejecución en el servidor del script *"BanerAdministrador.php"*, o por el script *"BanerAlumno.php"*, que ofrecen el contenido de los manuales. Uno de los vínculos llama a la función

ManualC(), el otro, a la función *ManualCPP()*. Ambas funciones provocan la apertura de una ventana de navegación en el cliente, mediante la llamada a la función característica de JavaScript *window.open()*, que cargan el contenido de un manual de C y un manual de C++ respectivamente.

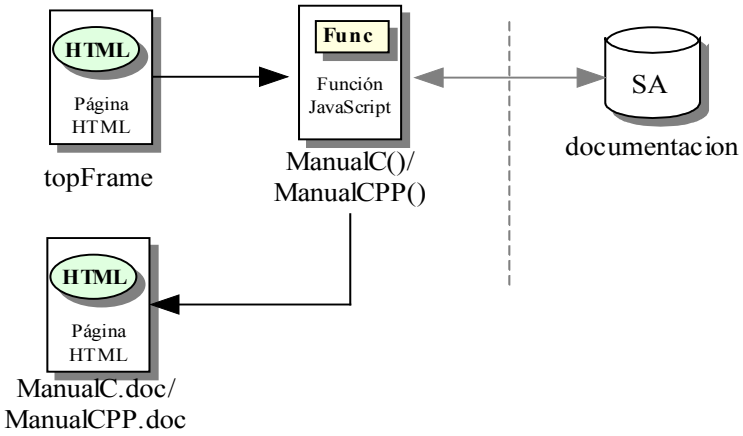


Figura 5.81 D.R. del Módulo de Manual de C y C++

5.3.11.4 Módulo de ayuda

Como en toda aplicación, se habilita al usuario una ayuda que detalla el correcto funcionamiento de la herramienta. El acceso a esta ayuda se sitúa en el marco *topFrame*.

La ejecución en el servidor del script *"BanerAdministrador.php"*, o del script *"BanerAlumno.php"*, carga en el cliente el vínculo que realiza una llamada a la función *ManualUsuario()*. Esta función mediante una función característica del lenguaje JavaScript produce la apertura de una nueva ventana de navegación cuyo contenido corresponde con el contenido del manual de usuario de la aplicación.

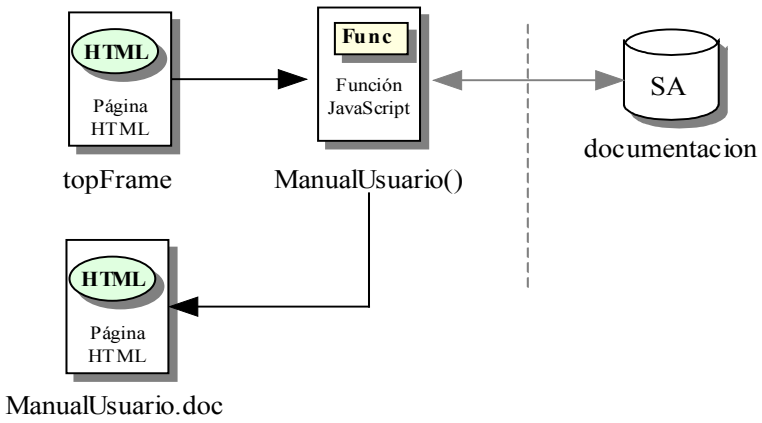


Figura 5.82 D.R. del Módulo de Ayuda

5.3.11.5 Módulo de cambio de contraseña

Todos los usuarios de la aplicación acceden a ella a través de unos valores que se encuentran almacenados en la tabla *usuarios*. Estos valores son impuestos por el administrador de la aplicación a la hora de insertar nuevos usuarios y recibidos mediante correo electrónico por los propios usuarios, a excepción del administrador de la aplicación, cuyos valores le son impuestos por el administrador del sistema.

Sin embargo, la aplicación ofrece la posibilidad de cambiar estos valores de acceso, de forma que, cada usuario pueda imponer unos valores propios para su entrada a la herramienta.

El "Módulo de cambio de contraseña" es el encargado de realizar dicha función y es accesible desde el bloque de trabajo y el bloque de administración. El marco *topFrame*, correspondiente a ambos bloques, es el encargado de presentar en el cliente los vínculos que realizan la llamada mediante la ejecución de los scripts *"BanerAdministrador.php"*, *"BanerAlumno.php"* y *"BanerAdministracion.php"*.

Los tres scripts cargan en el cliente los vínculos que llaman a ejecutar la función JavaScript *CambiarPassword()*. Esta función obliga la apertura de una nueva ventana de navegación y ordena la ejecución del script *"FramesCambiarPassword.php"* en el servidor.

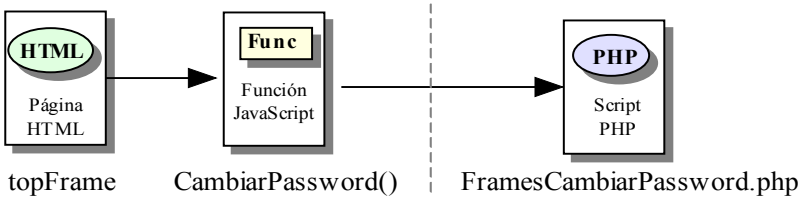


Figura 5.83 D.R. de la llamada al Módulo de Cambio de Contraseña

Este script provoca la división de la nueva página en tres marcos diferentes, *CambiarPassword1*, *CambiarPassword2* y *CambiarPassword0*, situados en la parte superior, media e inferior respectivamente.

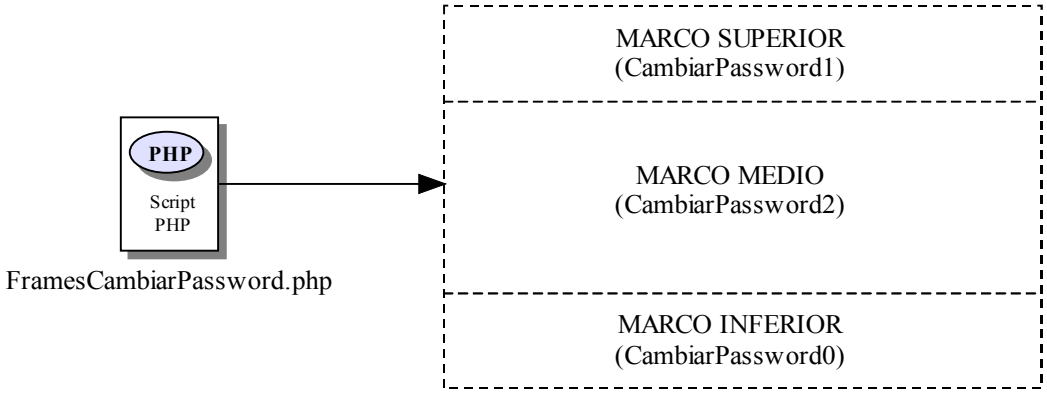


Figura 5.84 D.R. del Módulo de Cambio de Contraseña. Petición de marcos

El marco *CambiarPassword1* solicita la ejecución en el servidor del script *"CambiarPassword1.php"*. Este script únicamente muestra en el cliente el título de la operación solicitada, en este caso, cambiar contraseña, gracias a la inclusión del fichero *"Titulos.inc"*.

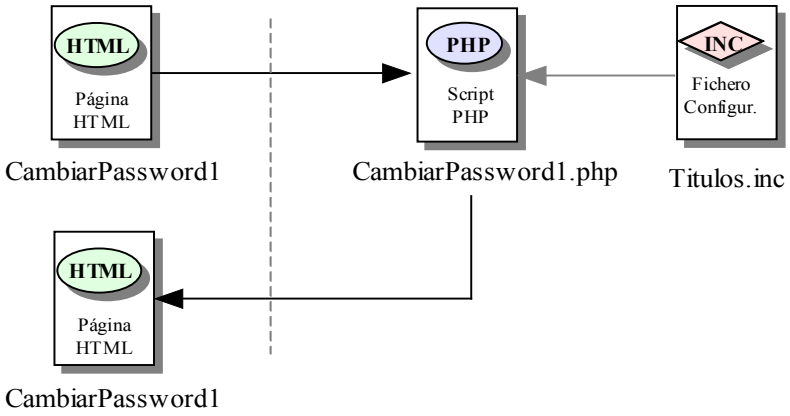


Figura 5.85 D.R. del Módulo de Cambio de Contraseña. Marco CambiarPassword1

El marco *CambiarPassword0* solicita la ejecución en el servidor del script *"CambiarPassword0.php"*. El script solamente muestra en el cliente, un vínculo que provoca la llamada a una función característica del lenguaje JavaScript, *top.close()*. Esta función ordena el cierre de la ventana.

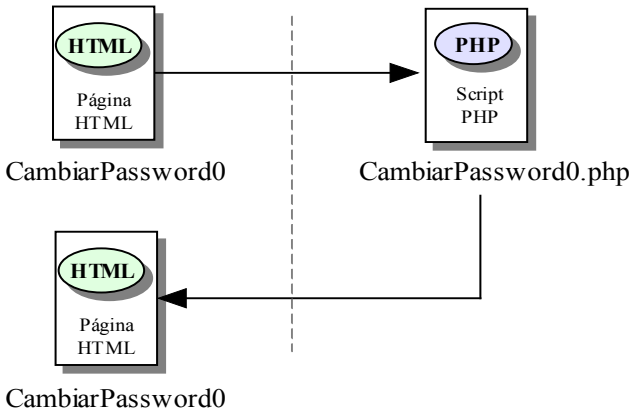


Figura 5.86 D.R. del Módulo de Cambio de Contraseña. Marco CambiarPassword0

El marco *CambiarPassword2* ordena la ejecución en el servidor del script *"CambiarPassword2.php"*. Este script presenta en el cliente un formulario de inserción con tres campos, donde el usuario debe introducir su contraseña actual, la nueva contraseña que desea tener y una corroboración de esta nueva contraseña. Una vez seleccionado el botón de conformidad, se ordena la ejecución en el servidor de un nuevo script,

"CambiarPassword3.php", en el mismo marco *CambiarPassword2* y, el cual recibe los valores introducidos en el formulario.

Lo primero que hace *"CambiarPassword3.php"* es comprobar que los dos últimos campos del formulario recibido del script *"CambiarPassword2.php"* son iguales. Si no es así, muestra en el cliente un mensaje de error indicando al usuario que debe introducir dos veces la nueva contraseña. Junto a este mensaje, el script muestra un botón que llama una función característica del lenguaje JavaScript, *history.go()*, con parámetro '–1' para obligar, otra vez, el script anterior.

Si los valores de la nueva contraseña son iguales, entonces el script *"CambiarPassword3.php"*, mediante la inclusión de la ejecución del script *"conexion.php"*, llama a la función *ConsultarBD()* para realizar en el servidor de Base de Datos una consulta que devuelva si en la tabla *usuarios* se encuentra el usuario registrado en la sesión y su contraseña coincide con la recibida mediante el formulario. Si no existe ningún registro en la tabla que cumpla estas condiciones, el script vuelve a ejecutar en el cliente la función JavaScript *history.go()* con parámetro '–1' para ejecutar el script anterior de nuevo.

Si por el contrario, existe el registro en la tabla *usuarios*, el script vuelve a llamar a la función *ConsultarBD()*, para esta vez modificar el registro con la nueva contraseña introducida en el formulario. Seguidamente, muestra en el cliente un mensaje informando del cambio de contraseña y un botón de conformidad que ordena la llamada a la función JavaScript, *top.close()*, la cual produce el cierre de la ventana.

En la figura 5.87 se muestra el diagrama de relaciones correspondiente al "Módulo de cambio de contraseña".

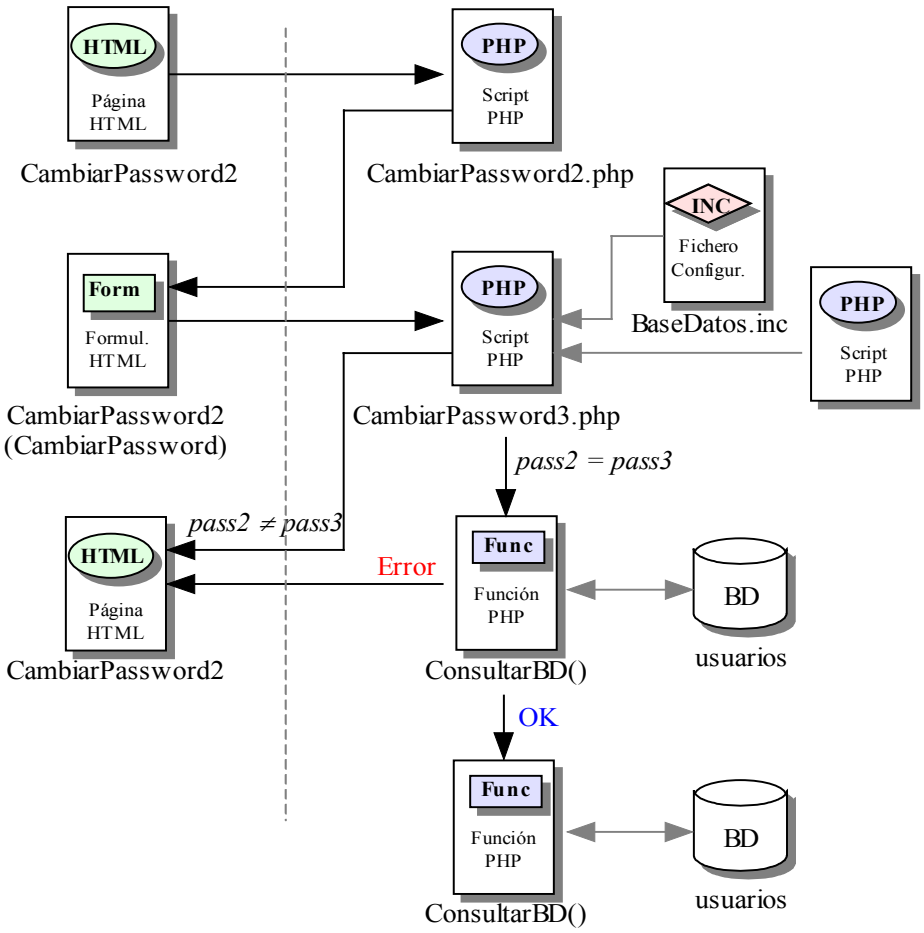


Figura 5.87 D.R. del Módulo de Cambio de Contraseña. Marco CambiarPassword2

6 Manual de administrador

6.1 Introducción

El manual de administrador describe, de forma concisa, el manejo de la aplicación desde el punto de vista de la gestión. Todas las acciones que se puedan realizar en este bloque quedan personalizadas bajo la figura del usuario de gestión o administrador y son las acciones que la aplicación requiere para su funcionamiento.

Se ha de tener en cuenta que la figura de administrador es única en la aplicación y que no existen diferentes niveles de administración. Sólo hay dos tipos de usuarios, los usuarios restringidos y el administrador. Asimismo, el administrador como tal, podrá darse de baja insertando antes un nuevo gestor, no permitiendo en ningún caso que la aplicación quede sin la figura del administrador.

Los aspectos cubiertos en este manual sólo incluyen aquellos que son realizables y accesibles únicamente por el administrador de la aplicación. Si bien el gestor está habilitado para acceder a ambos bloques, bloque de administración y bloque de trabajo, sólo serán descritos en este capítulo aquellos procedimientos que son propios de la gestión. Aquellos otros que sean pertenecientes al bloque de trabajo han de ser vistos en el manual de usuario contenido en el capítulo siete de la memoria.

La descripción planteada en este manual no contempla detalles relativos al desarrollo ni al funcionamiento interno de la aplicación.

6.2 Acceso. Autentificación de usuario

Para acceder a la aplicación como administrador, como cualquier otro usuario dado de alta, éste debe introducir en el navegador web, la siguiente dirección: http://servidor.teleco.ulpgc.es/programacion

Una vez establecida la conexión y previa carga de la página de inicio, el servidor procede a autentificar al usuario y reconocer a éste como administrador, y por consiguiente, otorgarle con privilegios no dados al resto de usuarios. Tanto la autentificación como el reconocimiento del usuario como gestor, se llevan a cabo con los valores de *Usuario* y *Contraseña* introducidos en la ventana de diálogo mostrada en la figura 6.1.

Figura 6.1 Ventana de verificación de usuario válido

Una vez comprobado el usuario y reconocido su nivel de gestor en la aplicación, se pasa a cargar la página de inicio, la cual es exactamente igual a la del resto de usuarios, que se presenta en el manual de usuario de la memoria, a excepción de un nuevo enlace, desde el cual, se permitirá entrar en el bloque de administración.

En caso de error, es decir, en caso de que los valores introducidos en los campos de *Usuario* y *Contraseña* no sean correctos, el servidor vuelve a presentar la ventana de diálogo anterior para que los valores de conexión sean introducidos y enviados de nuevo. Si se pulsa *Cancelar*, el proceso de conexión queda anulado.

6.3 Entorno de administración

Hay que tener en cuenta que el administrador de la aplicación es un usuario más, y como tal, accede directamente al bloque de trabajo. Para que el administrador pueda pasar del bloque de trabajo al bloque de administración, se le habilita una opción nueva, sólo hábil para el gestor, desde la cual accede a la parte de la gestión. Esta opción se encuentra en la parte superior de la página de inicio del usuario, *marco de encabezado*, del bloque de trabajo (el bloque de trabajo se detalla en el capítulo 7, correspondiente al manual de usuario). Si el usuario hace clic sobre esta opción, el navegador comenzará a cargar la página de inicio del bloque de administración.

Figura 6.2 Marco de encabezado del bloque de trabajo con enlace al bloque de administración

Una vez finalizada la carga de la página de inicio de la parte de administración, el navegador mostrará una estructura como refleja la figura 6.3. Esta estructura forma el entorno de la gestión, la cual es bastante diferente a la del entorno de trabajo. Esto es así puesto que, tanto los objetivos como las necesidades de los mismos son bastantes diferentes.

En este bloque de administración, el gestor será capaz de realizar todas las operaciones propias de la gestión de la aplicación.

El administrador permanecerá en este bloque hasta que finalice su sesión, o bien, hasta que pase al bloque de trabajo, desde el cual, podrá volver al bloque de administración. Esta operación la podrá realizar tantas veces como desee.

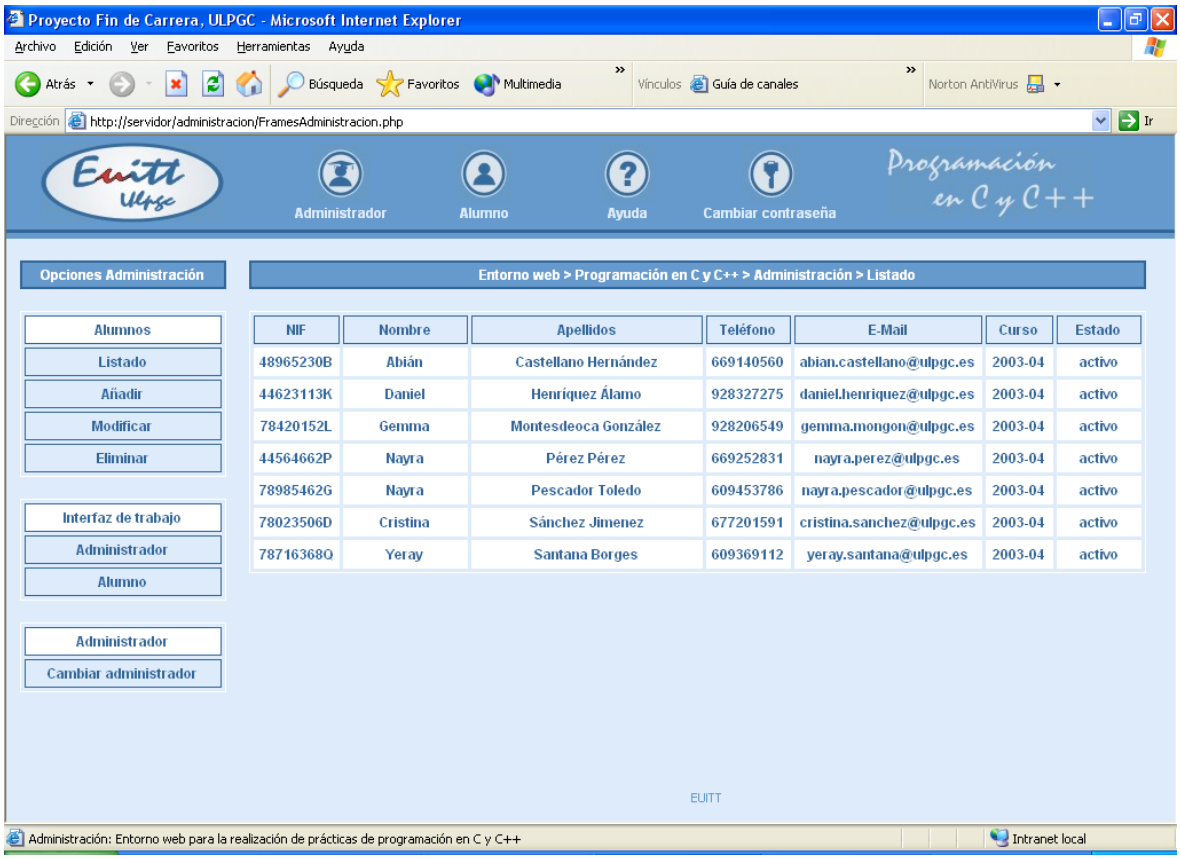


Figura 6.3 Página de entorno de usuario. Bloque de Administración

En la figura 6.3 también queda reflejado que la página de inicio de administración está fragmentada en tres partes o marcos. En la parte superior se encuentra el *marco de encabezado*, el cual consta, de entre otros, con un enlace al bloque de trabajo del propio administrador y otro enlace al bloque de trabajo de un alumno que deberá seleccionar. En la parte izquierda está el *marco de opciones*, en donde se presenta un menú con las diferentes operaciones que se trabajan en la administración de la aplicación y desde los cuales, el administrador llevará a cabo toda la gestión. Por último, en la parte derecha se encuentra el *marco de trabajo*, cuyo contenido depende de la opción escogida en el *marco de opciones* y en donde el administrador trabajará todos los datos pertenecientes a la gestión.

6.3.1 Marco de encabezado

Este marco se encuentra en la parte superior del bloque de administración y en él se presentan una serie de enlaces, tal y como se muestra en la figura 6.4.

Figura 6.4 Marco de encabezado

Si bien todas las operaciones pertenecientes al administrador de la aplicación para realizar la gestión de la misma, se encuentran en el *marco de opciones*, en el *marco de encabezado* se sitúan dos enlaces a dos de estas operaciones, *Administrador* y *Alumno*. El primero es el enlace a la operación de acceso al bloque de trabajo como administrador y el segundo es el enlace a la operación de acceso al bloque de trabajo como alumno. Estas dos opciones serán detalladas junto a las restantes en el apartado 6.4 de este manual.

Además de las dos opciones de acceso al bloque de trabajo, en el *marco de encabezado* se encuentran dos opciones más: la ayuda de la aplicación y la opción de cambio de contraseña. El enlace *Ayuda* accede directamente a este manual donde, el usuario encontrará la solución a cualquier duda que pudiera ser planteada durante la gestión de la aplicación. El enlace *Cambiar contraseña* presenta una nueva ventana donde el administrador podrá cambiar su dato de acceso, introduciendo su contraseña actual, la nueva contraseña deseada y una corroboración de esta nueva contraseña. Una vez realiza la opción de cambio de contraseña correctamente, el administrador ha de tener en cuenta que éste será su nuevo valor de conexión a la aplicación. En caso de existir error al introducir alguno de estos datos, se mostraría al usuario una notificación con el motivo que habría causado dicho error. La ventana de diálogo para el cambio de contraseña se presenta en la figura 6.5.

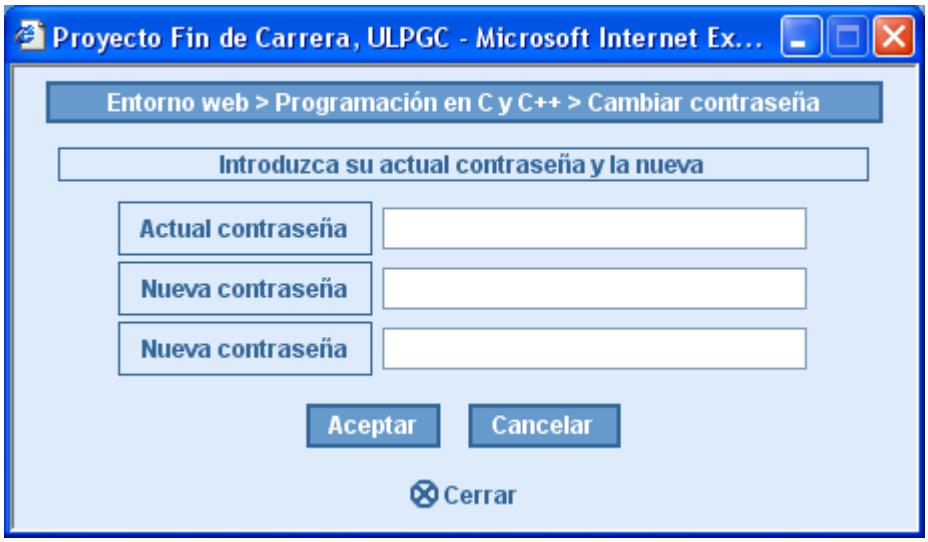


Figura 6.5 Ventana Cambiar contraseña

6.3.2 Marco de opciones

El *marco de opciones* se sitúa en la parte inferior izquierda del bloque de administración. Este marco contiene todas las operaciones que el administrador de la aplicación requiere para llevar a cabo toda la gestión. Estas operaciones se encuentran divididas en tres grupos, según sus funcionalidades. En la figura 6.6 se muestra el *marco de opciones* con la división de los tres grupos y las opciones en cada una de ellos. Estos tres grupos son: *Alumnos, Interfaz de trabajo* y *Administrador*.

Figura 6.6 Marco de opciones

6.3.3 Marco de trabajo

El *marco de trabajo* se encuentra en la parte inferior derecha del bloque de administración. El contenido de este marco dependerá en todo momento de la opción escogida por el administrador en el *marco de opciones*. Cada vez que el gestor selecciona una de estas opciones, el *marco de trabajo* cambiará la presentación del mismo. Las distintas posibilidades del *marco de trabajo* se detallan en el siguiente apartado de este manual junto con las funcionalidades de las distintas operaciones del *marco de opciones*. En la figura 6.7 se muestra el *marco de opciones* en una de sus posibilidades.

Figura 6.7 Marco de trabajo

6.4 Operaciones de administración

En este apartado se describen de forma detallada todas y cada una de las operaciones del bloque de administración. Cada una de ellas tiene su enlace de acceso en el *marco de opciones* y todas son sólo accesibles por el administrador de la aplicación, el cual es el único usuario con permiso para entrar a este bloque.

Primeramente, se explican todas las operaciones pertenecientes al primer grupo del *marco de opciones*, *Alumnos*, en donde las acciones que se realizan gestionan a todos los usuarios de la aplicación.

A continuación, se detallan las operaciones del segundo grupo del *marco de opciones*, *Interfaz de trabajo*. Las operaciones pertenecientes a este grupo realizan las acciones de administración correspondientes al acceso, por parte del administrador, al bloque de trabajo como administrador o como un determinado alumno previamente seleccionado, permitiendo al gestor tener acceso a todo el trabajo realizado por cada uno de los alumnos.

Por último, se especifica la operación perteneciente al tercer grupo del *marco de opciones*, *Administrador*. En este apartado sólo existe una única opción en la que se modifican los valores de conexión del usuario como administrador. Será el propio gestor quien modifique estos valores, pudiendo ceder su papel de administrador a alguien externo a la aplicación, si permite que esta nueva persona introduzca los nuevos valores de conexión, y por tanto, sea ésta quién único tenga acceso a la aplicación como gestor. Si esto ocurre, habrá ocurrido un cambio de administrador.

6.4.1 Listado

Esta opción corresponde con la primera operación del grupo *Alumnos* perteneciente al *marco de opciones*. A través de esta operación, el administrador visualiza en el *marco de trabajo* el listado completo de todos los usuarios dados de alta en la aplicación, exceptuando al propio administrador, permitiendo a su vez, contemplar los datos de cada uno de ellos. En el listado, el administrador podrá observar, tanto los datos personales del alumno, como el NIF, nombre, apellidos, teléfono y dirección de correo, como sus datos concernientes a la escuela, es decir, el curso y el estado. El gestor de la aplicación deberá tener en cuenta la importancia del valor del campo de estado, el cual especifica si el usuario puede o no, tener acceso a la aplicación.

Para una mayor comprensión, el listado se encuentra encabezado por los nombres de los siete campos: NIF, Nombre, Apellidos, Teléfono, E-Mail, Curso y Estado, con los que se define al alumno. Cada uno de ellos presenta un enlace que permite hacer una ordenación del listado por el campo seleccionado. Es decir, si en el listado hacemos clic sobre el enlace *Nombre*, el orden del listado será modificado para presentarse en orden alfanumérico según el campo Nombre. Nótese que por defecto, el listado se muestra por orden según el campo Apellidos.

En la figura 6.8 se muestra un ejemplo de cómo se presenta el *marco de trabajo* en la opción *Listado*.

Figura 6.8 Marco de trabajo: Opción Listado

6.4.2 Añadir

Esta opción corresponde con la segunda operación del grupo *Alumnos* perteneciente al *marco de opciones*. Una vez el administrador pulsa sobre este enlace, el *marco de trabajo* muestra un formulario donde el gestor de la aplicación debe introducir los datos de un nuevo alumno. En la figura 6.9 se muestra el aspecto del *marco de trabajo* una vez seleccionado el enlace *Añadir*.

Figura 6.9 Marco de trabajo: Opción Añadir

El administrador debe tener en cuenta, que si bien, no todos los campos son necesarios a la hora de dar de alta a un nuevo usuario, existen algunos que son indispensables de especificar. Para ello, la aplicación muestra un mensaje de alerta indicando el campo que debemos rellenar obligatoriamente, en caso de no haberlo hecho.

A la hora de la inserción del nuevo alumno, la aplicación también controla antes de incluir los datos, que no existe ningún usuario con el mismo NIF ni con el mismo nombre de usuario que el nuevo alumno a introducir. En caso de ocurrir esta situación, se mostraría una notificación de error, especificando un caso u otro, según proceda.

Si no ocurre ningún fallo al introducir los valores del nuevo usuario, se presenta una confirmación de que el alumno ha sido insertado correctamente en la aplicación.

6.4.3. Modificar

Se presenta como la tercera opción dentro del grupo *Alumnos* del *marco de opciones*. Si el gestor hace clic sobre este enlace, en el *marco de trabajo* se presenta un listado, parecido al mostrado en la opción de *Listado*, a excepción de una nueva columna mediante la cual se selecciona el usuario a modificar. Nótese que al igual que en el caso de la opción *Listado*, existen enlaces que permiten la ordenación alfanumérica de los usuarios según el campo seleccionado en el enlace. En la figura 6.10 se presenta el *marco de trabajo* una vez seleccionada la opción *Modificar*.

Figura 6.10 Marco de trabajo: Opción Modificar

Cuando el administrador selecciona a un alumno haciendo clic sobre *Aceptar*, el *marco de trabajo* cambia para presentar los datos de dicho alumno y poder así editarlos. El valor del NIF no puede modificarse. Cambiados los valores del alumno, el administrador actualiza los datos pulsando sobre el botón de *Aceptar*. En caso de querer borrar los cambios antes de que hayan sido salvados, el administrador debe pulsar sobre el botón de *Cancelar*.

La figura 6.11 muestra el *marco de trabajo* en la edición de los datos de un alumno.

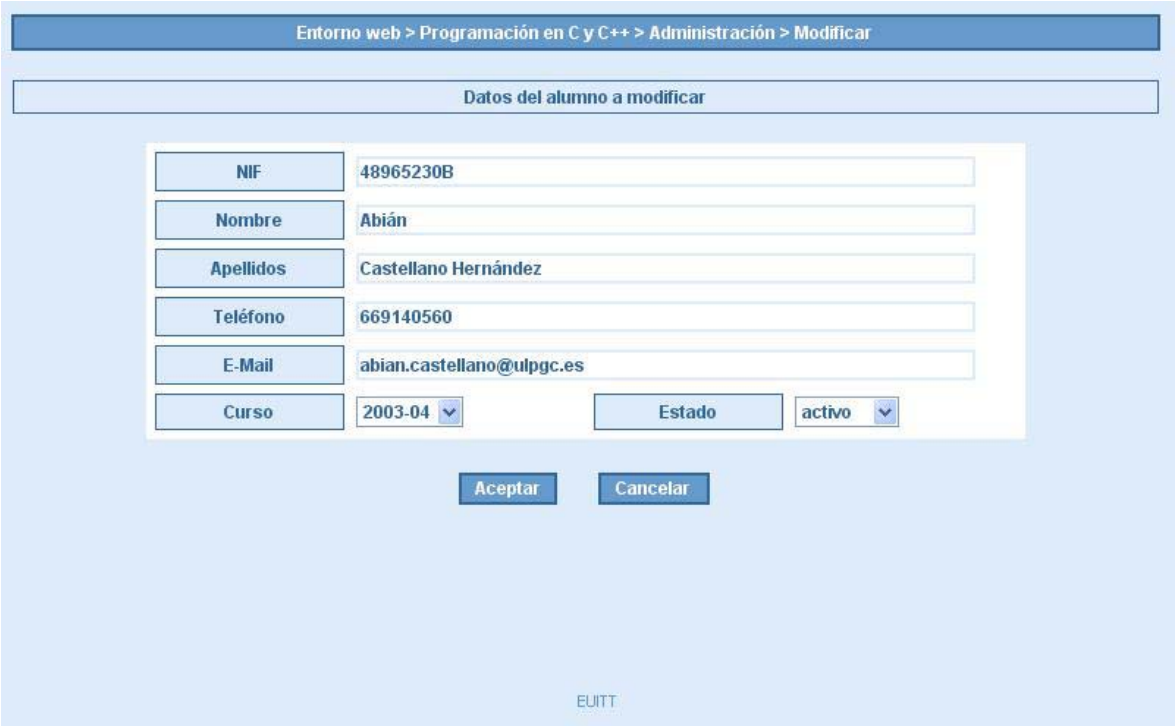


Figura 6.11 Marco de trabajo: Formulario Modificar

6.4.4 Eliminar

La opción *Eliminar* es la última de las operaciones presentadas en el grupo *Alumnos* del *marco de opciones*. Haciendo clic sobre este enlace, el *marco de trabajo* vuelve a mostrar, como en casos anteriores, un listado con posibilidad de ordenación según algún campo, con la novedad de que pueden seleccionarse más de un usuario a la vez, incluso permitiéndose la selección total de todos los usuarios. Esto permite al administrador la eliminación masiva, facilitando el trabajo del gestor, no teniendo éste que eliminar alumno por alumno, en caso de así requerirlo. Una vez seleccionado, el alumno o los alumnos, el administrador debe pulsar sobre *Aceptar* y corroborar, en una nueva ventana de confirmación, que desea la eliminación para concluir con dicha operación. Una vez realizado esto, se muestra un mensaje de verificación.

En la figura 6.12 se muestra el *marco de trabajo* una vez escogida la opción de *Eliminar* del *marco de opciones*.

Figura 6.12 Marco de trabajo: Opción Eliminar

6.4.5 Administrador

En apartados anteriores, se ha resaltado que la figura del gestor, como tal, tiene acceso a los dos bloques que presenta la aplicación. Cuando el administrador accede a la aplicación se presenta el bloque de trabajo, en él se muestra un enlace mediante el cual se accede al bloque de administración. La opción *Administrador* dentro del grupo de *Interfaz de trabajo* realiza la operación inversa a esto, es decir, a través de este enlace, el administrador pasa del bloque de administración al bloque de trabajo. Por tanto, el gestor tiene durante su conexión, la posibilidad de pasar del bloque de trabajo al bloque de administración y viceversa tantas veces como desee. Recuérdese que esta opción también se presenta en el *marco de encabezado*.

6.4.6 Alumno

El administrador de la aplicación tiene asignado el trabajo de gestionar a todos los usuarios, dar de alta a nuevos alumnos, restringir sus accesos en caso de así requerirlo,

eliminarlos cuando proceda y ceder su papel de administrador a un nuevo usuario si así lo desea. Además, el administrador, al igual que el resto de usuarios, puede hacer uso del bloque de trabajo incluyendo todas las operaciones que éste presenta. No obstante, el gestor posee el privilegio de poder acceder al bloque de trabajo de cualquiera de los usuarios dados de alta en la aplicación, es decir, tiene la posibilidad de acceder al bloque de trabajo de un alumno como si de éste se tratara. De esta manera, se pretende que el gestor pueda, en todo momento, tener acceso al trabajo realizado por los alumnos. Esta operación se consigue haciendo clic sobre el enlace *Alumno* perteneciente al grupo de *Interfaz de trabajo* del *marco de opciones*. Una vez el administrador pulsa en este enlace, el marco de trabajo muestra el listado con todos los alumnos, de la misma manera que en opciones anteriormente descritas, y una columna donde se selecciona un determinado usuario para acceder a su bloque de trabajo. Después de seleccionar el alumno, se pulsa sobre el botón de *Aceptar* presentándose en el navegador el bloque de trabajo de éste, mostrando su directorio de trabajo y quedando hábiles todas las opciones para poder gestionar todo el trabajo realizado por él. Por consiguiente, el gestor no sólo puede administrar la aplicación, sino también el trabajo de todos los usuarios.

Figura 6.13 Marco de trabajo: Opción Alumno

6.4.7 Cambiar administrador

La figura del administrador queda bien diferenciada del resto a la hora de tener acceso a determinadas operaciones que presenta la aplicación. Al mismo tiempo que posee mayor privilegio, tiene mayor responsabilidad sobre ella. Todo esto hace que la figura del gestor sea indispensable. Por ello, la aplicación no puede quedar sin esta figura, lo que no implica que no se pueda cambiar el papel de administrador de un usuario a otro. Para realizar esta operación se hace clic sobre el enlace *Cambiar administrador*, única operación del grupo *Administrador* del *marco de opciones*. Cuando el usuario hace clic sobre ella, el *marco de trabajo* muestra un formulario donde se solicita el *Usuario* y *Contraseña* del administrador, es decir, el gestor debe introducir los mismos valores con los que se conecta a la aplicación. Una vez comprobados los valores introducidos, el *marco de trabajo* muestra un nuevo formulario, figura 6.14, donde se introducen los nuevos valores de *Usuario* y *Contraseña* con los que el nuevo administrador accederá a la aplicación. De no existir error al realizar esta operación, se solicitará al usuario de la sesión abierta (antiguo administrador) que cierre la conexión quedando el nuevo administrador creado.

Figura 6.14 Marco de trabajo: Opción Cambiar administrador

7 Manual de usuario

7.1 Introducción

Ésta es una ayuda donde se describe de forma detallada el manejo de la aplicación desde el punto de vista del usuario. En ella se especifican todas y cada una de las opciones que existen en la aplicación para su correcto funcionamiento. La ayuda no hace referencia a los detalles relativos al desarrollo ni al funcionamiento interno de dicha aplicación.

7.2 Acceso. Autentificación de usuario

Para acceder a la aplicación debe cargarse en el navegador web la siguiente dirección: http://servidor.telematica.ulpgc.es/programacion. Una vez cargada, se procede a la autentificación del usuario dentro del sistema, para lo cual se muestra la ventana de diálogo de la figura 7.1.

Figura 7.1 Ventana de verificación de usuario válido

En ella, el usuario debe introducir los valores de *Usuario* y *Contraseña* que previamente le fueron asignados por el administrador de la aplicación y que el usuario recibe de forma automática a través de correo electrónico.

Al hacer clic sobre *Aceptar* se envían los valores introducidos al servidor para su autentificación. Una vez se comprueba que el usuario está dado de alta en el servidor se pasa a comprobar que tiene permiso de entrada a la aplicación por parte del administrador. Si no hay error en los valores introducidos y comprobado que el usuario puede acceder a la aplicación, se procede a la carga de la página de inicio. Si el usuario no tiene permiso para acceder, se cargará en el navegador una página donde se le informa de su situación de acceso denegado.

En caso de que el usuario no sea ni siquiera autentificado, se vuelve a presentar de nuevo la ventana de verificación para un nuevo envío de los valores. Si se hace clic sobre *Cancelar*, el proceso de conexión a la aplicación será cancelado.

7.3 Entorno de usuario

Una vez cargada la pagina de inicio, el navegador mostrará una estructura como refleja la figura 7.2.

Esta estructura forma el entorno en el cual el alumno trabajará mientras dure la conexión. Esta presentación está fragmentada en cuatro partes o marcos. En la parte superior se encuentra el *marco de encabezado*. En ella, el usuario podrá, entre otras cosas, tener acceso a los manuales de los lenguajes que trata la asignatura para la cual se ha desarrollado esta aplicación. A la izquierda, está el *marco de navegación*, con una estructura en forma de árbol desplegable. En la parte derecha superior, se encuentra el *marco de operaciones*, donde se presenta una barra de botones con las diferentes opciones y, en la parte derecha inferior, está el *marco de presentación*, donde únicamente se muestra el nombre y contenido de un fichero.

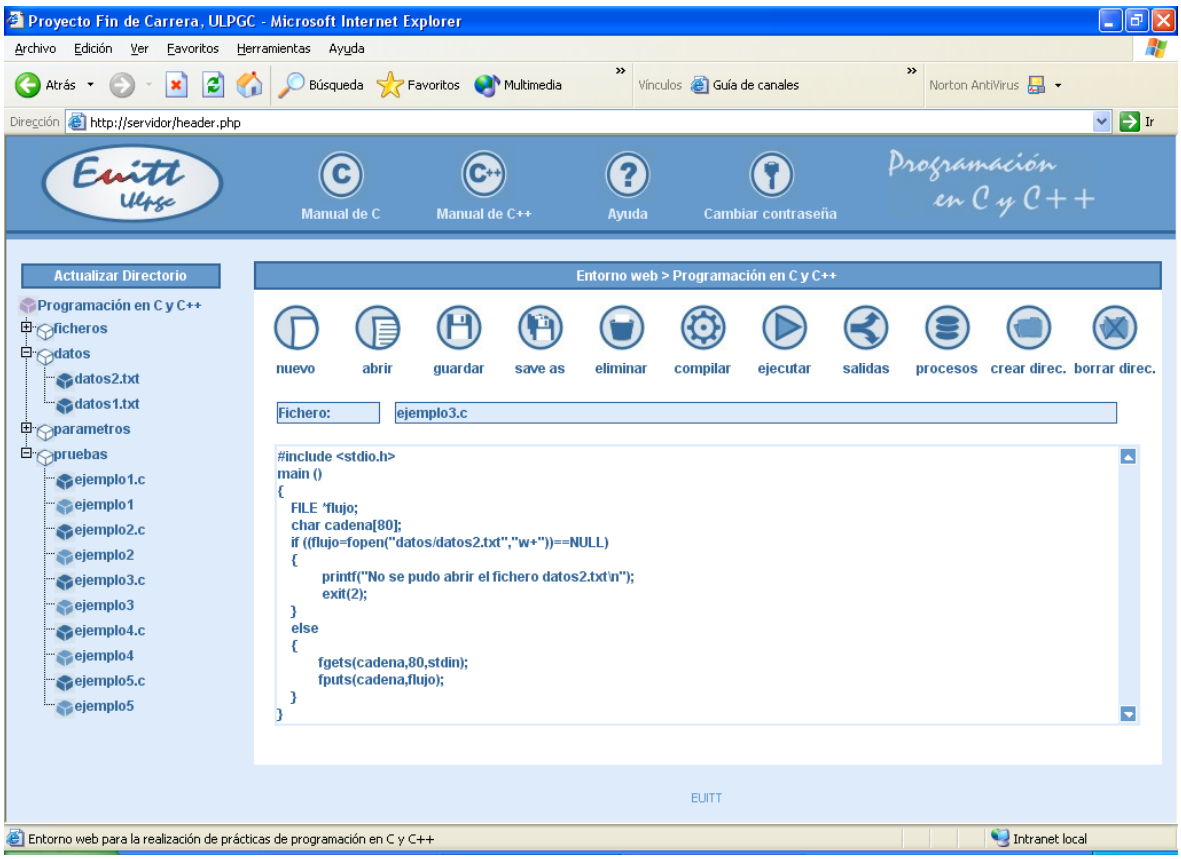


Figura 7.2 Página de entorno de usuario. Bloque de trabajo

7.3.1 Marco de encabezado

Este *marco de encabezado*, situado en la parte superior del entorno de usuario y cuya presentación se muestra en la figura 7.3, contiene cuatro diferentes opciones: manual de C, manual de C++, ayuda de la aplicación y cambio de contraseña.

Figura 7.3 Marco de encabezado

Manual de C: enlace a un manual del lenguaje C. El usuario puede hacer uso de este manual a la hora de crear los programas en C.

Manual de C++: enlace a un manual del lenguaje C++. El usuario puede hacer uso de este manual a la hora de crear los programas en C++.

Ayuda de la aplicación: enlace a esta ayuda.

Cambio de contraseña: cuando el administrador da de alta a un nuevo usuario, debe asignarle un *Nombre de usuario* y una *Contraseña*, a través de los cuales, el alumno podrá acceder a la aplicación. Este *Nombre de usuario* y *Contraseña* serán enviados por correo electrónico al propio alumno para que éste tenga constancia de dicha información. Una vez el alumno utiliza estos datos para entrar en la página de inicio, puede cambiar la contraseña inicialmente asignada por el administrador haciendo clic en esta opción. El usuario debe introducir la contraseña actual, la nueva contraseña deseada y una corroboración de la nueva contraseña, tal y como se muestra en la figura 7.4.

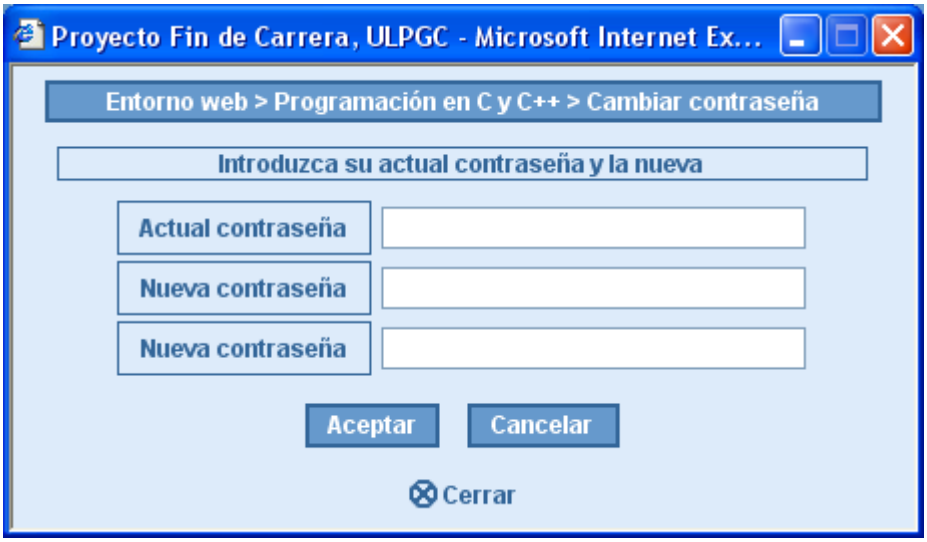


Figura 7.4 Ventana Cambiar contraseña

Si todo ha ido bien, se mostrará un mensaje de confirmación de cambio de contraseña. En caso contrario, se mostrara un mensaje de error indicando que ha habido un fallo al introducir la contraseña actual o un mensaje de error indicando que los dos campos de nueva contraseña no son iguales, según proceda un caso u otro.

7.3.2 Marco de navegación

En la parte izquierda de la interfaz de trabajo del usuario, está el *marco de presentación*. En él existe una estructura en forma de árbol desplegable. Dicho árbol agrupa una serie de nodos que son sensibles a las pulsaciones del ratón y constituyen los enlaces a los ficheros que se cargarán en el *marco de presentación*.

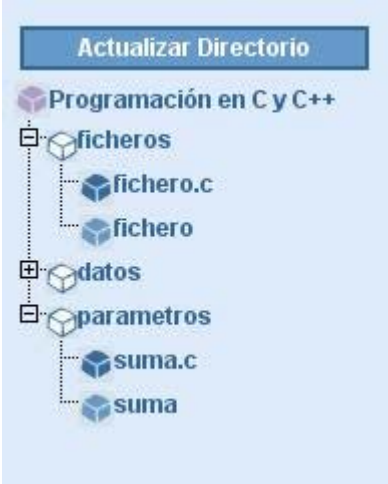


Figura 7.5 Marco de navegación

En aquellas ocasiones, en las que el texto de cada nodo exceda las dimensiones del marco de navegación, es posible actuar sobre las barras de desplazamiento del marco para visualizar este texto al completo. De la misma forma, si el número de nodos es tan grande que no se puede visualizar todos los nodos a la vez, se podrá hacer uso de estas barras para desplazarse de arriba abajo y viceversa para poder observar el contenido del árbol en su totalidad.

En la figura 7.5, se observa que hay nodos que llevan a su lado un pequeño cuadro. Esto es indicativo de que estos nodos son expansibles, es decir, que al hacer clic sobre alguno de ellos, se desplegarán los nuevos subnodos agrupados en su interior que a su vez podrán ser expansibles o no. De la misma manera, esto se consigue haciendo clic en el propio enlace en lugar de hacerlo sobre el cuadro.

El contenido del *marco de navegación* depende de cada usuario. Cada vez que un usuario es dado de alta en la aplicación, se crea un directorio de trabajo asociado a dicho usuario, siendo éste independiente y sólo accesible por el propio usuario y por el administrador de la aplicación. Este directorio de trabajo, inicialmente vacío, es el que mostrará el *marco de presentación* y su contenido será el que el propio usuario vaya formando a medida que vaya creando nuevos directorios y archivos. Por tanto, los marcos de navegación mostrarán para cada usuario, diferentes contenidos unos de otros.

Notar que, haciendo clic en los enlaces de los directorios, no se muestra nada en el *marco de presentación*, tal y como se presenta en el inicio de la sesión, mientras que los enlaces de los ficheros muestran, tanto el nombre, como el contenido del fichero seleccionado en el enlace dentro del *marco de presentación*. Los ficheros ejecutables son los únicos que no tienen asignado enlace puesto que el contenido de éstos no es relevante para el usuario.

Además de la estructura de árbol desplegable, en la parte superior del *marco de navegación* se encuentra un botón fijo de *Actualizar Directorio*. Cada vez que el usuario haga clic en este botón, el contenido del árbol se refrescará y la estructura del árbol desplegable volverá a la forma inicial, es decir, todos los nodos volverán a la forma sin expandir. Algunas opciones del *marco de operaciones* realizarán esta actualización de directorio de forma automática, mientras que en otras, será conveniente que el usuario la realice de forma manual para refrescar el listado de su directorio en el árbol mostrando así los posibles cambios realizados en él.

7.3.3 Marco de operaciones

En la parte derecha superior del entorno de usuario se muestra el *marco de operaciones*. Tal y como se muestra en la figura 7.6, este marco está formado por once botones, cada uno de los cuales representa un enlace a las diferentes operaciones que el usuario puede realizar en esta aplicación. Estas operaciones se detallan en el apartado 4 del presente manual.

Figura 7.6 Marco de operaciones

7.3.4 Marco de presentación

El *marco de presentación* está ubicado en la parte derecha inferior del entorno de usuario. El mismo está constituido por dos partes, una donde se especifica el nombre de un fichero, y otra, donde se muestra el contenido de dicho fichero. Cada vez que el usuario vaya a crear un nuevo fichero deberá introducir el texto en la parte de contenido. El *marco de presentación* se muestra en la figura 7.7.

Figura 7.7 Marco de presentación

7.4 Operaciones de usuario

En este apartado se hará una descripción pormenorizada de todas aquellas operaciones que el usuario puede realizar en la aplicación.

En ella, y a través de la web, se podrá llevar a cabo la realización de prácticas de programación con los lenguajes de C y C++. El usuario podrá crear todo tipo de ficheros incluyendo los archivos con extensión *.c y .cpp*. Una vez guarde el fichero, si lo ha hecho con una de estas extensiones, el alumno podrá compilarlo para su posterior ejecución. Todo ello, se realizará de forma remota, de manera que la aplicación también constará de operaciones para que el usuario pueda controlar estas ejecuciones desde su máquina local. A continuación se detallan las operaciones del usuario.

7.4.1 Nuevo

La aplicación esta diseñada para que el usuario implemente sus propios programas con los lenguajes de programación de C y C++, los compile y los ejecute, quedando todo ello de manera fehaciente, por parte del administrador de la aplicación. Para ello, el usuario cada vez que desee crear un nuevo fichero, hará clic en esta opción del *marco de operaciones*, dejando el campo de contenido del *marco de presentación* vacío, para que el alumno pueda empezar a introducir el texto del nuevo fichero.

Por ejemplo, si el usuario desea crear un nuevo programa en C que imprima por pantalla el mensaje "Hola mundo", deberá hacer clic en la opción *Nuevo* y teclear el siguiente texto dentro del campo de contenido:

```
#include <stdio.h>
main() {
    printf("Hola mundo");
}
```

En la figura 7.8 se muestra como queda el *marco de presentación* cuando se inserta el nuevo texto.

Si existiese anteriormente un fichero abierto en la parte del *marco de presentación* y los cambios de éste no estuviesen salvados, al hacer clic sobre esta opción, dichos cambios no serían guardados, por lo que el usuario perdería la información. De igual manera, se perdería toda la información si se estuviese creando un nuevo fichero y antes de ser guardado se hiciera clic en la opción de *Nuevo*. Por este motivo, se recomienda siempre al usuario que antes de seleccionar ninguna opción, se asegure que el fichero ha sido salvado previamente.

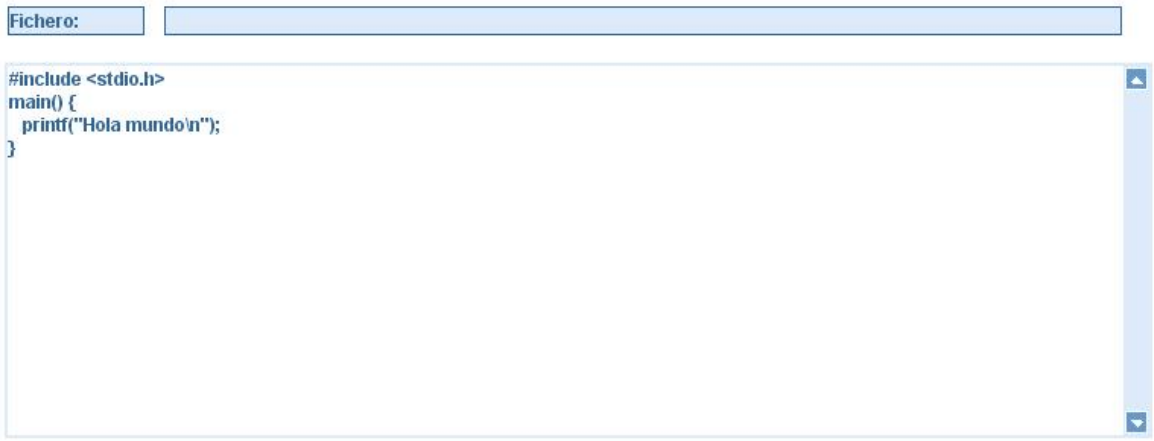


Figura 7.8 Marco de presentación cuando se inserta texto para un nuevo fichero

7.4.2 Abrir

Todos los ficheros creados por el usuario son accesibles en todo momento por éste. Existen dos formas de abrir un fichero, una mediante el árbol desplegable situado en el *marco de navegación*, y la otra haciendo clic en la opción de *Abrir* del *marco de operaciones*.

Una vez el usuario seleccione la opción de *Abrir*, se abrirá una nueva ventana que consta de tres partes. La parte situada en la zona superior indica la opción escogida, es decir, indica que el alumno ha seleccionado la opción de *abrir*.

La parte media muestra el contenido del directorio del usuario, pudiendo existir dentro de él tres tipos de elementos: directorios, ficheros ejecutables y resto de ficheros. Los tres tipos están visiblemente diferenciados, como se muestra en la figura 7.9 y sus funciones son totalmente distintas. Los directorios contienen un enlace a dicho directorio, es decir, una vez el usuario hace clic sobre el enlace de un directorio, el contenido de la zona media de la nueva ventana cambia y muestra el contenido del directorio seleccionado. Además, existe la posibilidad de que el usuario vuelva al directorio anterior haciendo uso del enlace *Subir un nivel ...* Los ficheros ejecutables, al igual que en la estructura de árbol desplegable, presentado en el *marco de navegación*, no presentan enlaces puesto que el contenido de éstos no es de interés para el usuario. El resto de ficheros muestran un enlace

en el que pulsando con el ratón sobre uno de ellos, abrirá el fichero seleccionado en el campo de contenido del *marco de presentación* y cuyo nombre se mostrará en el campo fichero de dicho marco. Al mismo tiempo que la aplicación abre el fichero, la nueva ventana de selección se cierra.

Si una vez abierta la nueva ventana, el usuario decide no abrir ningún archivo, puede cerrarla sin tener que pulsar sobre ningún enlace de fichero, y por consiguiente, sin tener que abrir ningún archivo. Esta posibilidad se presenta en la parte inferior de la ventana donde se muestra un enlace de *Cerrar*. Este enlace permite que la ventana se cierre quedando el *marco de presentación* de la misma manera que antes de seleccionar la opción de *Abrir*.

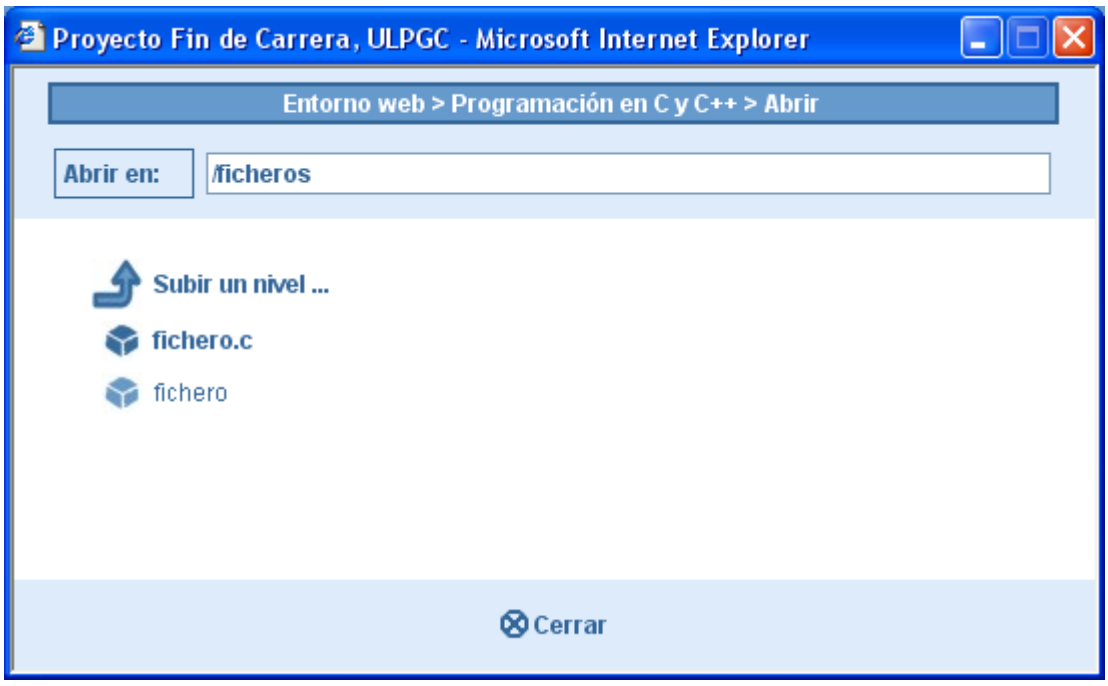


Figura 7.9 Ventana Opción Abrir

7.4.3 Guardar

Para que se cree un nuevo fichero, no sólo basta con editar su contenido sino que éste debe ser posteriormente salvado. La opción *guardar* salva los cambios presentados en el fichero mostrado en ese momento en el campo de contenido del *marco de presentación,* con el mismo nombre que tiene dicho fichero y el cual se presenta en el campo fichero del

mismo marco. Si este campo fichero esta vacío, es decir, si el fichero es nuevo y no ha sido salvado nunca, la opción *Guardar* realizará la misma función que la opción *Guardar como*.

7.4.4 Guardar como

Esta operación tiene una funcionalidad parecida a la de Guardar. La diferencia que presentan es que, en esta opción, se crea un nuevo fichero, mientras que *Guardar*, sólo modifica el contenido de uno ya creado con anterioridad.

La operación de *Guardar como* abre una nueva ventana de navegación, la cual consta de tres partes. La parte superior de la ventana, además de mostrar la operación escogida, es decir, a parte de indicar que se ha seleccionado la operación de *Guardar como*, también presenta un campo de texto, no modificable por el usuario, y cuyo contenido irá variando según la parte media de la misma ventana. La zona media muestra el contenido del directorio de trabajo del usuario. Al igual que en la opción *Abrir*, el contenido puede estar formado por tres tipos diferentes de elementos, los directorios, los ficheros ejecutables y el resto de ficheros. En el caso de *Guardar como*, los únicos elementos que presentan enlaces son los directorios. Estos enlaces permiten al usuario ver el contenido del directorio seleccionado en la misma zona de la ventana, a la misma vez que, en la parte superior de ella, se va mostrando la ruta del directorio seleccionado. Esto le va indicando al usuario la ruta donde se creará el nuevo fichero, cuyo contenido será el que presente en ese momento el campo de contenido mostrado en el *marco de presentación*. En la parte inferior de la ventana se muestra un campo de texto donde el usuario introduce un nombre con el que quedará creado el archivo en la aplicación.

Téngase en cuenta que no se podrá crear un fichero con el mismo nombre que uno ya existente ni tampoco que contenga caracteres especiales. Para controlar este tipo de errores, la aplicación muestra una ventana informativa indicando el tipo de error que se ha producido.

Del mismo modo que en la opción de *Abrir*, si una vez el usuario selecciona la operación de *guardar como* y se abre la nueva ventana, decide no guardar ningún fichero,

puede pulsar con el ratón sobre el enlace de *cerrar* situado en la parte inferior de dicha ventana, forzando el cierre de la misma y haciendo que el contenido del marco de presentación no sea salvado, y por consiguiente, que no se cree ningún archivo nuevo.

La figura 7.10 muestra cómo se guardaría el fichero del apartado 7.4.1 con el nombre de "prueba.c", dentro de un directorio llamado "pruebas".

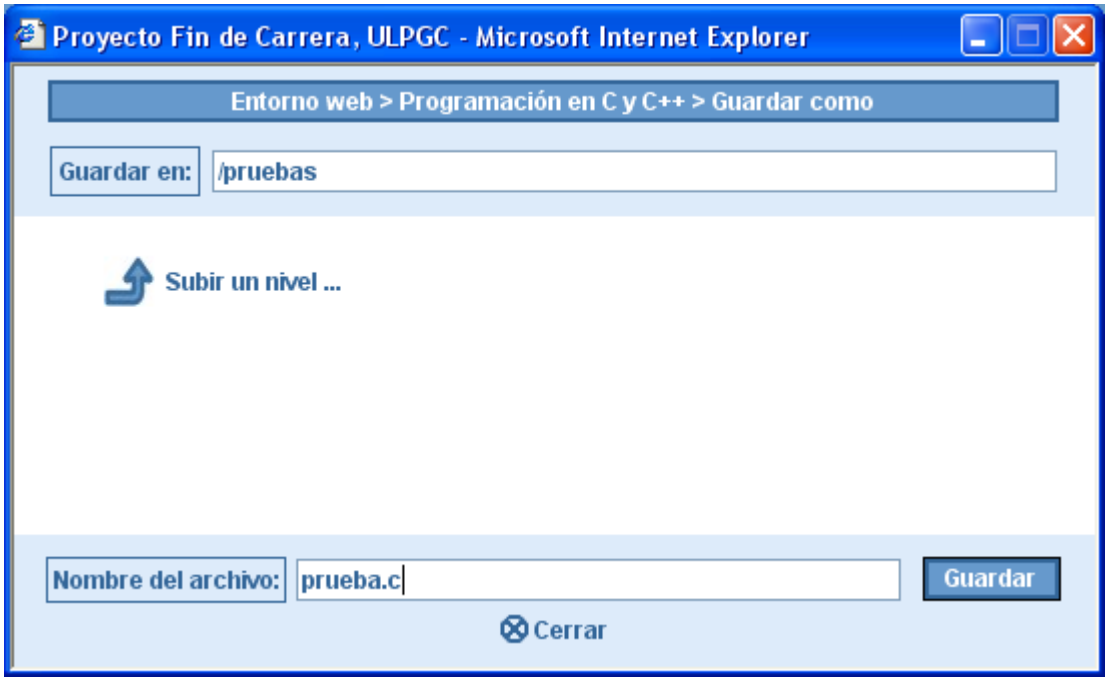


Figura 7.10 Ventana Opción Guardar Como

7.4.5 Eliminar

Al igual que un usuario puede crear tantos archivos como desee dentro de la aplicación, también puede eliminarlos. Para eliminar un fichero, el usuario debe previamente abrirlo en el campo de contenido del *marco de presentación*, bien mediante el árbol desplegable presentado en el *marco de navegación*, o bien mediante la opción de *Abrir* situada en el *marco de operaciones*. Abierto el fichero, el usuario debe hacer clic sobre la operación de *Eliminar,* presentándose una pequeña nota informativa, mostrada en la figura 7.11, donde tendrá la posibilidad de confirmar que desea eliminar el fichero, o por

el contrario, donde cancelará la operación de eliminación. Si el usuario hace clic sobre *Aceptar*, el fichero y su ejecutable, si éste existiese, serán automáticamente borrados.

Cuando se elimina el archivo, se actualiza, de forma automática, el contenido del *marco de navegación*, mostrando en el árbol desplegable, el contenido del directorio de trabajo del alumno sin el fichero eliminado. En caso de haber cancelado la operación, el fichero no es eliminado y permanece abierto en el *marco de presentación*.

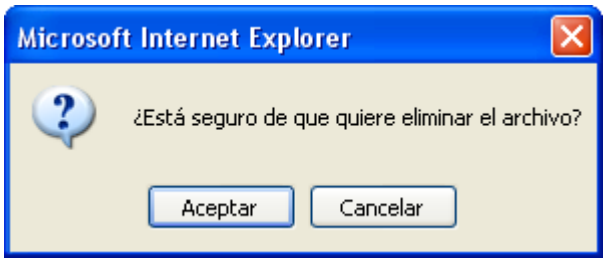


Figura 7.11 Ventana de confirmación de eliminación

7.4.6 Compilar

Para que un programa desarrollado bajo el lenguaje de programación de C o C++ pueda ser ejecutado, el programa debe ser previamente compilado puesto que es el fichero compilado el que verdaderamente se manda a ejecutar. La aplicación, *"Entorno Web para la Realización de Prácticas de Programación en C y C++"*, contiene opciones que realizan tanto la compilación como la ejecución, simplemente haciendo clic en los enlaces oportunos y bajo una serie de condiciones descritas en este manual.

Después de haber creado un fichero con extensión *.c o .cpp* dentro de la aplicación, éste se puede mandar a compilar. Para ello, se debe tener abierto dicho fichero en el *marco de presentación* y una vez abierto, el usuario simplemente debe hacer clic sobre el enlace de la opción *Compilar* del *marco de operaciones*. Esta opción abre una nueva ventana donde se muestra el resultado de compilación. Si no existe ningún error, se crea el fichero ejecutable asociado a él, con el mismo nombre que el fichero mandado a ejecutar pero sin extensión y situado en el mismo directorio. Notar que este nuevo fichero ejecutable es visible para el usuario desde el momento de su creación como si de cualquier otro fichero

se tratara, aunque no su contenido. En caso de existir error, el fichero ejecutable no es creado y el usuario puede observar los motivos del error en la nueva ventana. También se presenta el enlace de *Cerrar* que permite al usuario el cierre de dicha ventana. En la figura 7.12 se muestra un ejemplo con el resultado de la compilación del archivo "prueba.c".

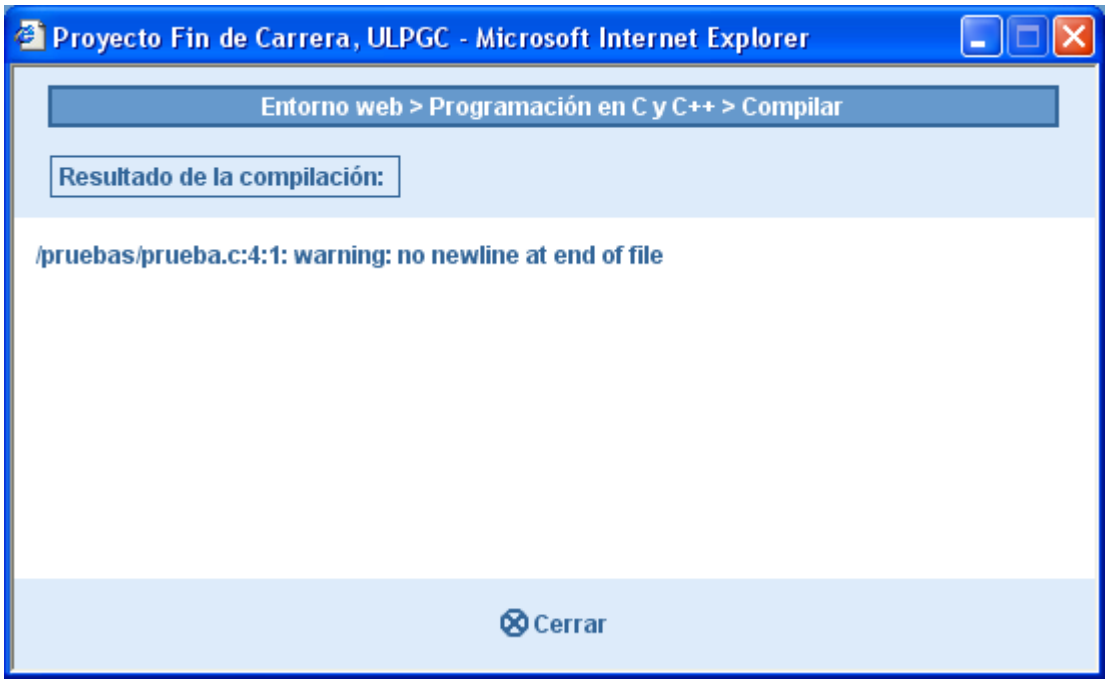


Figura 7.12 Ventana Opción Compilar

7.4.7 Ejecutar

Todo fichero compilado puede ser mandado a ejecutar. Para poder ejecutar un programa dentro de la aplicación *"Entorno Web para la Realización de Prácticas de Programación en C y C++"*, sólo deben cumplirse dos requisitos. El primero es que el fichero con extensión *.c o .cpp* tiene que haber sido compilado correctamente y por consiguiente, tener un fichero ejecutable asociado a él, y el segundo es que el fichero con extensión *.c o .cpp* debe estar abierto en el *marco de presentación* a la hora de mandar a ejecutarlo.

Si se cumplen las dos condiciones anteriores, el usuario puede mandar a ejecutar el programa. Para ello, debe hacer clic sobre el enlace de *Ejecutar* mostrado en el *marco de*

operaciones. Esta opción abre una nueva ventana fragmentada en tres partes, tal y como se muestra en la figura 7.13.

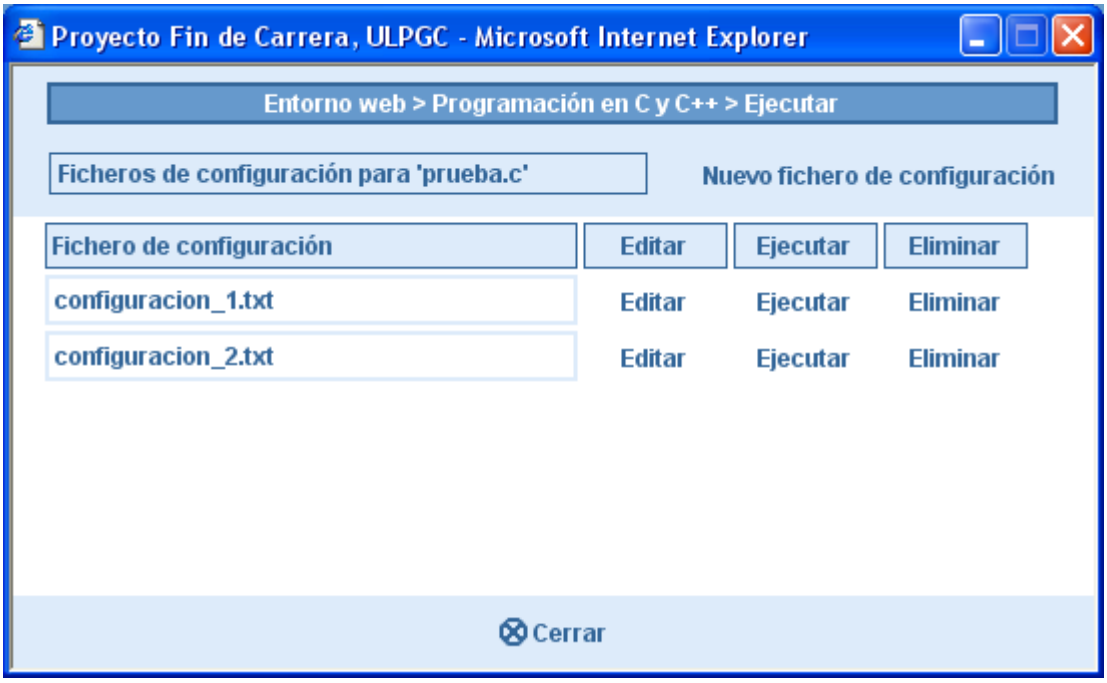


Figura 7.13 Ventana Opción Ejecutar

La parte superior especifica la opción escogida de ejecutar y presenta un enlace que permite crear un nuevo fichero de configuración. Un fichero de configuración es aquel fichero donde se almacena toda la información que el programa necesita para su ejecución. En él, el usuario podrá introducir, mediante un formulario, un nombre para el fichero, los parámetros necesarios para el programa, los cuales deben ir separados por espacios en blanco unos de otros, el nombre de un fichero de entrada en caso de que el programa lea la entrada de un fichero, el nombre de un archivo de salida donde almacenar el resultado de la ejecución y por último, un *timeout* especificando el tiempo máximo, en segundos, de espera para la ejecución del mismo. La figura 7.14 muestra la ventana con el formulario correspondiente para insertar un nuevo fichero de configuración.

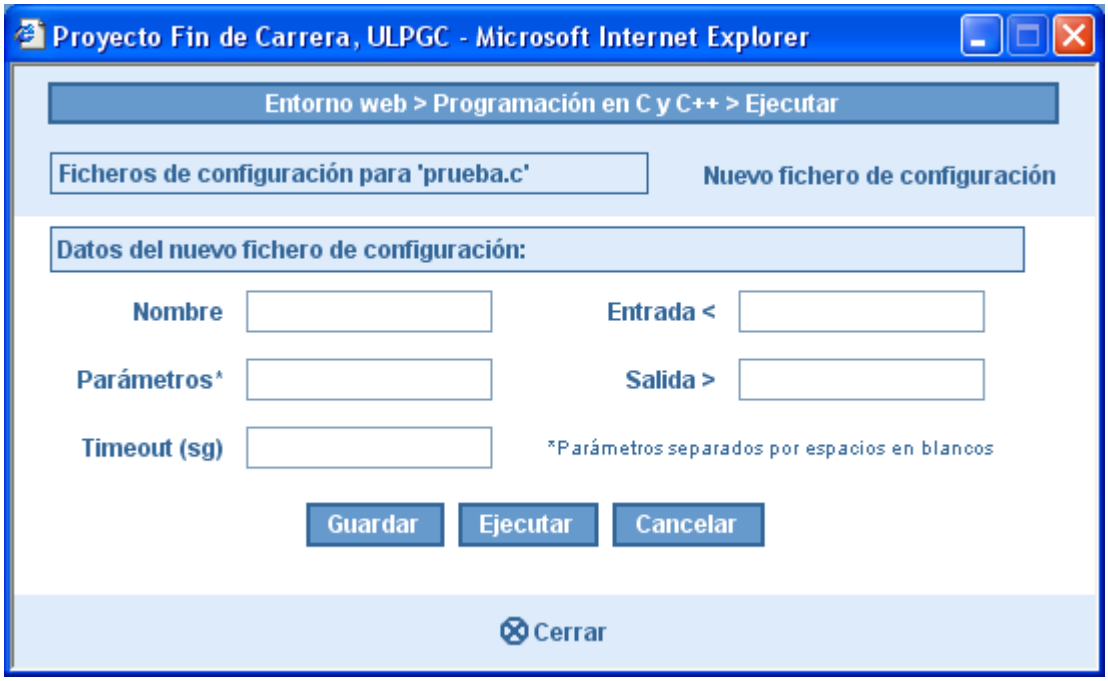


Figura 7.14 Formulario Nuevo Fichero de Configuración

Si el usuario selecciona el botón *Guardar*, el nuevo fichero de configuración es creado con los valores introducidos y la ventana de la opción ejecutar es cerrada. Si, por el contrario, selecciona el botón *Ejecutar*, no sólo se crea el nuevo fichero, sino que se manda a ejecutar el programa con la información introducida. El botón *Cancelar*, limpia los cambios del formulario que no hayan sido salvados.

En la zona media de la ventana correspondiente a la opción ejecutar, tal y como presenta la figura 7.13, se muestra el listado de los ficheros de configuración existentes, creados por el usuario, y tres enlaces por cada uno de ellos, *Editar*, *Ejecutar* y *Eliminar*.

El enlace *Editar* muestra un formulario, igual al mostrado en la figura 7.14 pero sin el campo nombre, donde se pueden modificar los valores del fichero de configuración, dependiendo de la necesidad del programa a ejecutar. Los tres botones presentados, *Guardar*, *Ejecutar* y *Cancelar*, tienen la misma funcionalidad que en el caso de crear uno nuevo, a excepción de que, en esta ocasión, sólo se modifica uno ya existente.

El enlace *Ejecutar*, al igual que los botones *Ejecutar*, lanza la ejecución del programa con los datos contenidos en el fichero de configuración y la aplicación actúa de la misma forma que el punto anterior. El enlace *Eliminar* elimina el fichero de configuración de la lista.

En la parte inferior de la ventana se muestra un enlace que permite al usuario el cierre de la ventana.

Una vez, el usuario manda a ejecutar el programa, si los datos del fichero de configuración no permiten su correcto funcionamiento, se presentará un mensaje de error como se muestra en la figura 7.15.

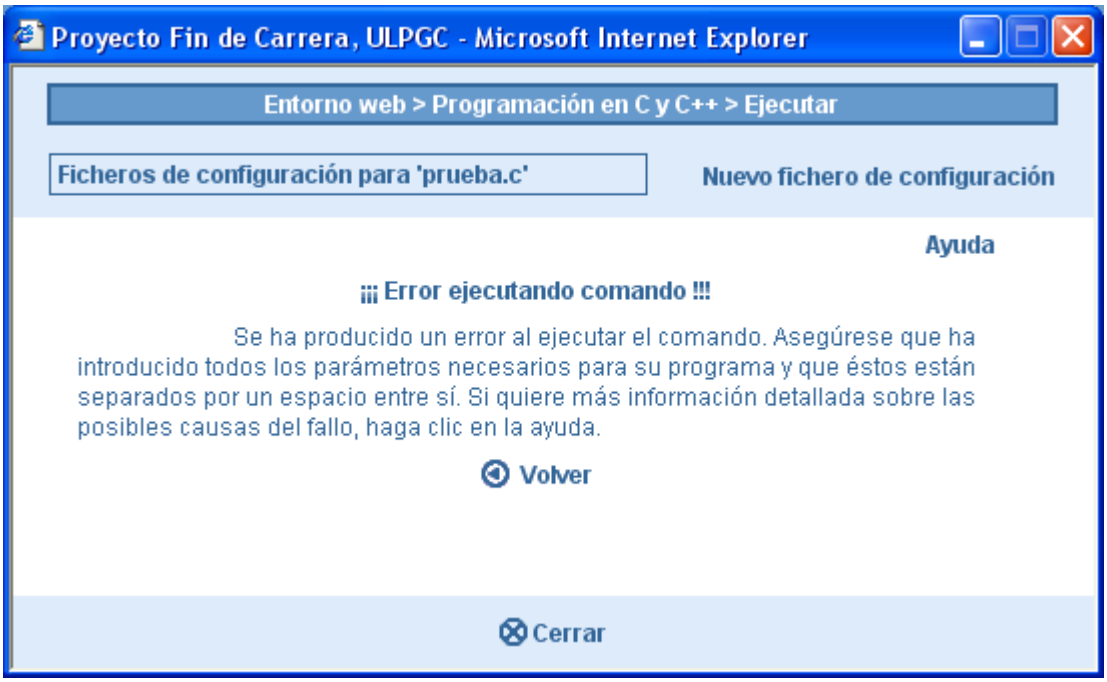


Figura 7.15 Mensaje de error ejecutando programa

Antes de mandar a ejecutar un programa, es conveniente que el usuario sepa que la ejecución se realiza de forma remota. Cuando un programa es mandado a ejecutar, la ventana de operación de ejecutar permanece abierta hasta que la ejecución finalice. De esta forma, una vez se cierre esta ventana de forma automática, se estaría informando al usuario que la ejecución ha finalizado. Si pasados unos segundos, o el tiempo especificado en el *timeout*, según el fichero de configuración, la ventana continua abierta, sería muy probable

que el código del programa haya provocado que el servidor se quede de forma permanente ejecutando dicho programa. Una forma de comprobar si esto es así, es volver a la página principal de entorno del usuario y seleccionar cualquier otra opción del *marco de operaciones*. Si al seleccionar una de estas opciones se observa que la aplicación ha quedado colgada, se confirmaría la hipótesis. Cuando esto ocurra, el usuario debe volver a entrar en el sistema, abriendo una nueva ventana de navegación, de la misma manera que lo hizo en un principio y como se describe en los primeros puntos de este manual. En la nueva sesión, el usuario volverá a tener disponibles todas las operaciones de la aplicación, la cual consta de una opción, que se describe en el punto 7.4.9, que le permite saber cuáles de sus programas mandados a ejecutar han finalizado y cuáles permanecen aún vivos en el servidor provocando la situación descrita, y añade la posibilidad de finalizarlos. El usuario debe siempre terminar con todos los programas que tenga activos en el servidor.

A continuación, se describe una serie de ejemplos de ejecución de programas con todos los casos posibles, es decir, un ejemplo de ejecución simple, otro con entrada / salida, uno que maneje ficheros, otro con timeout y por último, uno que se quede colgado. Estos ficheros serán llamados “ejemplo1.c”, “ejemplo2.c”, “ejemplo3.c”, “ejemplo4.c” y “ejemplo5.c”, respectivamente, y serán guardados en la carpeta “pruebas”.

CÓDIGO DE “EJEMPLO1.C”:

```
main () {
    int a,b,c;
    if (a>b) { c=a; }
    else { c=b; }
}
```

CAMPOS DEL FICHERO DE CONFIGURACIÓN:

```
Salida = ‘salida1.txt’
```

CÓDIGO DE “EJEMPLO2.C”:

```
#include <stdio.h>
int main(int argc, char *argv[]) {
int parametro1, parametro2, suma;
```

```
    parametro1 = atoi(argv[0]);
    parametro2 = atoi(argv[1]);
    suma = parámetro1+parametro2;
    printf("La suma de %d + %d es = %d",parametro1,parametro2,suma);
}

CAMPOS DEL FICHERO DE CONFIGURACIÓN:
   Parámetros = 2, 3; Salida = 'salida2.txt'
```

```
CÓDIGO DE "EJEMPLO3.C":
#include <stdio.h>
main () {
    FILE *flujo;
    char cadena[80];
    if ((flujo=fopen("datos/datos.txt","w+"))==NULL) {
            printf("No se pudo abrir el fichero datos.txt\n");
            exit(1);
    }
    else {
           fgets(cadena,80,stdin);
           fputs(cadena,flujo);
    }
}

CAMPOS DEL FICHERO DE CONFIGURACIÓN:
   Entrada = 'datos/entrada.txt'; Salida = 'salida3.txt'
```

```
CÓDIGO DE "EJEMPLO4.C":
#include <stdio.h>
main() {
    int contador1=0;
    int contador2=1;
    while (contador1<contador2) {
        contador1++;
        contador2++;
    }
}
```

```
CAMPO DEL FICHERO DE CONFIGURACIÓN:
  Salida = ‘salida4.txt’; Timeout =’5’;
```

```
CÓDIGO DE “EJEMPLO5.C”:
#include <stdio.h>
main()
{
   int contador1=0;
   int contador2=1;
   while (contador1<contador2) {
      contador1++;
      contador2++;
   }
}

CAMPOS DEL FICHERO DE CONFIGURACIÓN:
  Salida = ‘salida5.txt’
```

7.4.8 Salidas

Visto en el apartado anterior del manual, uno de los parámetros del fichero de configuración es el nombre de un fichero de salida, en donde se almacena el resultado producido por el programa ejecutado, lo que implica que son creados al mismo tiempo que estos programas son ejecutados. Para consultar estos ficheros, el usuario debe hacer clic en la opción *Salidas* del *marco de operaciones*. Una vez seleccionada esta operación, se muestra una nueva ventana, figura 7.16, donde se listan todos los ficheros de salida, presentando cada uno de ellos dos enlaces, *Consultar* y *Eliminar*.

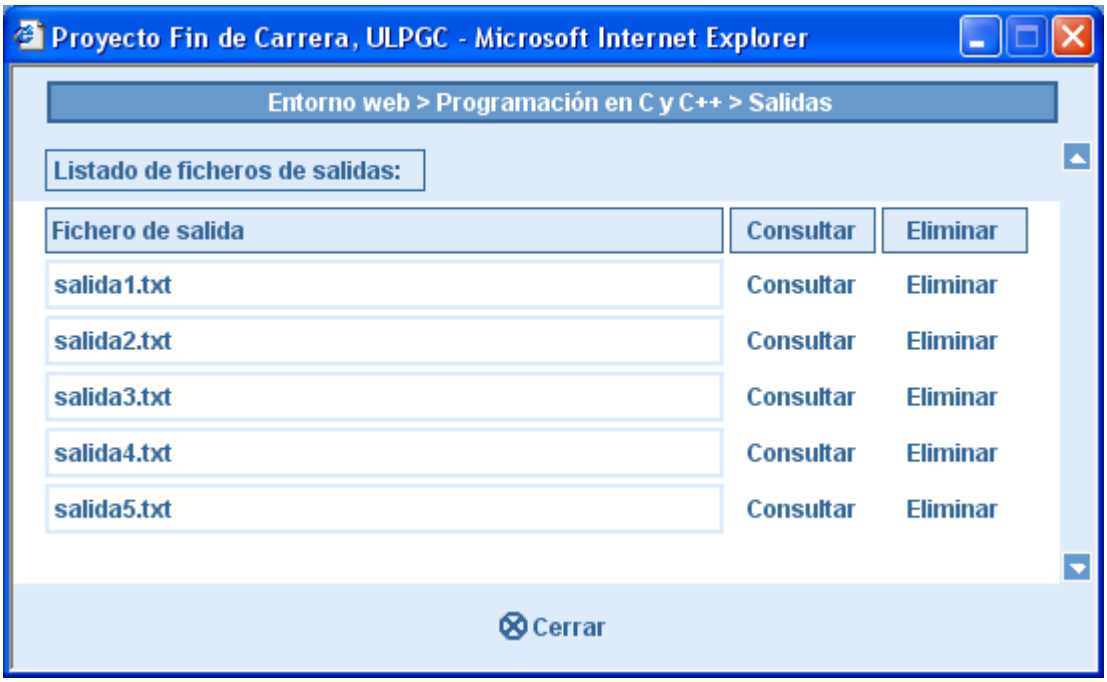


Figura 7.16 Ventana Opción Salidas

El enlace *Consultar* muestra, en la misma ventana, el contenido del fichero, con la posibilidad de poder volver al listado de ficheros de salida. El enlace *Eliminar*, elimina el fichero del listado.

7.4.9 Procesos

En el apartado 7.4.7, se explicó que existe la posibilidad de que haya programas lanzados por el usuario que estén en ejecución, de forma permanente en el servidor. La forma que tiene el usuario de saber si tiene algún programa en esta situación es haciendo clic en la opción de *Procesos* en el *marco de operaciones*.

En esta opción se abre una nueva ventana, figura 7.17, donde se muestra un listado con todos los programas que el usuario ha ordenado ejecutar. La información que se presenta se muestra en tres columnas diferentes, una indica el fichero mandado a ejecutar o proceso, otra muestra el estado del proceso y la última columna indica si el proceso ha finalizado o, por el contrario, puede matarse.

Cuando un proceso finaliza pueden darse dos casos, que el proceso haya terminado por si solo o que el proceso haya terminado por agotarse el tiempo especificado en el *timeout* del fichero de configuración, y por tanto, se haya obligado a su finalización. Esta es la información que se presenta en la columna de estado, mostrando en este campo la palabra *normal* para el primer caso, y *timeout* para el segundo. Si el proceso no ha finalizado el campo de estado indica *en ejecución*.

Existen dos posibles valores para el tercer campo, *finalizado*, en caso de haber terminado la ejecución del proceso en el servidor, o *matar*, indicando que el proceso continúa ejecutándose y permitiendo al usuario hacer clic sobre el enlace para terminar el proceso. Téngase en cuenta que cada vez que el usuario vea que tiene procesos en esta circunstancia debe pulsar en el enlace para no tener procesos ejecutándose de forma permanente en el servidor.

En la parte superior de la ventana existe un enlace que permite borrar el historial de procesos del usuario, con lo cual, sólo se mostrarían en el listado, todos los procesos que el usuario mande a ejecutar a partir de ese momento. Para que esto pueda ocurrir, el usuario no puede tener ningún proceso vivo, es decir, si al hacer clic en el enlace existiesen procesos en ejecución, se mostraría un mensaje de error y no se eliminaría el historial.

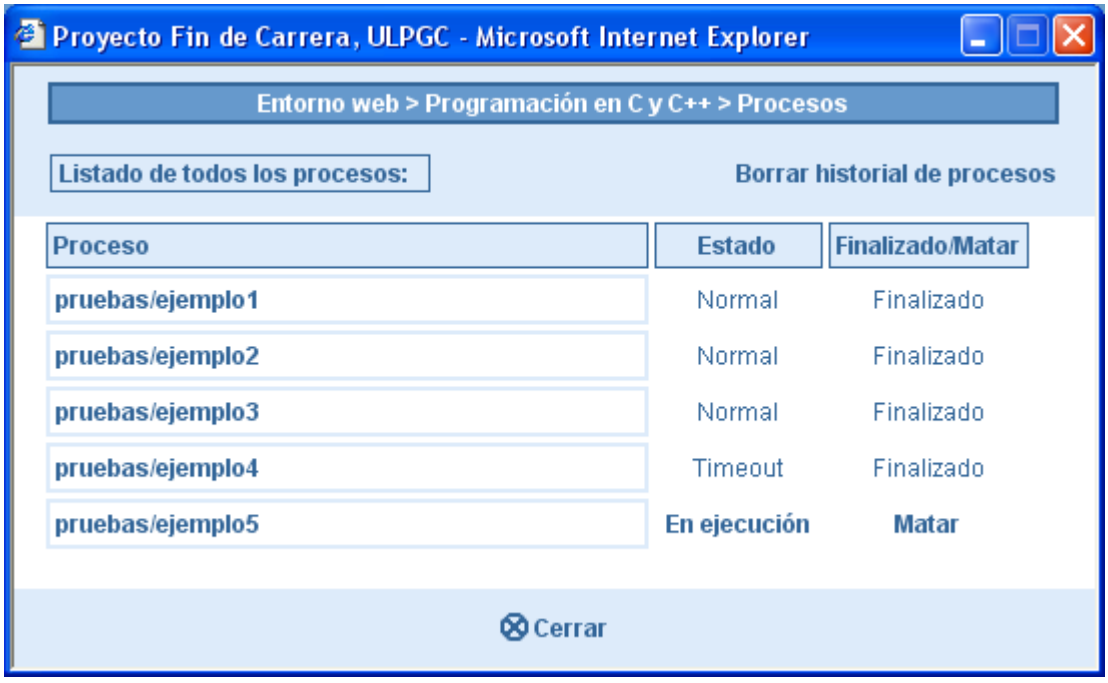


Figura 7.17 Ventana Opción Procesos

En la zona inferior de la ventana hay un enlace que permite al usuario el cierre de la misma.

7.4.10 Crear directorio

Cada usuario dado de alta en la aplicación tiene asignado un directorio de trabajo a partir del cual, el propio usuario irá creando sus ficheros. Si éste lo desea, puede crear dentro de su directorio de trabajo, nuevos directorios para su utilización. Para ello, debe hacer clic sobre el enlace *Crear directorio* en el *marco de operaciones*. Cuando el usuario selecciona esta opción, se abre una nueva ventana fragmentada en tres partes como muestra la figura 7.18.

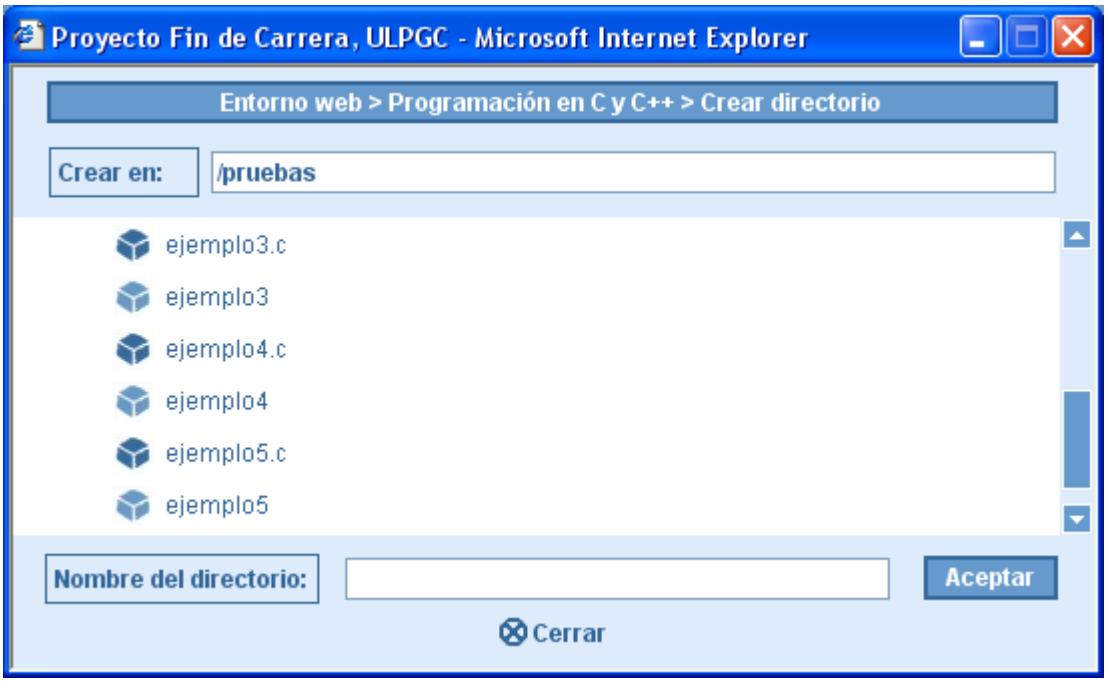


Figura 7.18 Ventana Opción Crear directorio

La zona superior de la nueva ventana muestra la opción escogida de crear directorio y un campo de texto no modificable por el usuario, cuyo valor irá variando según la zona media de la ventana. En la zona media se muestra el contenido del directorio de trabajo del usuario en el que se incluyen los tres tipos de elementos: directorios, ficheros ejecutables y el resto de ficheros. Al igual que en la opción de *Guardar como* del *marco de operaciones*,

en esta opción, sólo presentan enlaces los directorios, accediendo a través de ellos a su contenido. La ruta completa del directorio, al cual el alumno está accediendo, es el valor que irá adquiriendo el campo de la zona superior de la ventana y en donde se creará el nuevo directorio. En la parte inferior el usuario introduce el nombre con el que desea crear el directorio. En ella también el usuario podrá cerrar la ventana si decide no crear una nueva carpeta después de haber seleccionado esta opción.

7.4.11 Borrar directorio

De igual forma que el usuario puede crear directorios dentro de la aplicación, también puede eliminarlos. Para ello, el usuario debe hacer clic en el enlace *Borrar directorio* del *marco de operaciones*. Una vez selecciona esta opción, se presenta una nueva ventana formada por tres partes, tal y como se muestra en la figura 7.19.

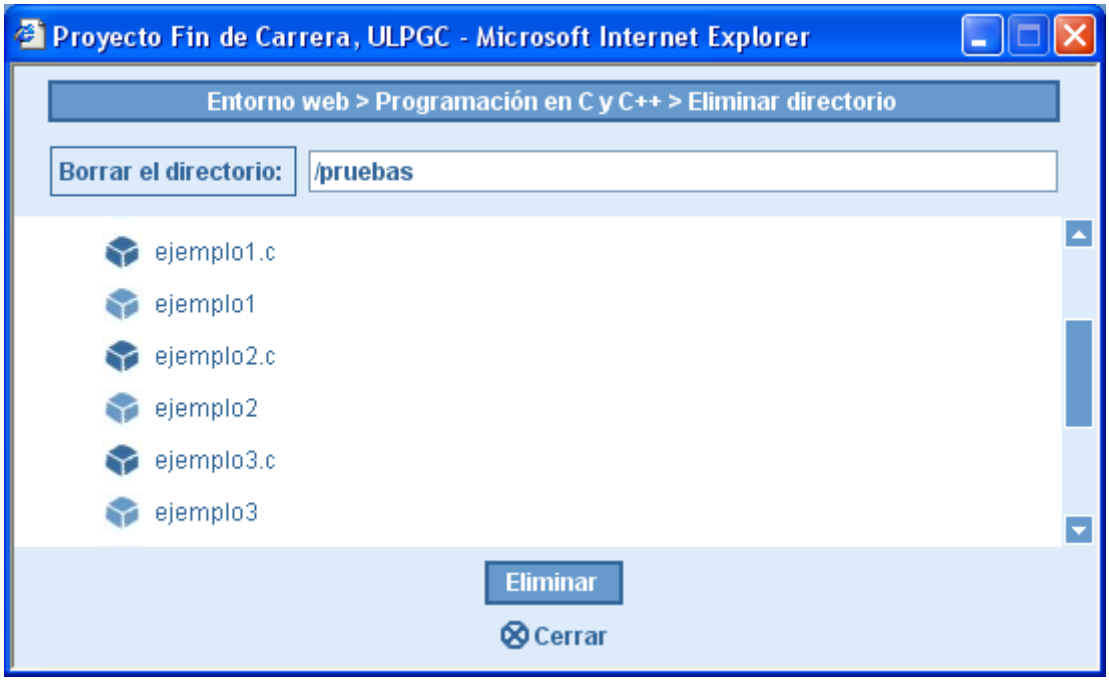


Figura 7.19 Ventana Opción Borrar directorio

En esta ventana, y al igual que en la opción de *Crear directorio*, el usuario podrá seleccionar el directorio que desea eliminar al mismo tiempo que ve su contenido y cuyo valor se irá actualizando en la zona superior de la ventana. Cuando el usuario hace clic en

Eliminar se muestra una ventana para verificar que desea borrarlo. Si pulsa con el ratón sobre *Aceptar*, el directorio y todo su contenido será eliminado y en la ventana se mostrará un mensaje, informando que la eliminación ha sido satisfactoria y se presentará un botón *Aceptar*. Una vez se pulse este botón la ventana se cerrará y el árbol desplegable del marco de navegación se actualizará de forma automática. En caso de *Cancelar*, el directorio no es eliminado y se cerrará la ventana de la operación.

Algo muy importante que debe tener en cuenta el usuario a la hora de eliminar un directorio, es que cuando se borra una carpeta, también se elimina todo su contenido.

8 Conclusiones

8.1 Objetivos cumplidos

El objetivo de este Proyecto Fin de Carrera era el de realizar una aplicación web que permitiese el control y seguimiento personalizado de la realización de las prácticas de la asignatura *Programación Avanzada* correspondiente a la Titulación de *Telemática*.

Con esta aplicación, los alumnos del laboratorio podrán llevar a cabo sus prácticas a través de Internet, donde podrán crear sus propios programas para, posteriormente, lanzarlos desde el cliente a compilar y ejecutar en el servidor. Asimismo, los resultados de estas ejecuciones pueden ser visibles desde la propia aplicación. Además, los alumnos tendrán acceso a un sistema de archivos Unix que les permitirá modificar y gestionar sus ficheros desde un entorno gráfico y sencillo.

También se ha desarrollado un sistema de publicación de listado de procesos, desde el cual, el alumno podrá observar todos sus procesos lanzados en el servidor y sus correspondientes estados. De igual forma, la aplicación facilita la opción de terminar con todos aquellos que se indiquen como vivos.

Por último, se dotó a la aplicación de un sistema de gestión de usuarios y control de acceso al sistema, ya que al ser una aplicación accesible desde Internet, había que restringir su entrada a usuarios no autorizados. El módulo de seguridad, incluido al principio de cada archivo, actuará permitiendo ver la información contenida o denegando su visualización, dependiendo de las comprobaciones realizadas en dicho módulo.

Para la realización de este Proyecto se han utilizado una serie de scripts CGI programados en lenguaje PHP y páginas HTML. Como servidor se utilizó un PC con el sistema Linux instalado, siendo éste un sistema robusto y potente para mantener servicios de Internet y, siendo el único sistema operativo con el que se garantiza el correcto funcionamiento de la aplicación. Como motor de Base de Datos se empleó MySQL, que es un servidor de Base de Datos válido para Linux.

Para la realización de la aplicación, ha sido necesario realizar un estudio de los siguientes temas:

- ***Sistema Operativo Linux:*** Linux es un sistema operativo gratuito, de libre distribución (bajo licencia GNU) y sumamente flexible, lo que ha hecho que se convierta en el más usado en los diferentes sistemas informáticos que forman Internet. Es muy robusto y son casi inusuales los virus que atacan a este sistema, en comparación con otras plataformas como Windows, donde aparecen decenas de virus cada día.
- ***Lenguaje HTML:*** El lenguaje HTML ha sido utilizado para la realización de las distintas páginas web de las que se compone la aplicación.
- ***Lenguaje PHP:*** El lenguaje PHP ha sido usado para insertar código dentro de las páginas HTML. Su integración con servidores web Apache es muy buena. Permite trabajar con funciones POSIX de C sobre UNIX y su interfaz a Bases de Datos MySQL funciona muy bien. A todo esto, hay que añadir que es gratuito.
- ***Servidor Web Apache:*** Apache es un servidor web empleado para lanzar las diferentes páginas web de la aplicación. Es un servidor flexible muy extendido. Es gratuito y de libre distribución bajo licencia GNU.
- ***Servidor de Base de Datos MySQL:*** Servidor de Base de Datos empleado para administrar la base de datos de la aplicación. Es un servidor de Base de Datos relacionales muy rápido, robusto y multiusuario. Es usado, principalmente, en Internet en conjunción con PHP. Es software libre, publicado bajo licencia GLP (GNU Public License).

Resumiendo, tanto el objetivo principal de este Proyecto, como otros fines anteriormente mencionados, han sido satisfechos con la realización de la aplicación. Cabe destacar la gran ventaja que tiene dicha aplicación, de permitir, no sólo la realización de prácticas de programación bajo los lenguajes de C y C++, sino también de cualquier otro lenguaje posible de compilar en el servidor, con simples cambios realizados en el código fuente. De esta forma, la aplicación no queda desfasada ante la aparición continua de nuevos lenguajes de programación.

9 Bibliografía

[1] Advanced Programming in the UNIX Environment
W. Richard Stevens
Addison-Wesley
INF 681.3.06 STE adv

[2] Unix Programación Avanzada
Francisco Manuel Márquez García
RA-MA Editorial, 1996
INF 681.3.06 MAR uni

[3] El lenguaje de programación C.
Brian W. Kernighan, Dennis M. Ritchie
Prentice Hall. Segunda edición.

[4] PHP 4. Serie práctica
Pedro Pablo Fábrega
Prentice Hall

[5] El protocolo HTTP
http://cdec.unican.es/libro/http.htm

[6] Servidor Apache
Mohamed J. Kabir
Anaya Multimedia S.A., 1999

[7] Sistema Operativo Linux
http://www.monografias.com/trabajos/solinux/solinux.shtml

[8] Código HTML
http://www.uca.es/manual-html/

[9] JavaScript
http://148.216.5.25/JavaScriptTut/cap2-1.htm

[10] CSS
http://www/terra.com/informatica/que-es/cascadin.cfm

[11] Base de Datos. Introducción
http://www.cs.us.es/cursos/bd-2002/Teoria/Tema2.pdf

[12] Definición de SQL. Lenguaje de Consulta Estructurado
http://descargas.clubdeprogramadores.com/tema.php?Id=116

[13] MySQL
http://www.programatium.com/mysql.htm

PLIEGO DE CONDICIONES

Pliego de condiciones

1 Introducción

El pliego de condiciones especifica los requisitos mínimos para el correcto funcionamiento de la aplicación. Estos requisitos se plantean desde dos perspectivas diferentes. Por un lado, se tendrán unas condiciones a cumplir por el cliente que utiliza la herramienta y, por otro lado, se tendrán otras condiciones para el servidor en donde ésta reside. Además, el pliego de condiciones describe los aspectos relativos a la instalación de la herramienta y a la configuración del sistema para su correcto funcionamiento e incluye cuestiones relativas a su mantenimiento.

2 Condiciones para el cliente

En este apartado se describen los requisitos que se deben cumplir para conseguir un perfecto funcionamiento de la aplicación en el equipo del cliente o usuario.

Para poder hacer uso de la herramienta, se debe disponer de un ordenador con acceso a Internet. Para el acceso a las páginas Web que la componen, se debe tener un navegador Internet Explorer 5.0 o Netscape Navigator 4.5 (o versiones superiores) y que en el navegador que se utilice, esté permitido el uso de las cookies. En caso de usarse otros navegadores, ha de asegurarse que sean capaces de soportar marcos, formularios y tablas, del mismo modo que deben ser compatibles con JavaScript 1.2.

La herramienta es independiente del sistema operativo o plataforma (Windows, Linux, Mac, etc.) pero debe tener una tarjeta gráfica que soporte una resolución de 1024x768 píxeles. En una resolución menor no se contempla el diseño gráfico de las distintas páginas que componen el sitio Web, mientras que la visualización de las mismas bajo resoluciones mayores, aunque no sea la más óptima, es igualmente satisfactoria. Del mismo modo, se recomienda emplear una paleta como mínimo de 256 colores, aunque el uso de otras mayores produce una mejor visualización de los gráficos.

Igualmente, existen otros requisitos hardware que establecen un mínimo de carácter general. Es posible que con ordenadores de características inferiores, el acceso también se produzca aunque, lógicamente, salvando las distancias. Un procesador Pentium 200 MHz o similar y 32 MB de memoria RAM son las condiciones básicas que se proponen.

3 Condiciones para el servidor

La herramienta ha sido desarrollada para ser instalada únicamente en máquinas UNIX/Linux, que deben cumplir con una serie de requisitos en cuanto al software instalado en ellas. Estos requisitos son los siguientes:

- Tener instalado un Servidor Web Apache 1.3.3 (UNIX).
- Tener instalado un módulo PHP para compresión de scripts PHP.
- Tener instalado un sistema gestor de Bases de Datos MySQL.

Del mismo modo que en el caso del cliente, también existen unos requisitos hardware mínimos recomendables. Un procesador Pentium 200 MHz o similar, memoria RAM suficiente para la instalación del sistema operativo y todo el software anteriormente mencionado, unidad de CD-ROM y más de 10 MB de espacio libre en disco duro, son las condiciones requeridas.

4 Instalación y mantenimiento

En este apartado se describirán todos los aspectos relativos a la instalación de la herramienta en el servidor y la configuración del sistema para su correcto funcionamiento. Se incluyen además, cuestiones relativas al mantenimiento de la herramienta.

4.1 Introducción

Todos los comandos, ubicaciones y archivos de configuración que se mencionen en el apartado de instalación y mantenimiento, suelen ser estándar y normalmente no varían de una distribución Linux a otra (Red Hat, SuSE, Caldera, etc.). En este sentido, es posible, aunque improbable, encontrar algunas diferencias entre un sistema y otro. Si se diera este caso, la persona encargada de la instalación y mantenimiento de la herramienta tendrá que salvar estas distancias adaptando la situación a su servidor en particular.

Debido al carácter técnico de la información que se proporciona en los siguientes apartados, su lectura y comprensión implican conocimientos acerca del sistema operativo Linux, único sistema operativo válido en el servidor para el funcionamiento de la aplicación. El punto de vista planteado cubre sólo aspectos relativos a la instalación y mantenimiento de la herramienta y no pretende ser, en ningún momento, un tutorial sobre Linux.

En caso de duda o necesidad de profundizar en ciertas cuestiones, se recomienda una lectura complementaria de la amplia bibliografía que se puede encontrar sobre el tema, tanto de forma escrita, como en la red.

4.2 Administrador del sistema

En capítulos pertenecientes a esta memoria, se han distinguido para la aplicación *Entorno web para la realización de prácticas de programación en C y C++* dos tipos de usuarios. Por un lado, los usuarios restringidos o alumnos, los cuales acceden sólo al

bloque de trabajo para la edición de programas, compilación, ejecución y control de sus propios procesos. Por otro lado, el administrador o gestor de la aplicación, el cual es el encargado del mantenimiento de la herramienta desde el punto de vista de acceso web.

Sin embargo, existe una última categoría, sólo mencionada en este capítulo, denominada *administrador del sistema*, que de una manera u otra, también accede a la herramienta. Este usuario podrá coincidir o no con el administrador de la aplicación pero, independientemente de este hecho, se trata de la persona encargada del mantenimiento de los servicios que el servidor proporciona y posee los privilegios y permisos necesarios para llevar a cabo su tarea.

En adelante, todas las tareas y cuestiones que se explican en este capítulo quedarán personalizadas bajo la figura de este usuario. Esto es debido a que la instalación y mantenimiento de la aplicación y los servicios relacionados implican el acceso a archivos que, por regla general, están reservados a un usuario de estas características, y no a un usuario común que posea cuanta en el sistema.

El administrador será el responsable de la instalación inicial, y a él se le presuponen de entrada:

- Conocimientos acerca del sistema operativo que reside en el servidor (Linux).
- Conocimientos acerca de la administración de cuentas y configuración de permisos en un sistema UNIX.
- Conocimientos acerca del mantenimiento, configuración e instalación de módulos de un servidor web (Apache en este caso).

4.3 Instalación de la aplicación

La instalación de la aplicación requiere completar una serie de pasos previos que son necesarios para el funcionamiento óptimo de la herramienta. Primeramente, instalado Apache en el servidor, se pasa a explicar cómo habría que modificar sus ficheros de configuración para, entre otros, especificar la ubicación donde residirán los archivos de la

herramienta. Posteriormente se instalan estos archivos y se configuran sus permisos. También, y mediante un usuario MySQL, se importará un fichero que contendrá la base de datos con la que trabajará la aplicación. Finalizados estos pasos, se pasará a explicar cómo editar los ficheros de configuración propios de la aplicación, en caso de así requerirlo. Por último, se podrá dar inicio a la herramienta.

4.3.1 Servidor Web Apache. Directivas

Apache es el servidor web que ha de utilizarse para las conexiones del cliente con el servidor durante el uso habitual de la aplicación. Apache se considera instalado y en funcionamiento en el sistema y, en este apartado, se hará un resumen de las características que se consideran más importantes en su configuración, en especial aquellas relacionadas con la herramienta objeto de este trabajo.

La configuración del Apache se realiza mediante tres archivos de texto en los que se especifican todas las directivas que rigen su funcionamiento. Estos archivos suelen encontrarse en */etc/http/conf*, aunque su localización puede variar de un sistema a otro y sus nombres son: *acces.conf*, *httpd.conf*, *srm.conf*.

Las directivas de configuración se encuentran repartidas entre estos archivos aunque en algunos sistemas se pueden encontrar agrupadas todas juntas en *httpd.conf*. En el CD de instalación, y bajo el directorio */ApacheConfiguracion*, se encuentra este archivo, a modo de ejemplo, para una instalación de Apache en un sistema Linux cualquiera.

A continuación se explican y comentan aquellas directivas de configuración que se consideran más importantes en relación a esta herramienta.

En el archivo *httpd.conf* se encuentran las directivas *User* y *Group*. A través de ellas se establece el usuario y grupo (previamente creados) que van a ser usados para el proceso de Apache. Como se puede ver, los valores especificados son *laboratorio*.

```
# User/Group: The name (or #number) of the user/group to run apache
# as.
```

```
#  . On SCO (ODT 3) use "User nouser" and "Group nogroup".
#  . On HPUX you maynot be able to use shared memory as nobody, and
#  the suggested workaround is to create a user www and use that
#  user.
#  NOTE that some kernels refuse to setgid (Group) or semctl
#  (IPC_SET) when the value of (unsigned) Group is above 60000;
#  don´t use Group nobody on these systems!
#
User laboratorio
Group laboratorio
```

A través de la directiva *SeverName*, se especifica el nombre de dominio del servidor. Este nombre será el que constituya la dirección de acceso web a nuestra herramienta. Hay que tener en cuenta que no se puede poner un nombre cualquiera. El que sea especificado ha de estar dado de alta en el servidor DNS como una máquina que está accesible con su dirección IP asociada.

```
# ServerName allows you to set a host name which is sent back to
# clients for your server if it´s different than the one the
# program would get (i.e. use "www" instead of the host´s real
# name).
#
# Note: You cannot just invent host names and hope they work. The
# name you define here must be a valid DNS name for your host. If
# you don´t understand this, ask your network administrator.
# If your host doesn´t have a registered DNS name, enter its IP
# address here. You will have to access it by its address (e.g.,
# http://123.45.67.89/) anyway, and this will make redirections
# work in a sensible way.
#
ServerName servidor
```

Mediante el uso de la directiva *DocumentRoot* se especifica el directorio en donde se sitúan las páginas web disponibles, cuando la petición web del cliente está dirigida a un usuario activo de la máquina. En este caso el directorio que contiene las páginas de la herramienta es */home/ProgramacionAvanzada/public_html*

```
# DocumentRoot: The directory out of which you will serve your
```

```
# documents. By default, all requests are taken from this
# directory, but symbolic links and aliases may be used to point to
# other locations.
#
DocumentRoot /home/ProgramacionAvanzada/public_html
```

A través de la directiva *DirectoryIndex* se especifica el archivo que Apache proporciona al cliente cuando la petición web está encaminada a un directorio, sin especificar ninguna página en concreto. Por ejemplo, si el cliente realiza la petición: *http://servidor.telematica.ulpgc.es/programacion/,* en realidad se está accediendo a la página *http://servidor.telematica.ulpgc.es/programacion/index.php.*

```
# DirectoryIndex: Name of the file or files to use as a pre-written
# HTML directory index. Separate multiple entries with spaces.
#
DirectoryIndex index.php
```

La directiva *AddType* es una manejadora de archivos mediante la cual se le indica a Apache que procese las páginas con extensión .php.

```
# AddType allows you to tweak mime.types without actually editing
# it, or to make certain files to be certain types.
#
AddType application/x-httpd-php .php .php3
```

La directiva *ServerAdmin* almacena una dirección de correo electrónico donde se enviarán los problemas ocurridos con el servidor.

```
# ServerAdmin: Your address, where problems with the server should
# be e-mailed. This address appears on some server-generated pages,
# such as error documents.
#
ServerAdmin webmaster@servidor.telematica.ulpgc.es
```

Las directivas anteriores han de ser consideradas en conjunto y son las más importantes en relación al funcionamiento de la herramienta. Si por algún casual, el

administrador hubiera considerado conveniente usar otros valores para éstas, tendrá que tenerlo en cuenta y efectuar los cambios necesarios.

4.3.2 Instalación de archivos

Los archivos de la aplicación se distribuyen con el CD de instalación y están en formato comprimido. Para tener acceso a ellos se ha de montar la unidad de CD-ROM en el sistema de archivos del servidor. El montaje de esta unidad dependerá de cómo esté configurado el servidor, ya que es posible que ésta venga especificada en el momento del arranque de la máquina. Suponiendo que la unidad no esté especificada de antemano, las instrucciones a seguir son: insertar el CD-ROM en la unidad de CD, comprobar que el directorio */cdrom* existe y escribir el siguiente comando:

```
mount /dev/hdd –t iso9660 –r /cdrom
```

Una vez la unidad está montada se puede comprobar que está correctamente en el directorio */cdrom* con el comando disk free (df). A partir de ese momento, se podrá acceder al CD-ROM como a un directorio normal en */cdrom*.

Posiblemente, y debido a los privilegios que requiere la instrucción *mount*, el administrador tendrá que hacer uso de su propia cuenta para ejecutar esta instrucción.

Por otro lado, suponiendo que la unidad venga especificada en el arranque, la instrucción para el montaje de la unidad sería:

```
mount /cdrom
```

En el CD de instalación, los archivos se encuentran ubicados en el directorio */ProgramacionAvanzada*, y son los siguientes:

```
public_html.tar.gz
programacion.tar.gz
database.tar.gz
```

Para proceder a la instalación habrá que situarse en el directorio */home/ProgramacionAvanzada* e introducir las siguientes instrucciones:

```
tar xvfz /ProgramacionAvanzada/public_html.tar.gz
tar xvfz /ProgramacionAvanzada/programacion.tar.gz
tar xvfz /ProgramacionAvanzada/database.tar.gz
```

Los archivos se descomprimirán y se crearán todos los subdirectorios necesarios, quedando una estructura de directorios como la que se muestra en la figura 1.

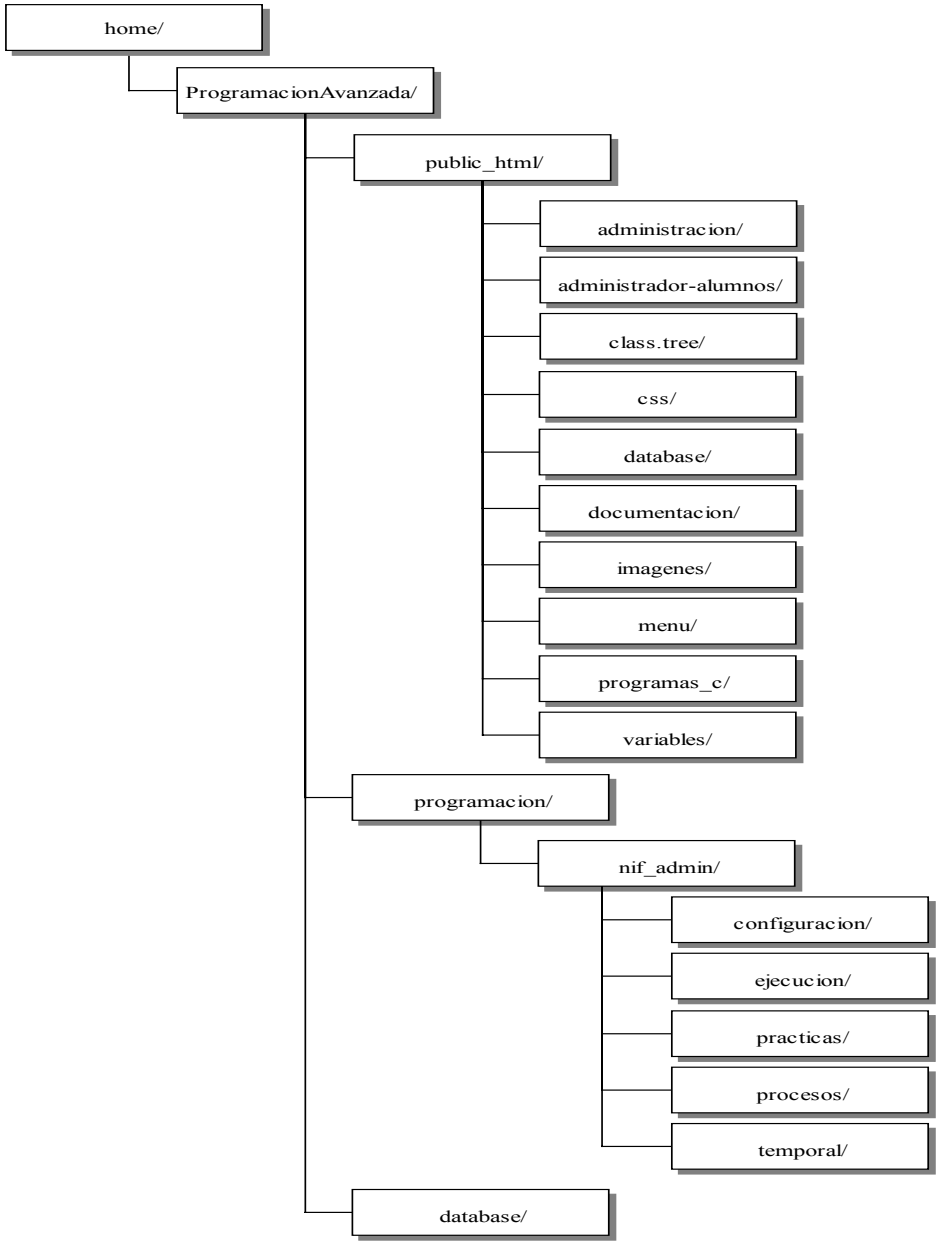


Figura 1 Estructura de directorios

Terminada la instalación, la unidad de CD-ROM puede ser desmontada con la instrucción:

```
umount /cdrom
```

4.3.3 Configuración de permisos

En el apartado anterior vimos como quedaba la estructura de directorios una vez instalados los archivos de la herramienta. Algunas de estas carpetas y ficheros necesitan tener todos los permisos disponibles para todos los usuarios del sistema y son:

```
/home/ProgramacionAvanzada/public_html/programas_c/miscript
/home/ProgramacionAvanzada/public_html/programas_c/padre
/home/ProgramacionAvanzada/public_html/programas_c/padre2
/home/ProgramacionAvanzada/public_html/programas_c/padre_timeout
/home/ProgramacionAvanzada/public_html/programas_c/padre_timeout2
/home/ProgramacionAvanzada/programacion/nif_admin/configuración/
/home/ProgramacionAvanzada/programacion/nif_admin/ejecución/
/home/ProgramacionAvanzada/programacion/nif_admin/practicas/
/home/ProgramacionAvanzada/programacion/nif_admin/procesos/
/home/ProgramacionAvanzada/programacion/nif_admin/temporal/
```

Los grupos de permisos (lectura: r, escritura: w y ejecución: x) tienen que estar todos activados. Para ello, se ejecutará el comando:

```
chmod 0777 Nombre del Directorio/Fichero
```

Después de esto, el administrador del sistema debería ver en cada uno lo siguiente:

```
drwxrwxrwx      en el caso de directorios
-rwxrwxrwx      en el caso de archivos
```

4.3.4 Importación de la base de datos

Para arrancar la aplicación, ésta debe leer unos contenidos mínimos de una base de datos. La estructura de esta base de datos, junto con sus valores, pueden ser encontrados en el directorio */home/ProgramacionAvanzada/database* dentro del archivo *datos*. El administrador debe importar este fichero o, en su defecto, ejecutarlo. Para ello, previamente debe, como usuario de MySQL, crear una base de datos llamada *ProgramacionAvanzada* y realizar la operación bajo ella.

4.3.5 Ficheros de configuración de la aplicación

Existen tres ficheros de configuración que permiten, al administrador del sistema, realizar ciertos cambios que afectan, tanto a la instalación de la aplicación, como a la presentación de la misma.

En el directorio */home/ProgramacionAvanzada/public_html/variables* se encuentran estos tres ficheros llamados *BaseDatos.inc*, *Directorios.inc* y *Titulos.inc*. Para el correcto funcionamiento de la herramienta, estos archivos deben guardan perfecta concordancia con los valores descritos en los apartados anteriores. A continuación se detallan cada uno de estos ficheros.

***BaseDatos.inc*.** Este fichero almacena los valores relacionados con la base de datos. Estos valores son cuatro:

- *host*: nombre de la máquina donde se ubica la base de datos.
- *log_mysql*: usuario MySQL.
- *pass_mysql*: contraseña del usuario MySQL.
- *BD*: nombre de la base de datos (ProgramacionAvanzada).

***Directorios.inc*.** En este fichero se almacenan los valores relacionados con las rutas de los archivos de la herramienta. En él, el administrador sólo debe editar los valores de dos de las variables contenidas. Estas dos variables son:

- *directorio_raiz*: nombre del directorio de donde colgarán las carpetas personales de los usuarios dados de alta en el sistema. Si el administrador decide modificar este valor, deberá editar la estructura de directorios representada en la figura 3.1, cambiando el nombre del directorio */programacion/* por el nuevo valor introducido.
- *directorio_web*: nombre del directorio que almacena las páginas de la aplicación. De igual manera, si se edita este valor, la estructura de directorios debe ser modificada.

***Titulos.inc*.** Este fichero sólo almacena dos variables que modifican, a modo de título, la representación estética del bloque de trabajo y del bloque de administración.

- *titulo*: variable que contiene el título correspondiente al bloque de trabajo.
- *titulo_adm*: variable que contiene el título correspondiente al bloque de administración.

4.3.6 Inicio de la aplicación

Una vez que se hayan completado los pasos de los apartados anteriores (configuración de Apache, instalación de archivos, etc.), la herramienta ya se encuentra disponible para uso final por el cliente (sólo por el administrador de la aplicación). En el directorio */home/ProgramacionAvanzada/database*, el administrador del sistema puede encontrar el archivo que contiene la clave que emplea la aplicación para la autentificación de usuario válido. Este archivo se llama *gestor* y proporciona unos valores por defecto para la conexión del usuario de gestión.

La dirección web que da acceso a la herramienta es:

```
http://servidor.telematica.ulpgc.es/programacion/
```

A partir de este momento, el uso de la herramienta se le atribuye al administrador web, el cual será el responsable de su mantenimiento desde el punto de vista de acceso web. *Como primer paso, se le recomienda el cambio de la clave de gestor de la aplicación*.

4.4 Mantenimiento

Durante el funcionamiento normal de la herramienta, la figura del administrador del sistema no es necesaria, ya que las tareas de gestión son realizadas siempre por el administrador de la aplicación. Sin embargo, y raramente, ante la existencia de problemas graves (fallos internos en el servidor ajenos a la aplicación, cortes en el suministro eléctrico, etc.), la figura del administrador del sistema es indispensable para restablecer el correcto funcionamiento de la herramienta.

El mantenimiento de la aplicación implica la realización de aquellas tareas relacionadas con el control de los errores producidos, solución de problemas de acceso a los archivos y de control de procesos en el servidor.

En los sistemas Linux, las aplicaciones y servicios disponibles se ejecutan en forma de procesos. En general, los procesos aceptan los datos de entrada a través de la *entrada estándar* (normalmente el teclado) y generan la salida de datos a través de la *salida estándar* (normalmente el terminal o monitor). Existe además una tercera vía llamada *salida estándar de error* (normalmente un archivo de texto) en donde se registran los errores producidos durante la ejecución del proceso.

Estas entradas y salidas son, por regla general, redireccionables, es decir, el usuario bajo el que se ejecuta el proceso puede variar las entradas y salidas por defecto, como por ejemplo, redireccionar la salida estándar a un archivo de texto en vez de al terminal.

El proceso principal bajo el que se ejecuta la aplicación *Entorno web para la realización de prácticas de programación en C y C++*, es el servidor Apache. Apache direcciona los errores producidos durante el funcionamiento de la herramienta a un archivo

de texto que se suele denominar *error_log* y que, normalmente, se encuentra situado en el directorio */var/log/httpd*.

Estos errores son excepcionales y, en ocasiones, se producirán debido a ausencias de archivos que siempre han de estar presentes. En otras ocasiones, los errores se producen por imposibilidad de creación o modificación de archivos. Esto puede ser debido a fallos en el sistema de archivos del servidor, o bien a la modificación accidental de permisos y pertenencia de los directorios.

PRESUPUESTO

Presupuesto

1 Introducción

El presupuesto de este Proyecto se ha realizado, tomando como referencia para el cálculo de las tarifas honorarias, la publicación *"Propuesta de baremos orientativos para el cálculo de honorarios"*, publicada por el Colegio Oficial de Ingenieros Técnicos de Telecomunicaciones (COITT).

Esta propuesta establece que, para trabajos tarifados por tiempo empleado, se aplique la siguiente fórmula:

$$\mathbf{H = H_n 58 + H_e 63}$$

Siendo:

- *H = Honorarios*
- *Hn = Horas en jornada normal*
- *He = Horas fuera de la jornada normal*

Estos honorarios tendrán una reducción en función del número de horas, aplicando los siguientes coeficientes de reducción:

- Hasta 36 horas; *Coeficiente C = 1*
- Exceso de 36 horas hasta 72; *Coeficiente C = 0,90*

- Exceso de 72 horas hasta 108; *Coeficiente C = 0,80*
- Exceso de 108 horas hasta 144; *Coeficiente C = 0,70*
- Exceso de 144 horas hasta 180; *Coeficiente C = 0,65*
- Exceso de 180 horas hasta 360; *Coeficiente C = 0,60*
- Exceso de 360 horas hasta 510; *Coeficiente C = 0,55*
- Exceso de 510 horas hasta 720; *Coeficiente C = 0,50*
- Exceso de 720 horas hasta 1080; *Coeficiente C = 0,45*
- Exceso de 1080 horas; *Coeficiente C = 0,40*

El tiempo empleado en la realización de la herramienta se considera de la siguiente forma:

- El Proyecto se inicia desde las primeras recopilaciones de datos en septiembre de 2002 hasta la finalización de la herramienta en diciembre de 2003.
- Los periodos en los cuales se exceda de 8 horas diarias de trabajo contabilizarán las horas excedentes como extraordinarias.
- Los días laborables se contabilizarán de 4 semanas al mes y 5 días a la semana.

Para realizar el presupuesto de este Proyecto se han determinado las siguientes fases:

- ✓ *Fase de Análisis*
- ✓ *Fase de Ejecución*

2 Fase de análisis

La *Fase de análisis* del Proyecto se ha desarrollado a lo largo de 6 meses, con una media de 4 horas en horario laboral de tarde. Durante este tiempo se llevó a cabo las siguientes etapas:

- ***Información:*** Incluye el tiempo empleado en la recopilación de información necesaria para adquirir los conocimientos mínimos para la elaboración de la aplicación. En esta etapa cabe destacar, principalmente, el estudio del lenguaje PHP, del Sistema de Gestión de Bases de Datos MySQL, del Control de Procesos en UNIX y, finalmente, de los lenguajes HTML y JavaScript, y utilización de las hojas de Estilo CSS.
- ***Definición:*** Definición funcional de cada una de las partes del Proyecto, donde se especifica el objetivo del mismo, explicando el oficio de cada parte y sus interrelaciones.
- ***Análisis:*** Tiempo en el que se consigue definir los módulos necesarios para realizar la aplicación. Examinando las funcionalidades y acercando esta definición a la codificación de la misma. Se detallan los módulos, las funciones, los procedimientos, la interfaz gráfica, etc., consiguiendo plasmar con claridad los algoritmos de la aplicación.

En la tabla 1 se muestra el importe por honorarios obtenido:

Descripción	Tiempo	Precio/Hora	Coeficiente C	Total
Información	160	-	-	0,00 €
Definición	80	-	-	0,00 €
Análisis	240	58,00 €	0,60	8.352,00 €
SUBTOTAL 1				**8.352,00 €**

Tabla 1 Costes obtenidos en la Fase de Análisis

3 Fase de ejecución

La *Fase de ejecución* del Proyecto ha tenido una duración de 12 meses, con una media diaria de 4 horas en horario laboral de tarde. Durante ese tiempo se llevó a cabo las siguientes etapas:

- ***Codificación:*** Período de tiempo en el que se realiza el desarrollo de los programas. Se implementan los algoritmos definidos en el análisis. Se implantan en el servidor y se comprueba su funcionamiento, lo que lleva consigo, dentro de esta etapa, otra etapa de depuración del código diseñado.
- ***Memoria:*** Tiempo empleado en la redacción de la memoria del Proyecto, la cual es la exposición detallada del mismo, recopilación total de datos, estudios y cálculos realizados.

En la tabla 2 se muestra el importe por honorarios obtenido.

Descripción	Tiempo	Precio/Hora	Coeficiente C	Total
Codificación	640	58,00 €	0,50	18.560,00 €
Memoria	320	58,00 €	0,60	11.136,00 €
SUBTOTAL 2				**29.696,00 €**

Tabla 2 Costes obtenidos en la Fase de Ejecución

4 Material utilizado

Para la realización y ejecución de este Proyecto, se han utilizado distintos recursos hardware y software. Se ha supuesto un período de amortización de 3 años durante, aproximadamente 250 días al año y trabajando 8 horas diarias. Por lo que, en total serán 6000 horas de uso. Para calcular el gasto por material, partimos del coste del material y calculamos el coste por horas de uso de dicho material (dividiendo el coste total por 6000 horas de uso). Por último, se multiplica el coste por hora de uso con las horas de uso reales.

En la tabla 3 obtenemos los siguientes gastos:

Material	Cantidad	Precio/Unidad	Coste/Hora	Horas/Uso	Amortizado
Pentium IV 1.5 GHz	1	1.100,00 €	0,18 €	1440	259,20 €
Windows XP Profess.	1	-	-	-	0,00 €
Internet Explorer 6.0	1	-	-	-	0,00 €
Servidor Apache 1.3	1	-	-	-	0,00 €
PHP	1	-	-	-	0,00 €
PHPEd 3.0	1	-	-	-	0,00 €
MySQL	1	-	-	-	0,00 €
MySQL-Front 2.4	1	-	-	-	0,00 €
Office XP	1	798,00 €	0,13 €	320	41,6 €
CorelDraw 10	1	-	-	-	0,00 €
SmartDraw 5.0	1	-	-	-	0,00 €
WinSCP3	1	-	-	-	0,00 €
SUBTOTAL 3					**300,80 €**

Tabla 3 Costes obtenidos del material utilizado

5 Presupuesto final

El presupuesto final del Proyecto viene dado por la suma de los subtotales, con lo que el coste total se muestra en la tabla 4.

Período	Importe total (Euros)
Coste Fase de Análisis	8.352,00 €
Coste Fase de Ejecución	29.696,00 €
Coste Material	300,80 €
Total	**38.348,80 €**

Tabla 4 Coste total del Proyecto

El presupuesto total del Proyecto *"Entorno web para la realización de prácticas de programación en C y C++"*, asciende a **TREINTA Y OCHO MIL TRESCIENTOS CUARENTA Y OCHO CON OCHENTA EUROS (38.348,80 €).**

Fdo.: Dª. Ana Teresa Pulido Cabrera
Ingeniera Técnica de Telecomunicaciones

ANEXOS

Anexos

ANEXO 1 Descripción applet JavaScript

En este apartado se hace una descripción completa del applet JavaScript que ha sido usado en la aplicación. Para ello, se explicarán todos los parámetros de configuración, así como los argumentos necesarios para su construcción. También se incluye un ejemplo práctico (a nivel de código HTML y PHP) que muestra el aspecto final del applet.

A1.1 Applet TreeView

La inserción del applet *TreeView* en una página HTML genera una estructura de árbol desplegable, a modo de estructura de navegación, que permite la carga de otras páginas al pulsar sobre los nodos que componen dicho árbol.

Este applet ha sido usado para generar el entorno de navegación de la aplicación en el bloque de trabajo. A continuación se resumen los parámetros y características de este applet.

A1.2 Uso del applet

Para generar el árbol jerárquico, se crea un objeto correspondiente a la clase *Tree*, definida en el fichero *"class.tree.php"*. En esta clase se definen varios atributos y métodos,

los cuales se encargan de producir el aspecto final del árbol desplegable en la página HTML, según los argumentos recibidos.

A1.2.1 Argumentos

Los nodos del applet se generan a partir de *cadenas de nodo*. Cada cadena está compuesta de los argumentos necesarios que definen el nodo. Los argumentos se encuentran separados por el carácter coma ",". Las *cadenas de nodo* están compuestas de cinco argumentos cada una: *Padre*, *Texto*, *Enlace*, *Icono_Abierto* e *Icono_Cerrado*.

- ***Padre:*** El *padre* proporciona la ruta del nodo, es decir, indica a qué otro nodo está anidado, pudiendo ser éste, el propio nodo raíz. El nivel de anidamiento de nodos puede ser todo lo profundo que se desee. Siempre que se defina a un nodo como padre de otro, el segundo nodo queda como hijo del primero.
- ***Texto:*** El *texto* trata de una cadena de caracteres, a modo de nota informativa, que se muestra en el navegador, definiendo al nodo en cuestión.
- ***Enlace:*** El *enlace* proporciona la URL hacia la página o recurso que se desea mostrar cuando el nodo es seleccionado. El URL tiene que ser completo, es decir, no se puede especificar una dirección relativa a un parámetro URL base. Del mismo modo, el enlace no puede utilizarse para especificar marcos HTML en donde mostrar la página asociada al nodo.
- ***Icono_Abierto:*** El *Icono_Abierto* facilita la imagen que se muestra a la izquierda del nodo y la ruta de acceso al fichero ".gif" o ".jpg" de esta imagen. La imagen se muestra cuando el nodo está abierto (seleccionado o expandido).
- ***Icono_Cerrado:*** El *Icono_Cerrado* proporciona la imagen que se muestra a la izquierda del nodo y la ruta de acceso al fichero ".gif" o ".jpg" de esta imagen. La imagen se muestra cuando el nodo está cerrado (sin seleccionar o sin expandir).

A1.2.2 Parámetros

Los parámetros son usados para añadir las *cadenas de nodo* al applet y para definir los valores que especifican cómo ha de mostrase la estructura del árbol. Éstos se definen en el fichero *"class.tree.php"* y son los siguientes:

- ***Tree_basefrm:*** Indica el marco donde se muestra el contenido de la página. Este marco debe estar definido previamente antes de la ejecución del applet. Si no es así, se utiliza el marco por defecto "_top", con lo cual, la carga de la página asociada al nodo reemplazaría al applet.
- ***Usetextlinks:*** Indica si el *texto*, definido como argumento en el apartado anterior, presenta el mismo enlace que la imagen del nodo (TRUE) o, por el contrario, no presenta enlace ninguno (FALSE).
- ***Startallopen:*** Especifica si el árbol comienza en su forma expandida (TRUE), o mostrando todos los nodos cerrados sin expandir (FALSE).
- ***Add_folder:*** Método de la clase *Tree*, encargado de mostrar el nodo, permitiendo a éste ser padre de otro.
- ***Add_document:*** Método de la clase *Tree*, que presenta el nodo, no pudiendo éste ser padre de ningún otro nodo, es decir, de este nodo no podrán colgar otros nodos.

A1.3 Ejemplo del applet

El ejemplo que se muestra a continuación no corresponde con el código implementado en la aplicación. Tan sólo se trata de mostrar con un ejemplo práctico, cómo se genera el árbol jerárquico a partir del applet JavaScript *TreeView*.

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0 Transitional//EN”>
<HTML lang=”es” xml:lang=”es”>
<HEAD>
<link href=”css/estilos.css” rel=”stylesheet” type=”text/css”>
</HEAD>
<BODY bgcolor=”#ddebfa”>
<?
  include (“class.tree/class.tree.php”);
  $what = “Applet JavaScript TreeView”;
  $tree = new Tree();
  $padre = $tree->open_tree($what,”URL1”);
  $$padre = $tree->add_folder($padre,”Carpeta 1”,”URL2”,”FolderOn.jpg”,”FolderOff.jpg”);
  $hijo1 = $$padre;
  $tree->add_document($hijo1,”Fichero 1”,”URL3”,”FileOn.jpg”,”FileOff.jpg”);
  $$padre = $tree->add_folder($padre,”Carpeta 2”,”URL4”,”FolderOn.jpg”,”FolderOff.jpg”);
  $hijo2 = $$padre;
  $tree->add_document($hijo2,”Fichero 2”,”URL5”,”FileOn.jpg”,”FileOff.jpg”);
  $tree->close_tree();
?>
</BODY>
</HTML>
```

El aspecto final del applet, una vez que se visualiza la página en el navegador, es el mostrado en la figura A1.1

Figura A1.1. Applet JavaScript TreeView

ANEXO 2 Listado de ficheros y funciones

En este apartado se muestran dos listados, un listado de ficheros y un listado de funciones. El primero de ellos presenta todos los archivos que han sido creados para la realización de la aplicación, resumiendo la funcionalidad de cada uno de ellos. Los ficheros han sido clasificados en diferentes tablas dependiendo del módulo al que corresponden. Por otro lado, el listado de funciones presenta las más destacadas funciones PHP y JavaScript, divididas en distintas tablas según el archivo donde han sido definidas.

A2.1 Listado de ficheros

Módulo de autentificación de usuario	
Fichero	**Descripción**
index.php	Controla el acceso a la aplicación a usuarios autorizados.

Bloque de administración	
Fichero	**Descripción**
Frames Administracion.php	Divide el bloque de administración en cuatro marcos diferenciados.
Baner Administracion.php	Presenta una serie de enlaces a diferentes opciones, propias del bloque de administración, habilitadas únicamente al administrador.
Enlaces Administracion.php	Presenta los enlaces a todas las opciones propias del bloque de administración, habilitadas únicamente al administrador.
Firma.php	Carga un enlace a la página de la EUITT.

Gestión de usuarios	
Fichero	**Descripción**
Listado.php	Muestra el listado de todos los alumnos junto a los datos personales de cada uno de ellos.
Nuevo1.php	Carga un formulario donde el administrador debe introducir los datos de un nuevo alumno.

Nuevo2.php	Da de alta a un nuevo alumno y crea una estructura de directorios asociada al nuevo usuario insertado.
Modificar1.php	Muestra el listado de todos los alumnos, permitiendo la selección de un único usuario para su modificación.
Modificar2.php	Carga un formulario con los datos del alumno previamente seleccionado, permitiendo la modificación de dichos valores.
Modificar3.php	Guarda los cambios realizados en la modificación de usuarios.
Eliminar1.php	Muestra el listado de todos los alumnos, permitiendo la selección para su eliminación. Permite la selección masiva.
Eliminar2.php	Elimina a los alumnos previamente seleccionados.
Cambiar Administrador1.php	Carga un formulario donde el usuario debe introducir su nombre de usuario y contraseña, comprobándose sus privilegios de gestor para poder realizar el cambio de administrador.
Cambiar Administrador2.php	Carga un formulario donde el usuario debe introducir los datos del nuevo administrador de la aplicación.
Cambiar Administrador3.php	Elimina al usuario administrador y crea al nuevo gestor de la aplicación.

Otras opciones de administración	
Fichero	**Descripción**
Interfaz Administrador1.php	Realiza el paso del bloque de administración al bloque de trabajo del propio administrador.
Interfaz Alumno1.php	Muestra el listado de todos los alumnos, permitiendo la selección de un único usuario para el acceso a su bloque de trabajo.
Interfaz Alumno2.php	Realiza el paso del bloque de administración al bloque de trabajo correspondiente a un usuario previamente seleccionado.

Bloque de trabajo	
Fichero	**Descripción**
Frames Administrador.php	Divide el bloque de trabajo del administrador en cuatro marcos diferenciados.
Baner Administrador.php	Presenta al usuario administrador una serie de enlaces a diferentes opciones, propias del bloque de trabajo.
Frames Alumno.php	Divide el bloque de trabajo de los alumnos en cuatro marcos diferenciados.
Baner Alumno.php	Presenta a los alumnos una serie de enlaces a diferentes opciones, propias del bloque de trabajo.
Menu Administrador.php	Presenta los enlaces a todas las opciones propias del bloque de trabajo, habilitadas a todos los usuarios registrados en la aplicación.
Principal Administrador.php	Carga el editor de texto correspondiente al bloque de trabajo. También realiza diferentes operaciones con dicho contenido.

Gestión de directorios	
Fichero	**Descripción**
Frames CrearDir.php	Divide la ventana de navegación, correspondiente a la opción de crear un directorio, en tres marcos diferenciados.
Crear Directorio1.php	Informa al usuario de la opción escogida y carga un formulario con un solo campo cuyo valor corresponde con la ruta donde el nuevo directorio va a ser creado.
Crear Directorio2.php	Muestra el contenido del directorio de trabajo del usuario. Por cada subdirectorio encontrado presenta un enlace desde el que se accede a su contenido y especifica donde va a ser creado el nuevo directorio.
Crear Directorio3.php	Carga un formulario donde el usuario debe especificar el nombre del nuevo directorio a crear.
Frames BorrarDir.php	Divide la ventana de navegación, correspondiente a la opción de eliminar un directorio, en tres marcos diferenciados.

Borrar Directorio1.php	Informa al usuario de la opción escogida y carga un formulario con un solo campo cuyo valor corresponde con el directorio a eliminar.
Borrar Directorio2.php	Muestra el contenido del directorio de trabajo del usuario. Por cada subdirectorio encontrado presenta un enlace desde el que se accede al contenido de dicha carpeta y especifica el directorio a borrar.
Borrar Directorio3.php	Habilita el botón que permite la eliminación del directorio.
DirEliminado.php	Informa que el directorio ha sido eliminado satisfactoriamente y refresca el árbol de navegación correspondiente al bloque de trabajo.

Gestión de archivos	
Fichero	**Descripción**
FramesAbrir.php	Divide la ventana de navegación, correspondiente a la opción de abrir un archivo, en tres marcos diferenciados.
Abrir1.php	Informa al usuario de la opción escogida y carga un formulario con un solo campo cuyo valor corresponde con el directorio donde se sitúa el usuario.
Abrir2.php	Muestra el contenido del directorio de trabajo del usuario. Por cada elemento encontrado presenta un enlace, permitiendo la apertura de un fichero o mostrando el contenido de la carpeta seleccionada.
Abrir3.php	Carga un enlace que provoca el cierre de la ventana de navegación.
Frames GuardarComo.php	Divide la ventana de navegación, correspondiente a la opción de guardar como, en tres marcos diferenciados.
GuardarComo1.php	Informa al usuario de la opción escogida y carga un formulario con un solo campo cuyo valor corresponde con el directorio donde se va crear el nuevo archivo.
GuardarComo2.php	Muestra el contenido del directorio de trabajo del usuario. Por cada subdirectorio encontrado presenta un enlace, desde el que se accede a su contenido y especifica donde va a ser creado el nuevo archivo.
GuardarComo3.php	Carga un formulario donde el usuario debe especificar un nombre

	para el nuevo archivo a crear.
Eliminar.php	Elimina el archivo que se encuentra abierto en el editor de texto.

Módulo de compilación	
Fichero	**Descripción**
Frames Compilar.php	Divide la ventana de navegación, correspondiente a la opción de compilar, en tres marcos diferenciados.
Compilar1.php	Informa al usuario de la opción escogida de compilar.
Compilar2.php	Realiza la compilación del archivo abierto en el editor de texto de la aplicación y muestra el resultado.
Compilar3.php	Carga un enlace que provoca el cierre de la ventana de navegación.

Módulo de ejecución	
Fichero	**Descripción**
Frames Ejecutar.php	Divide la ventana de navegación, correspondiente a la opción de ejecutar, en tres marcos diferenciados.
Ejecutar0.php	Informa al usuario de la opción escogida y carga un enlace desde el que se accede a la inserción de un nuevo fichero de configuración.
Ejecutar1.php	Presenta un listado de todos los ficheros de configuración creados por el usuario. Junto a cada fichero, carga tres enlaces, editar, ejecutar y eliminar.
Ejecutar2.php	Carga un formulario cuyos campos permiten la creación/edición de un fichero de configuración.
Ejecutar3.php	Ordena la ejecución del fichero abierto en el editor de texto con los datos contenidos en el fichero de configuración previamente seleccionado.
Ejecutar5.php	Elimina el fichero de configuración previamente seleccionado.
Ejecutar6.php	Carga un enlace que provoca el cierre de la ventana de navegación correspondiente a la opción de ejecutar.

Gestión de procesos	
Fichero	**Descripción**
Frames Procesos.php	Divide la ventana de navegación, correspondiente a la opción de procesos, en tres marcos diferenciados.
Procesos1.php	Informa al usuario de la opción escogida y carga un enlace que provoca la eliminación del listado de procesos.
Procesos2.php	Presenta un listado de todos los procesos lanzados por el usuario, especificando el estado de cada uno de ellos
Procesos3.php	Carga un enlace que provoca el cierre de la ventana de navegación.
Eliminar Expediente.php	Elimina el listado de todos los procesos lanzados por el usuario.
MatarProceso.php	Ordena “matar” a un proceso previamente seleccionado y el cual se ha indicado como “vivo”.

Otras opciones de trabajo	
Fichero	**Descripción**
Frames Ejecutados.php	Divide la ventana de navegación, correspondiente a la opción de salidas, en tres marcos diferenciados.
Ejecutados0.php	Informa al usuario de la opción escogida de salidas.
Ejecutados1.php	Presenta un listado de todos los ficheros de salida creados mediante la ejecución de programas. Junto a cada fichero, carga dos enlaces, consultar y eliminar.
Ejecutados2.php	Muestra el contenido de un fichero de salida previamente seleccionado.
Ejecutados3.php	Realiza la eliminación de un fichero de salida previamente seleccionado.
Ejecutados6.php	Carga un enlace que provoca el cierre de la ventana de navegación.
FramesCambiar Password.php	Divide la ventana de navegación, correspondiente a la opción de cambiar contraseña, en tres marcos diferenciados.
Cambiar Password0.php	Carga un enlace que provoca el cierre de la ventana de navegación.

Cambiar Password1.php	Informa al usuario de la opción escogida.
Cambiar Password2.php	Carga un formulario donde el usuario debe introducir su actual contraseña, la nueva contraseña deseada y una corroboración de la nueva contraseña.
Cambiar Password3.php	Realiza el cambio de contraseña del usuario.

Árbol de navegación	
Fichero	**Descripción**
foldertree.php	Construye el árbol jerárquico de navegación, perteneciente al bloque de trabajo, a partir del directorio de trabajo del usuario.

Ficheros con definiciones de funciones destacadas	
Fichero	**Descripción**
conexion.php	Contiene la definición de la función PHP, encargada de realizar la conexión con el servidor de base de datos.
class.tree.php	Contiene la definición de la clase Tree(), utilizada en la construcción del árbol de navegación perteneciente al bloque de trabajo.
Funciones.js	Contiene las definiciones de todas las funciones JavaScript correspondientes al menú presentado en el bloque de trabajo.
Funciones Ficheros.inc	Contiene las definiciones de todas las funciones PHP relativas al manejo de directorios y archivos.

A2.2 Listado de funciones

conexion.php	
Función	**Descripción**
ConsultarBD()	Realiza la conexión con el servidor de base de datos y ejecuta la consulta recibida por parámetro.

foldertree.php	
Función	**Descripción**
CrearArbol()	Lee el contenido del directorio de trabajo del usuario y lo muestra en forma de árbol desplegable para su navegación.

Funciones.js	
Función	**Descripción**
nuevo()	Vacía el contenido del editor de texto preparándolo para la inserción de un nuevo fichero.
abrir()	Abre la ventana correspondiente a la opción de abrir.
guardar()	Guarda el contenido, presente en el editor de texto, en el fichero que se encuentra abierto.
guardarcomo()	Abre la ventana correspondiente a la opción de guardar como.
eliminar()	Abre la ventana correspondiente a la opción de eliminar. Posteriormente, llama a la función JavaScript nuevo() y refresca el árbol jerárquico de navegación.
compilar()	Abre la ventana correspondiente a la opción de compilar.
ejecutar()	Abre la ventana correspondiente a la opción de ejecutar.
salidas()	Abre la ventana correspondiente a la opción de salidas.
en_ejecucion()	Abre la ventana correspondiente a la opción de procesos.
creardir()	Abre la ventana correspondiente a la opción de crear un directorio.
borrardir()	Abre la ventana correspondiente a la opción de borrar un directorio.

FuncionesFicheros.inc	
Función	**Descripción**
LeerFichero()	Devuelve el contenido del fichero cuyo nombre y ruta son recibidos por parámetros.
GuardarFichero()	Guarda en el fichero indicado como primer parámetro, el texto recibido como segundo parámetro.
EliminarFichero()	Elimina el fichero cuyo nombre y ruta son recibidos por parámetros.
EliminarDirectorio()	Elimina, de forma recursiva, el directorio recibido por parámetro.