“Generador de aplicaciones Web”

25 de junio de 2002

Índice General

Capítulo 1

Introducción

1.1 ¿ Qué es el WAG?

El Generador de Aplicaciones Web o Web Aplication Generator (WAG) es una herramienta que permite facilitar el proceso de creación de aplicaciones Web que impliquen el acceso a una bases de datos. Está diseñada para que, al ser instalada e integrada con un servidor Web, se consiga dar a un grupo de usuarios, poco o nada familiarizados en la programación en Internet, un servicio de hospedaje de aplicaciones Web que permitan publicar datos almacenados en una base de datos. Por una lado, los usuarios podrán crear sus propias aplicaciones Web sin necesidad de tener conocimientos sobre ningún lenguaje de programación, pudiendo actualizar los datos que serán publicados en las aplicaciones Web desde su hogar, haciendo uso del navegador. Por otro lado, eliminaŕa al Webmaster de gran parte del esfuerzo de administración y mantenimiento de las aplicaciones y de las bases de datos pertenecientes a dichos usuarios.

1.2 ¿Qué utilidad tiene el WAG?

Como ya es bien conocido por todos, Internet ha experimentado una extraordinaria evolución en los últimos años. Internet ha dejado de ser un proyecto dentro del ámbito militar para reducir los riesgos de quedar aislado ante un ataque militar, para convertirse en un sistema de comunicación casi imprescindible para un gran número de personas. El gran auge de la Internet, se ha debido a que los servicios que esta proporciona a sus usuarios han llegado a ser más seguros y fáciles de utilizar, junto con la aparición de la World Wide Web (www), que es un servicio de consulta de documentos de hipertexto. Este servició ha sido el que finalmente, Internet goce de la popularidad que actualmente tiene. No obstante, Internet proporciona en la actualidad gran cantidad de servicios, entre los que podemos destacar:

- Correo electrónico: Permite comunicarse con cualquier persona que tenga acceso a Internet, de un modo rápido y económico.

- Transferencia de archivos (FTP): Permite acceder a los sistema de archivos de cualquier computador de acceso público.
- Foros de discusión: Existe un gran número de grupos de discusión, donde se puede participar.
- Noticieros: Puedes recibir información de forma automática.
- World Wide Web: Proporciona la posibilidad de acceder a documentos de hipertexto.
- Comercio electrónico: Puedes realizar compras a través de la Web.
- Servicios financieros a través de la Web.
- Búsqueda de información de cualquier tipo.
- etc.

Paralelamente al auge de la Internet, se ha producido una gran evolución en el desarrollo y mantenimiento de las páginas Web. Inicialmente, la gran mayoría de páginas Web eran de tipo estático, es decir, el usuario sólo podía visualizar la información que estas contenían. La tendencia actual es la de crear páginas Web dinámicas (también conocida como aplicación Web), las cuales permiten al usuario una mayor interacción con el sistema y ofrecen una información mejor dirigida, escogida y elaborada. Además las páginas Web dinámicas son más fácilmente instaladas y, sobre todo, mantenidas.

Por contra, una mayor interacción por parte del usuario, conlleva la necesidad de reunir y procesar la información proporcionada por él y emitir un resultado. Esto implica que el desarrollo de una aplicación Web lleve asociada una capa de programación. Esto quiere decir, que un usuario que quiera publicar datos almacenados en una base de datos a través de una aplicación Web, por simple que la aplicación sea, es necesario una capa de programación. El problema es que no todos los usuarios disponen de los conocimientos suficientes para crear su propia aplicación.

La herramienta WAG surge como una alternativa para paliar este problema, estando orientada, principalmente, a aquellas personas que carecen de los conocimientos necesarios para crear, por sí mismos, su propia aplicación Web (o servir en la mayor ayuda posible).

WAG proporciona una aplicación Web que permita publicar los datos que un usuario posea almacenados en una base de datos propia, y todo esto sin la necesidad de programar. El usuario podrá actualizar los datos publicados en la aplicación haciendo uso de la propia aplicación, desde su propio hogar a través de un navegador. Básicamente, un usuario debe proporcionar al sistema WAG el modo en que están estructurados los datos, así como el conjunto de datos que desea publicar.

WAG también permite adaptarse a usuarios con distintos grados de familiarización con los métodos de desarrollo de aplicaciones Web.

Desde el punto de vista del Webmaster, WAG habilita un procedimiento sencillo de administración y mantenimientos, tanto de las aplicaciones Web, como

de las bases de datos asociadas a las mismas. Como se puede deducir, será tarea del Webmaster, hacer las veces de administrador de la herramienta WAG, aunque no tendrá que crear páginas ni formularios individualizados para los distintos usuarios.

1.3 Breve descripción del funcionamiento de WAG

Como ya se ha explicado anteriormente, el WAG residirá en el lado del servidor Web, y trabajará apoyándose en él. Los usuarios, por su parte, proporcionarán al sistema la estructura de los datos que van a ser publicados. Además, serán los propios usuarios los encargados de proporcionar los datos, así como de realizar las pertinentes actualizaciones de los mismo.

El sistema WAG proporciona un conjunto de aplicaciones Web, que permiten que los datos sean accedidos por cualquier internauta que acceda a la misma desde su navegador Web. Se generará una aplicación Web independiente para cada definición de datos[1] realizada por un usuario . Estas aplicaciones Web serán la vía utilizada, por los usuarios propietarios de las aplicaciones, para realizar el mantenimiento de los datos.

Por su parte el 'Administrador del WAG' (que normalmente corresponderá con el Webmaster), será el encargado de instalar el sistema WAG, será el encargado de administrar y mantener las aplicaciones Web y las bases de datos asociadas a las mismas. Además, será el encargado de habilitar un espacio reservado, dentro del sistema de ficheros del servidor, para cada usuario. En este espacio, cada usuario podrá almacenar algunos ficheros de configuración del sistema WAG, que permitirán, aunque de forma opcional, adaptar la apariencia de las aplicaciones a sus necesidades.

[1] Las definiciones de datos se realizarán agrupadas en bases de datos y tablas de datos. Por tanto, definir la estructura de los datos implica definir una base de datos, la cual se subdividirá en tablas de datos.

Capítulo 2

Tecnologías utilizadas

2.1 Bases de datos relacionales.SQL.PostgresSql

2.1.1 Los sistemas de base de datos

Un sistema de base de datos es un sistema computarizado cuya finalidad general es almacenar información y permitir a los usuarios recuperar y actualizar esa información con base en peticiones. La información en cuestión puede ser cualquier cosa que sea de importancia para los usuarios.

Un sistema de base de datos está formado por cuatro componentes principales:

- **Los datos.** Información de interés para los usuarios. En general, los datos de una base de datos serán tanto integrados como compartidos. La integración de datos permita la eliminación, parte de la redundancia de datos. Por su parte, por datos compartidos entenderemos la posibilidad de que más de un usuario pueda acceder a una misma pieza individual de datos, probablemente con distintos fines.
- **El hardware.** Los componentes de hardware que componen el sistemas se pueden dividir en:
    1. Los volúmenes de almacenamiento secundario (principalmente discos magnéticos) que son utilizados para el almacenamiento físico de los datos, junto con los dispositivos asociados de E/S (unidades de disco, etc), los controladores de dispositivos, los canales de E/S, entre otros.
    2. Los procesadores hardware y la memoria principal asociada a estos, que permite la ejecución del software del sistema de base de datos.
- **El Software.** Entre la base de datos física (los datos que se encuentran almacenados físicamente) y los usuarios del sistema, hay una capa de software conocida como: sistema de administración de bases de datos (DBMS) , servidor de base de datos o administrador de base de datos.

El DBMS proporciona al usuario una percepción de la base de datos que está, en cierto modo, por encima del nivel del hardware y que maneja las operaciones del usuario (como las operaciones SQL), expresadas en términos de ese nivel más alto de percepción. Aunque el componente software más importante dentro de un sistema de base de datos es el DBMS, no es el único. También existen otras utilidades, tales como las herramientas de desarrollo de aplicaciones, ayudas de diseño, generadores de informes, y el más importante, el administrador de transacciones o monitor PT.

- Usuarios. Existen tres grandes clases de usuarios, que en cierto modo se solapan entre sí:

  1. Programadores de aplicaciones, que haciendo uso de algún lenguaje de programación (C, C++, etc), construyen programas que acceden a las bases de datos mediante la emisión de solicitudes al DBMS(por regla general , mediante comandos SQL).
  2. Usuarios finales, que son aquellos que interactúan con el sistema desde terminales de trabajo. El usuario final puede acceder a una base de datos mediante aplicaciones en línea, o bien a través de a software proporcionado por el propio sistema de base de datos.
  3. El administrador de la base de datos o DBA. La función del DBA consiste en crearla base de datos real e implementar controles técnicos necesarios para hacer cumplir las diversas decisiones tomadas por el administrador de datos (decide qué datos van a ser almacenados y establece política para el manejo y mantenimiento de dichos datos).

2.1.2 ¿ Qué es una base de datos ?

Una base de datos es un conjunto de datos persistentes que es utilizado por los sistemas de aplicaciones.

Por datos persistentes entenderemos aquellos datos, que una vez aceptados por el DBMS para entrar en la base de datos, en lo sucesivo sólo pueden ser removidos de la base de datos por alguna solicitud explícita al DBMS.

2.1.2.1 Entidades-interrelaciones

El término entidad se refiere a todos aquellos objetos distinguibles que van a ser representados en una base de datos. Además de las entidades básicas, existirán también interrelaciones (o vínculos) que asocian dichas entidades. Las interrelaciones son parte de los datos tanto como lo son las entidades. Como ejemplo de entidades dentro de un universo de discurso (Centro de Educativo) , tenemos: el alumno, el profesor y la asignatura . Una interrelación podría ser el hecho de que un alumno cursa una serie de asignaturas, las cuales son impartidas por un profesor determinado.

2.1.2.2 Propiedades

Una entidad es cualquier objeto acerca del cual queremos registrar información. Por tanto, podemos deducir que tantos las entidades como las interrelaciones, poseen propiedades que se corresponden con la información que deseamos registrar de ellas.

Continuando con el ejemplo anterior, serían propiedades de la entidad (alumno), su nombre, su edad, su dirección. Atributos de la interrelación 'asignaturas impartidas por un profesor' pueden ser el aula dónde es impartida una asignatura por un determinado profesor.

2.1.2.3 Datos y modelos de datos

Los datos pueden entenderse como hecho dados (proposiciones verdaderas), a partir de los cuales se puede inferir hechos adicionales. En las bases de datos relacionales (basadas en tablas), los datos son representados mediante filas, las cuales pueden interpretarse como proposiciones verdaderas. A parte, se proporcionan una serie de operadores que operan sobre las columnas de las tablas, y estos operadores soportan directamente el proceso de inferir proposiciones verdaderas adicionales, a partir de las ya dadas.

Por otro lado, un modelo de datos es una definición lógica, independiente y abstracta de los objetos, operadores y demás elementos que constituyen la máquina abstracta con la que interactúan los usuarios. Los objetos nos permiten modelar las estructuras de los datos. Los operadores nos permiten modelar su comportamiento. En este sentido, podemos entender un modelo de datos como un lenguaje de programación (abstracto) cuyos elementos nos permiten resolver una amplia gama de problemas específicos, los cuales no poseen conexión directa entre ellos.

Un modelo de datos puede tener una segunda interpretación. Se puede entender como un programa especifico escrito con el ese lenguaje (de la primera interpretación). Es decir, instanciar el modelo abstracto y adaptarlo a un problema específico.

2.1.3 Ventajas de las bases de datos

El uso de una base de datos proporciona como principales ventajas las que a continuación vamos a citar:

- **Independencia física de los datos.** La independencia física de los datos consiste en la inmunidad de las aplicaciones a cambios en las representación física y en la técnica de acceso, lo que implica que las aplicaciones involucradas no dependan de ninguna representación física o técnica de acceso en particular. Las aplicaciones implementadas en sistemas más antiguos (los sistemas anteriores a los relacionales, o incluso anteriores a las bases de datos) tienden a ser dependientes de los datos. Esto significa que la forma en que físicamente son representados los datos en el almacenamiento secundario y la técnica empleada para su acceso, son dictadas

por los requerimientos de la aplicación en consideración, y más aún, significa que el conocimiento de esa representación física y esa técnica de acceso están integrados dentro del código de la aplicación.

- **Los datos pueden ser compartidos.** Esto quiere decir que pueden compartir la información distintas aplicaciones, pero que además se pueden crear nuevas aplicaciones que usen dicha información sin tener que agregar información a la base de datos
- **Es posible reducir la redundancia de datos**. A diferencia de aquellos sistemas que no están basados en bases de datos, las aplicaciones deben tener sus propios ficheros de datos. Esto puede provocar una duplicidad indeseable de la información.
- **Es posible reducir la inconsistencia.** Si se reduce la redundancia, se reduce también la posibilidad de que se produzca inconsistencia en los datos, al estar los datos integrados.
- **Proporcionar un manejo de las transacciones.** Una transacción es una unidad lógica de trabajo, que por lo regular comprende varias operaciones de la base de datos (en particular varias operaciones de actualización de datos). Esto permite tener controladas unidades lógicas de trabajo, impidiendo que se realicen sólo a medias (o se llevan a cabo en su totalidad o se invalidad todas las operaciones que componen la transacción).
- **Permite mantener la integridad.** Es posible establecer restricciones de integridad de los datos, para impedir que se introduzcan datos inconsistentes en la base de datos. De este modo se llevarán a cabo comprobaciones de estas reglas de integridad cada vez que se realicen operaciones de actualización de los datos.
- **Es posible hacer cumplir la seguridad.** Es posible imponer a los usuarios que el único medio de acceder a la base de datos sea a través de canales adecuados y por tanto se pueden definir las reglas o restricciones de seguridad, que serán verificadas siempre que se intente acceder a datos sensibles.
- **Ayuda a cumplir los estándares.** Ayuda a la estandarización en la representación de datos, lo que ayuda al intercambio de información entre las aplicaciones.

2.1.4 Los sistemas Relacionales y otros sistemas

Un sistema relacional es un sistema en el cual:

1. Los datos son percibidos por el usuario como tabas (y nada más que tablas); y
2. Los operadores disponibles para el usuario permiten generar nuevas tablas a partir de las anteriores. Como ejemplo, existen operadores de proyección, que permiten obtener un subconjunto del columnas de una tabla,

siendo este subconjunto percibido como una tabla formada por dicho subconjunto de columnas.

La razón por la que a estos sistemas se les denomina relacionales es que el término *relación* significa, desde el punto de vista matemático, una tabla.

Por otro lado, un usuario que utilice un sistema no relacional verá otro tipo de estructuras de datos, es decir, o no existen tablas, o bien además de estas, ve otro tipo de estructuras.

Los sistemas anteriores a los relacionales, se pueden agrupar en: sistemas de listas invertidas (más antiguos), jerárquicos y de red (menos antiguos).

Los primeros productos relacionales comenzaron a aparecer a finales de los años setenta y principio de los ochenta. En la actualidad, la gran mayoría de de los sistemas de base de datos son relacionales y operan prácticamente en todo tipo de plataformas de hardware y software disponible. Como ejemplo de estos producto, cabe citar DB2, Ingres, Informix, SQL Server, Oracle y Sybase.

Recientemente están apareciendo algunos sistemas de objetos y objeto/relacionales. La aplicación WAG utiliza un sistema de este último tipo, en concreto PostgreSQL, que es un producto de libre distribución.

También existen otro tipo de sistemas, basados en nuevos enfoques, como son los sistemas multidimensionales y los basados en lógica (o sistema deductivo o experto).

2.1.5 SQL. Lenguaje de las bases de datos relacionales

SQL (Structured Query Language) es el lenguaje estándar para trabajar con bases de datos relacionales y es soportado prácticamente por todos los productos en el mercado. Originalmente, SQL fue desarrollado en IBM Research a principios de los años setenta, aunque se conocía como SEQUEL (Structured English QUEry Language); fue implementado por primara vez a gran escala en un producto de IBM llamado System R, y posteriormente en numerosos productos comerciales de IBM y de otros fabricantes. En el año 1986, ANSI e ISO dieron lugar a la primera versión de SQL (SQL1). Posteriormente, en el año 1992, se creo una norma muy extendida denominada SQL2. Por ultimo, la especificación SQL, ampliará el lenguaje para dotarlo de algunos conceptos de orientación a objetos y algunos otros conceptos nuevos en las bases de datos actuales.

SQL es un lenguaje que incluye operaciones tanto de definición como de manipulación de datos:

- **Operaciones de definición**: la instrucción CREATE TABLE permite crear una nueva tabla en la base de datos, asociándole un nombre, los nombres y los tipos de las columnas de la tabla y las claves primaria y ajena de dicha tabla.
- **Operaciones de manipulación de datos**: La manipulación de los datos de la base de datos se lleva a cabo mediante las operaciones de SQL siguiente: SELECT, INSERT, UPDATE y DELETE. Por un lado se puede llevar

a cabo las operaciones relacionales de restricción, proyección y join , mediante la instrucción SELECT (se usa por tanto para realizar las consultas) . Por otra parte, INSERT, UPDATE y DELETE permite la actualización de los datos de una tabla de datos (inserción, actualización y eliminación de tuplas, respectivamente).

2.1.5.1 Definición de datos en SQL

SQL emplea los términos tabla (table), fila(row) y columna (column) en vez de relación, tupla o atributo, respectivamente. Las órdenes de SQL para definir los datos son CREATE (crear) , ALTER (alterar o modificar) y DROP (eliminar)

- **Orden CREATE TABLE**

  La orden CREATE TABLE sirve para especificar una nueva relación dándole un nombre y especificando sus atributos y restricciones. Los atributos se especifican primero, y a cada uno se le da un nombre, un tipo de datos para especificar su dominio de valores, y quizá algunas restricciones.

```
CREATE TABLE <nombre tabla>
      (<nombre de columna> <tipo de columna>
                           [<restric-
ción de atributo>]
      {, <nombre de columna> <tipo de colum-
na>
                           [<restric-
ción de atributo>]}
      [<restricción de tabla> {,<restric-
ción de tabla>}])
```

  Entre los tipos de datos disponibles para los atributos están los numéricos, los de cadena de caracteres, los de cadena de bits y los de fecha y hora. Los tipos de datos numéricos incluyen números enteros de diversos tamaños (INTEGER o INT, y SMALLINT) y números reales de diversas precisiones (FLOAT, REAL, DOUBLE PRECISION). Podemos declarar números con formato empleando DECIMAL(i,j) (o DEC(i,j) o NUMERIC(i,j)). Por otro lado, los tipos de datos de caracteres tienen longitud fija (CHAR(n) o CHARACTER(n), donde n es el número de caracteres) o variable (VARCHAR(n) o CHAR VARYING(n) O CHARACTER VARYING(n), donde n es el número máximo de caracteres). En SQL2 existen además varios tipos de datos para fecha y hora. El tipo de datos DATE, tiene 10 posiciones y se compone por año-mes-día. El tipo de datos TIME, tiene al menos ocho posiciones, con las componentes hora-minutos-segundos. Un tipo de datos de marca de tiempo (TIMESTAMP) incluye los campos DATE y TIME, más un mínimo de seis posiciones para fracciones de segundo y un calificador de WITH TIME ZONE.

  SQL permite el valor nulo [1] (NULL) como valor de un atributo. Se puede

[1] El valor nulo hay que interpretarlo como la ausencia de valor, que no es lo mismo que el valor numérico 'cero' o el valor para las cadenas de caracteres 'cadena vacia'.

especificar una restricción NOT NULL si no se permiten valores nulos para un determinado atributo. También es posible definir un valor por defecto para un atributo mediante la cláusula DEFAULT <VALOR> a la definición del atributo. Si esta cláusula no se especifica, el valor por defecto para un atributo se considerará el valor nulo.

Después de las especificaciones de atributos (o columnas), podemos especificar restricciones de tabla adicionales, incluidas la de clave y de integridad referencial. La cláusula PRIMARY KEY especifica uno o más atributos que constituyen la clave primaria de una tabla. La cláusula UNIQUE (único) permite especificar claves alternativas[2]. La integridad referencial se especifica mediante la cláusula FOREING KEY (clave ajena[3]) .Una restricción de integridad referencial puede ser violada cuando se insertan tuplas cuando estas son eliminadas o cuando se modifica el valor de una campo de clave ajena. Por tanto, se puede especificar la acción que se debe realizar cuando ocurra alguna violación de este tipo. Las opciones son SET NULL (establecer el valor nulo), SET DEFAULT <valor> (establecer un valor por defecto) o CASCADE (propagar). Todas estas acciones pueden calificase con la instrucción ON UPDATE (al actualizar) u ON DELETE(al eliminar). A todas las restricciones se le puede asociar un nombre anteponiéndole la palabra CONSTRAINT, seguido del nombre de la restricción. El hecho de darle un nombre a una restricción, permite que esta pueda ser eliminada en un momento dado.

- **Orden DROP TABLE**

  Permite la eliminación de una tabla que ya no se necesita dentro de una base de datos.

  ```
  DROP TABLA <NOMBRE TABLA> [ CASCADE | RE-
  STRICT]
  ```

  Mediante el uso de la opción CASCADE, se propagará la eliminación de todas aquellas tablas cuya clave ajena haga referencia a la tabla. Por otra parte, RESTRICT impide que una tabla sea eliminada siempre que se haga referencia a ella en as definiciones de clave ajena de otra tabla.

- **Orden ALTER TABLE**

  La definición de una tabla de datos puede ser modificada mediante la orden ALTER TABLE. Las posibles acciones de alterar una tabla incluyen la adición o eliminación de un atributo (columna), la modificación de la definición de una columna y la adición o eliminación de restricciones de la tabla.

  Para añadir una columna a una tabla usaremos la siguiente orden:

  ```
  ALTER TABLE <nombre tabla> ADD |<defini-
  ción de columna>
  ```

[2]Una clave alternativa hacer la veces de la clave primari, de hay que se entienda como una alternativa a la clave primaria.

[3]Una clave ajena (FOREING KEY), es un conjunto de uno o más campos, los cuales actúan como clave primaria en otra tabla (tabla ajena).

Si por el contrario lo que queremos es eliminar una columna de la tabla, usaremos la orden siguiente, donde la cláusula CASCADE implica la eliminación de todas las restricciones que hagan referencia dicha columna.

```
ALTER TABLE <nombre tabla> DROP <nom-
bre columna> [CASCADE]
```

Se puede también modificar la definición de columna desechando una cláusula por omisión existente o definiendo una nueva cláusula de este tipo.

```
ALTER TABLE <nombre tabla> ALTER <nom-
bre columna> DROP <clausula>
ALTER TABLA <nombre tabla> ALTER <nom-
bre columna> <clausula>
```

Finalmente, se puede alterar las restricciones de tabla eliminándolas o añadiendo nuevas a la tabla.

```
ALTER TABLE <nombre tabla> DROP CON-
STRAINT <nombre_restricción>
ALTER TABLE <nombre tabla> ADD <nueva re-
stricción>
```

2.1.5.2 Consultas en SQL

SQL tiene una instrucción básica para obtener información de una base de datos: la instrucción SELECT (seleccionar).

- **Consultas SQL básicas**

  La forma básica de la instrucción SELECT consta de tres cláusulas SELECT, FROM y WHERE y se construye como sigue:

  ```
  SELECT <lista de atributos>
  FROM <lista de tablas>
  WHERE <condición>
  ```

  donde

  1. <lista de atributos> es una lista de nombres de los atributos cuyos valores va a obtener la consulta.
  2. <lista de tablas> es una lista de los nombres de las relaciones requeridas para procesar la consulta.
  3. <condición> es una expresión condicional (booleana) de búsqueda para identificar las tuplas que obtendrá la consulta.

- **Clausulas WHERE no especificadas y empleo de '*'**

  La omisión de la cláusula WHERE indica una selección de tuplas incondicional; por tanto, todas las tuplas de la relación especificada en la cláusula FROM son aceptadas y se seleccionan para el resultado de la consulta. Si se especifica más de una relación en la cláusula FROM y no hay cláusula WHERE, se selecciona el PRODUCTO-CRUZADO (todas las posibles combinaciones de tuplas) de esas relaciones.

  Si queremos obtener los valores de todos los atributos de las tuplas seleccionadas, no tenemos que listar los nombres de los atributos explícitamente en SQL; bastaría con especificar un asterisco '*', que significa todos los atributos.

- **Las tablas como conjuntos**

  SQL no considera una tabla como un conjunto[4], sino como multiconjuntos (o bolsas), donde puede haber elementos repetidos. Para obtener como resultado un conjunto de de tuplas sin que se repita ninguna, debemos añadir a la cláusula SELECT la palabra DISTINCT. Por tanto a las tablas se le pueden aplicar operaciones típicas de conjuntos tales como UNION (unión), EXCEPT (diferencia) e INTERSECT (intersección), teniendo en cuenta el matiz comentado anteriormente, que las tablas se consideran multiconjuntos. Los operadores conjuntos sólo se pueden aplicar a relaciones compatibles con la unión, de modo que debemos asegurarnos que las relaciones, a las que se le apliquen dichos operadores, deben tener los mismos atributos y éstos, deben de aparecer en el mismo orden.

- **Consultas anidadas y comparaciones de conjuntos**

  En algunas consultas es preciso obtener valores existentes en la base de datos para usarlos en una condición de comparación. Una forma cómoda de formular tales consultas es mediante consultas anidadas, que son consultas SELECT completas dentro de la cláusula WHERE de otra consulta, la cual se denomina consulta externa.

  El valor de comparación IN compara un valor v con un conjunto (o multiconjunto) de valores V y devuelve el valor TRUE (verdadero) si v es uno de los elementos contenidos en V. El operador IN también puede comparar una tupla de valores entre paréntesis y separados por comas, con un conjunto de tuplas compatibles con la unión.

- **Las funciones EXISTS y UNIQUE en SQL**

  EXISTS o su negado (NOT EXISTS) permite comprobar si el resultado de una consulta anidada correlacionada[5] está vacío (no contiene tuplas). En general, EXISTS(Q) devuelve el valor TRUE si hay por lo menos una tupla en resultado de la consulta anidada Q, y devuelve FALSE(falso) en caso contrario. Por su parte, NOT EXISTS(Q) devuelve TRUE si no

[4] Los conjuntos no pueden tener elementos(en este caso tuplas) repetidos, mientra que las tablas sí

[5] Una consulta se dice que es correlacionada cuando una condición en la cláusula WHERE de una consulta anidada hace referencia a un atributo de una relación declarada en la consulta exterior.

hay tuplas en el resultado de la consulta Q, y devuelve FALSE en caso contrario.

La función UNIQUE (Q) de SQL, devuelve el valor TRUE si no existe ninguna tupla repetida en el resultado de la consulta Q, y devuelve FALSE en caso contrario.

- **Funciones agregadas y agrupación**

  Una función agregada es una operación relacional (aplicada a conjuntos). SQL integra las siguientes funciones agregadas: COUNT (cuenta), SUM (suma), MIN (mínimo) , MAX (máximo) y AVG(promedio). La función COUNT devuelve el número de tuplas o valores especificados en una consulta. Las funciones SUM, MAX, MIN y AVG se aplican a un conjunto (o multiconjunto) de valores numéricos y devuelven, respectivamente, la suma, el valor máximo, el valor mínimo y el promedio (la media aritmética) de estos valores.

  En muchos casos, nos interesa aplicar las funciones agregadas a subgrupos de tuplas de una relación, con base a en los valores de algunos atributos. En estos casos necesitamos agrupar las tuplas que tienen el mismo valor para ciertos atributos, loa cuales denominamos atributos de agrupación, y aplicar la función de manera independiente a cada uno de esos grupos. SQL cuenta con una cláusula GROUP BY (agrupar por) para este fin. La cláusula GROUP BY especifica los atributos de agrupación, que también deberán aparecer en la cláusula SELECT.

  Existe una cláusula más HAVING(que tiene), que nos permite obtener los valores de las funciones sólo para aquellos grupos que cumplen una cierta condición. Esta cláusula suele aparecer junto a la cláusula GROUP BY. De este modo, con HAVING se especifica una condición en términos del grupo de tuplas asociado a cada valor de los atributos de agrupación, obteniendo como resultado en la consulta aquellos grupos que satisfagan la condición. Mientras que con el uso de una clausula WHERE se limitaría las tuplas a las que se aplican las funciones, una cláusula HAVING limita grupos de tuplas enteros.

2.1.5.3 Instrucciones de actualización en SQL

Las órdenes de actualización son las siguientes: INSERT (insertar), DELETE (eliminar) y UPDATE (modificar).

- **La orden INSERT**

  La orden INSERT permite añadir tuplas a una tabla.

  En su forma más simple, que permite la inserción de una tupla, debemos especificar el nombre de la relación y una lista de valores para la tupla. Los valores deberán listarse en el mismo orden en que se especificaron los atributos correspondientes en la instrucción CREATE TABLE.

  ```
  INSERT INTO <TABLA> VALUES (<VALOR>, {<VAL-
  OR})
  ```

Una segunda forma de la instrucción INSERT permite al usuario especificar explícitamente una correspondencia entre los nombres de los atributos y los valores asociados a los mismos. En este caso, los valores con valores NULL o DEFAULT se podrían omitir.

```
INSERT INTO <TABLA> (<ATRIBUTO1>, <ATRIBU-
TO2> ,... , <ATRIBUTOn>)
VALUES (<VALOR1>,<VALOR2>,...,<VALORn>)
```

Una tercera opción en la utilización de INSERT es la inserción de múltiples tuplas, al tiempo que crea la tabla y la carga con el resultado de una consulta.

```
INSERT INTO <TABLA> (<ATRIBU-
TO1>,<ATRIBUTO2>,...,<ATRIBUTOn>)
<CONSULTA SQL>
```

- **La orden DELETE**

  La orden DELETE elimina tuplas de una relación. Cuenta con una cláusula WHERE, similar a la de las consultas SQL, para seleccionar las tuplas que se van a eliminar. Las tuplas se elimina explícitamente de una sola tabla a la vez. No obstante, la eliminación de tuplas puede propagarse a otras tablas, si tal acción se especifica en las restricciones de integridad referencial. Dependiendo del número de tuplas seleccionadas por la cláusula WHERE, una sola orden DELETE puede eliminar cero, una o más tuplas. La eliminación de la cláusula WHERE indica que se deben de eliminar todas las tuplas contenidas en la tabla, aunque la tabla permanecerá en la base de datos con una tabla vacía.

  ```
  DELETE FROM <TABLA> WHERE <CONDICIONES>
  ```

- **La orden UPDATE.**

  La orden UPDATE sirve para modificar los valores de los atributos en una o más tuplas seleccionadas. Al igual que en la instrucción DELETE, una cláusula WHERE selecciona de una sola relación las tuplas que se van a modificar. Sin embargo, la actualización de un valor de clave primaria puede propagarse a los valores de clave externa de tuplas de otras relaciones, si tal acción se especifica en las restricciones de integridad referencial. Una cláusula adicional SET indica los atributos que se modificarán y sus nuevos valores. UPDATE únicamente afectará a una relación. Si queremos modificar varias relaciones, debemos emitir varias órdenes UPDATE.

  ```
  UPDATE <TABLA> SET <ATRIBU-
  TO1>=<VALOR> {, <ATRIBUTO>=<VALOR>}
  [WHERE <CONDICIÓN>]
  ```

2.1.5.4 Especificación de índices

SQL posee instrucciones para crear y eliminar índices sobre atributos de una tabla. El atributo o atributos sobre los que se crea un índice se denominan atributos de indexado. Los índices hacen más eficiente el acceso a tuplas con base en condiciones en las que intervienen sus atributos de indexado. Esto significa que, en general, la ejecución de una consulta tardará menos si algunos del los atributos implicados en las condiciones de la consulta están indexados. Esta mejoría puede ser muy importante si se trata de una consulta sobre relaciones de gran tamaño. En general, si están indexados los atributos que intervienen en las condiciones de selección y de reunión de una consulta, el tiempo de ejecución de dicha consulta se reduce. considerablemente.

En los SGBD relaciones, los índices se pueden crear y eliminar dinámicamente. La orden CREATE INDEX permite crear un índice, la cual recibe un nombre que permitirá su eliminación, cuando el índice ya no se necesite.

SQL permite dos opciones a la hora de crear índices. Por un lado, permite especificar la restricción de clave sobre un atributo o conjunto de atributos de indexado. La palabra reservada UNIQUE es la que permite especificar una clave. Por otro lado, se puede crear un índice de agrupamiento o no. Las condiciones de reunión y de selección serán más eficientes cuanto se especifican en términos de un atributo que tiene un índice agrupamiento. En este caso se usa la palabra reservada CLUSTER, al final de la orden CREATE INDEX.

Sobre una tabla sólo se podrá definir un índice de agrupamiento, pero tantos como se deseen que no sean de agrupamiento.

```
CREATE [UNIQUE] INDEX <NOMBRE_ÍNDICE> ON <NOM-
BRE_TABLA> [CLUSTER]
```

La orden DROP INDEX sirve para desechar un índice. Los índices se desechan porque su mantenimiento resulta costoso cada vez que se actualiza la relación base y requieren almacenamiento adicional.

```
DROP INDEX <NOMBRE_ÍNDICE>
```

2.1.6 PostgreSQL

2.1.6.1 ¿ Qué es PostgreSQL?

PostgreSQL es un sistema de gestión de base de datos de tipo *objeto-relacional*[6] basada en POSTGRES, Versión 4.2, desarrollada e el Departamento de Informática de la Universidad de California (Berkeley) . POSTGRES fue un proyecto dirigido por el Profesor Michael Stonebraker y contó con el apoyo de diversos organismos y empresas de E.E.U.U.

[6]Los sistemas de base de datos objeto-relacionales (o ORDBMS), no deben ser confundidos con los sistemas gestores de base de datos orientados a objetos. Los ORDBMS sólo incluyen algunas características típicas de la orientación a objetos.

PostgreSQL es software libre que proviene del código original de Berkeley. Proporciona soporte para SQL92 y SQL99, además de un conjunto de características actuales.

POSTGRES incorporó gran parte de los conceptos de objetos-relacionales que hoy día incluyen algunos gestores de base de datos comerciales. Los gestores de base de datos relacionales tradicionales (RDBMS) permiten la existencia de una colección de relaciones identificadas por un nombre, las cuales poseen atributos a los que se le asocia un determinado tipo de datos. Los sistemas de base de datos actuales permiten tipos de datos tales como números en coma flotante, enteros, cadenas de caracteres, monedas y fechas; pero como es bien sabido, esto no basta para satisfacer las necesidades de las aplicaciones del futuro. Por esta razón, PostgreSQL proporciona un conjunto de nuevos conceptos que permitirán a los usuarios ampliar de forma cómoda sus aplicaciones. Entre estos nuevos conceptos caben destacar:

- herencia
- nuevos tipos de datos
- funciones

Otras características que proporcionan flexibilidad y solidez al sistema son:

- restricciones
- disparadores
- reglas de integridad
- integridad en las transacciones

2.1.6.2 Características de PostgreSQL

Como se ha citado en el apartado anterior, PostgreSQL incluye nuevas características que proporcionan una mayor facilidad para crear aplicaciones modernas, mayor potencia y una mayor flexibilidad. Vamos, seguidamente a detallas cuáles las características principales con respecto a un sistema tradicional de base de datos relacional. Para ello tomaremos como referencia las características del estándar SQL92.

2.1.6.2.1 Nuevos tipos de datos A parte de los tipos de datos que incluye SQL, PostgreSQL soporta un conjunto extendido de nuevos tipos , entre los que podemos destacar:

1. Tipos geométricos: Estos tipos permiten representar objetos especiales de bidimensionales. El tipo point (punto) es la base del resto. Los tipos soportados son point (), line (), lseg (), box (), path (), polygon (), circle (). Además, se proporciona un conjunto de operadores y funciones que permiten realizar operaciones geométricas (escalado, translaciones, rotaciones, hallar puntos de intersección)

2. Tipos de dirección de red: PostgreSQL aporta tipos para almacenar direcciones IP y direcciones MAC, para evitar el uso del tipo cadena de caracteres para esta tarea, puesto que aporta restricciones y funciones asociadas a estos tipos de datos. Los tipos incluidos son cidr (), inet () y macaddr ().

3. Tipo de datos Boolean: Este tipo de datos está incluido en las especificaciones de SQL99.

4. Potente manejo de los tipos de datos sobre fechas.

5. Incluye dentro del tipo de datos numérico el tipo serial, que permite crear identificadores únicos para entradas en las tablas.

6. Tipo text, dentro del tipo de datos cadena de caracteres. Este tipo no está especificado por el estándar SQL, pero si es soportado por muchos RDBMS. Este tipo de datos admite una cadena de caracteres de longitud variable e 'ilimitada' (limitado por la capa de hardware).

7. Se permite definir columnas como arrays multidimensionales de longitud variable. Los arrays podrán ser de cualquier tipo predefinido en PostgreSQL o de un tipo definido por el usuario. No conviene abusar de es esta característica.

2.1.6.2.2 Funciones y operadores PostgreSQL proporciona un amplio conjunto de funciones y operadores para los tipos de datos soportados. Además, los usuarios pueden definir sus propias funciones y operadores. No obstante, hay que ser consciente que la mayoría de los operadores y funciones ofrecidas, exceptuando los operadores aritméticos básicos y de comparación, así como algunas funciones determinadas, no están incluidas en el estándar de SQL, con el consiguiente problema de portabilidad. De todos modo, la mayoría del los RDBMS incluyen funciones compatibles o muy similares.

2.1.6.2.3 Herencia Para entender este concepto, nos centraremos en un ejemplo. Vamos a crear dos tablas : por una lado, la tabla 'capital' que contiene las capitales de provincias, que son ciudades. Por otro lado, crearemos la tabla 'ciudad' . De forma natural, podemos ver que la tabla 'capital' debe de heredar los atributos de la tabla ciudad (una capital es una ciudad, que ejerce como capital).

```
CREATE TABLE ciudad (
nombre text,
poblacion float,
altitud int , -- metros
);
CREATE TABLE capital (
comunidad char(2)
) INHERITS (ciudad);
```

En este caso, una tupla de la tabla 'capital' hereda los atributos(nombre, población y altitud) de su padre, en este caso la tabla 'ciudad'. La tabla 'capital' posee un atributo extra 'provincia', que indica a que comunidad pertenece. En PostgreSQL, una tabla puede heredar desde ninguna o más de una tablas de datos, y las consultas se pueden hacer referencia a todas las tuplas de una tabla o todas las tuplas de una tabla, más las tuplas de las tablas hijas (descendientes).

Veamos esto con el siguiente ejemplo:

Muestra todas las ciudades con una altitud mayor a 500 metros, ya sean capital de comunidad o no

```
SELECT nombre, altitud
FROM ciudad
WHERE altitud>500;
```

Muestra todas las ciudades con una altitud mayor a 500 metros que no sean capital de comunidad.

```
SELECT nombre, altitud
FROM ONLY ciudad
WHERE altitud>500;
```

2.2 Servidores Web.Aplicaciones basadas en Web

2.2.1 Los servidores Web

Básicamente, los servidores Web proporcionan datos estáticos (páginas HTML e imágenes) a los navegadores Web (clientes Web). Un servidor Web recibe la petición de una página Web, identificada por su URL[7]. Esta URL identifica un fichero local situado en cualquier lugar del sistema de ficheros del servidor. En caso de ser encontrado el fichero, se carga de disco y se envía al navegador Web a través de la red (Internet). Este intercambio de datos entre cliente y servidor, se lleva a cabo gracias al protocolo HTTP (HyperText Transfer Protocol).

Este mecanismo tan sencillo, sólo es la base de los que hoy en día se conoce como el World Wide Web. Uno de lo avances más importantes sobre la tecnología Web es la aparición de páginas Web dinámicas, creadas en función de una petición del usuario, ya sea directa o indirectamente. La forma más utilizada de hacer estos es mediante la utilización de CGI (Common Gateway Interface) , los cuales definen cómo el servidor debe de ejecutar los programas locales y enviar las salidas generadas a través del servidor Web hacia el cliente Web. A todos los efectos, el usuario no tiene por qué enterarse que las páginas eran dinámicas (en el lado del servidor), porque el un CGI consiste básicamente en una extensión del protocolo del servidor Web.

[7]URL: Uniform Resource Locator

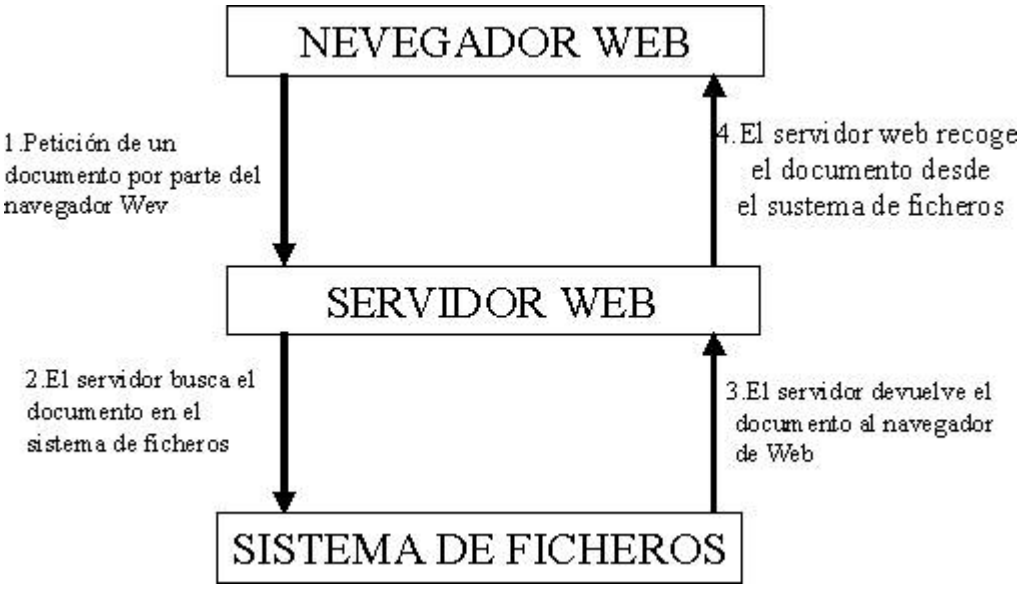

Figura 2.1: Funcionamiento básico de un servidor de Web

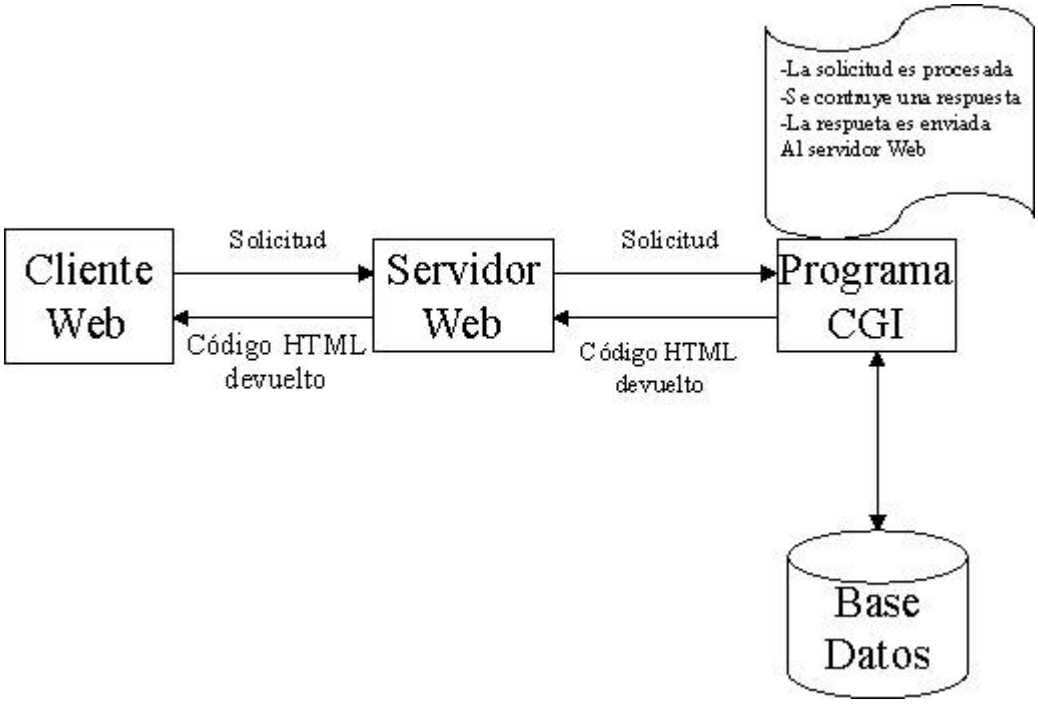

Figura 2.2: Servidor Web haciendo uso de CGI

Un segundo hecho muy importante para el desarrollo del comercio electrónico fue el desarrollo del HTTPS (HyperText Transfer Protocol, Secure), que proporciona una transmisión segura entre el cliente y el servidor Web. Esto implica que la transmisión de ciertos datos (los más confidenciales) entre cliente y servidor se llevará a cabo de forma segura, a través de un medio inseguro, como es en este caso la red.

2.2.2 Aplicaciones basadas en Web

2.2.2.1 Introducción

Paralelamente al desarrollo de Internet, se ha producido una evolución en los servicios que la red ha ido ofreciendo, así como en las herramientas software que los sustentan. El uso más común que se le ha dado a Internet es el de publicación de información, y es precisamente en este aspecto donde mayores facilidades software se están proporcionando. Actualmente FTP y sobre todo WWW son los servicios más utilizados.

Las páginas Web son texto ASCII escritos en el lenguaje HTML, que se transfieren entre los servidores de WWW y los exploradores de los clientes mediante el protocolo HTTP. La investigación y desarrollo en el campo de la creación y mantenimiento de páginas Web es uno de los más dinámicos en el ámbito de Internet, debido a sus consecuencias comerciales de utilización de la red.

Inicialmente, las páginas Web eran estáticas, en el sentido de que, a efectos de usuario, el único proceso realizado era el de visualización de sus contenidos (escritos en lenguaje HTML) por parte del explorador del cliente.

Las páginas estáticas se siguen utilizando ampliamente debido a que forman la base necesaria para la presentación de datos en muchos tipos de situaciones. También influye decisivamente la sencillez con que se pueden crear, instalar y mantener.

En el momento en que se quiere una interacción mayor entre los usuarios y el sistema que soporta las páginas Web, surge la necesidad de reunir y procesar las peticiones del cliente con el fin de ofrecerle informaciones mejor dirigidas, escogidas y laboradas. Por ejemplo, en una universidad que proporcione las notas de los alumnos a través de las páginas Web de cada departamento, un estudiante podría conseguir sus calificaciones navegando por cada departamento y asignatura entre una gran cantidad de páginas Web estáticas relacionadas jerárquicamente. Una alternativa más elegante y sencilla para el alumno sería preguntarle sus datos en un formulario y ejecutar alguna aplicación en el servidor que seleccionara sus calificaciones entre las distintas asignaturas en las que se ha matriculado, y presentarle todas las calificaciones obtenidas a raíz de la petición sencilla y directa realizada por el alumno.

Como podemos ver es necesaria, por tanto ,una fase de procesamiento que permita el intercambio de información entre los usuarios y el servidor de páginas Web, con lo que se introduce el concepto de páginas Web activas o dinámicas.

La capacidad de procesamiento que sustentan las páginas Web dinámicas se puede llevar a cabo siguiendo alguno de los siguientes modelos:

- Páginas activas en el cliente: el procesamiento se lleva a cabo en la máquina del usuario.
- Páginas activas en el servidor: el procesamiento en el equipo donde reside el servidor de Web.
- Mezcla de páginas activas en el servidor y en el cliente: procesamiento mixto.

La ventaja de las páginas activas en el lado del cliente es la descarga de trabajo del equipo servidor, puesto que el procesamiento se lleva a cabo en la máquina del cliente. Además se hace un mejor aprovechamiento del canal de comunicación, puesto que se minimiza el tráfico de datos entre cliente y servidor, al evitar continuos traspasos de información.

No obstante, el uso de páginas activas en el servidor se hace imprescindible en aquellas ocasiones en las que la información debe encontrarse centralizada, es decir, los datos y una buena parte de los cómputos no pueden ser eliminados del lugar 'central'. Será necesario el uso de páginas Web activas en el servidor, cuando las páginas van dirigidas a clientes con limitación hardware (móviles, televisiones, etc).

Una desventaja importante de las páginas activas en el lado del cliente es la dependencia que existente entre la tecnología, el navegador y la plataforma del usuario, problema que no afecta a las páginas Web activas en el servidor, puesto que en este caso las páginas devueltas al cliente estarán formadas por código HTML estándar, con lo que se asegura la compatibilidad entre los distintos navegadores y plataformas.

Lenguajes como PHP[8] y ASP[9] permiten la realización de páginas Web activas en el lado del servidor, a la vez que permiten su coexistencia con páginas activas en el cliente. Además, el acceso a las bases de datos es versátil y sencillo .

2.2.2.2 Funcionamiento de una aplicación Web

Tradicionalmente, en los servidores Web se ha utilizado el mecanismo CGI (Common Gateway Interface) para implementar páginas Web activas en el servidor. Lenguajes como PERL y C han sido muy empleados, aunque se podrían utilizar muchos otros con este propósito.

El funcionamiento básico de un programa CGI se basa en :

1. Lectura de datos proveniente de un formulario situado en una pagina Web, para lo cual se emplea el canal de entrada estándar.

[8]PHP posee un código abierto y es multiplataforma.. Se adapta perfectamente a servidores Web Apache, bajo Linux, aunque, trabaja igualmente bien en otras plataformas como Unix o Windows con servidores Web tales como Netscape o Microsoft.La primera versión fue creada por Rasmus Lerdorf (1994).

[9]ASP es una tecnología diseñada por Microsoft, que va ligada en el servidor al soporte de comunicaciones IIS de Windows NT/2000

2. Procesamiento de la información, lo que puede llevar incluido el acceso a base de datos mediante el lenguaje.
3. Escritura sobre el canal estándar de salida de las páginas HTML de respuesta. Esto conlleva la introducción de sentencias de escritura de códigos HTML.

Aunque un CGI es un método conceptualmente sencillo, tiene como inconvenientes principales:

- Resulta difícil de mantener (esto depende en parte, de la organización que se haga del código)
- La ejecución de CGI es ineficiente, puesto que es necesario la carga del código en memoria cada vez que se realiza una petición por parte de un usuario. Esta

Una alternativa a la utilización de CGI son las aplicaciones ISAPI (Internet Server API) y la versión de Netscape para sus servidores Web NSAPI. Estas aplicaciones, cuyo código es reentrante, pueden soportar las peticiones simultáneas de diversos clientes con una sola imagen en memoria. El gran inconveniente de las aplicaciones ASAPI es que su creación es costosa debido a su complejidad técnica.

Por último, la utilización de lenguajes como PHP o ASP, permite la creación de aplicaciones Web con una sencillez mayor a la empleada en la programación de CGI y con una eficacia igual a la proporcionada por ISAPI (o NSAPI). Haciendo uso de estos lenguajes, las páginas Web pueden ser diseñadas con editores de HTML, puesto que las instrucciones ejecutables y el código HTML están suficientemente delimitados. Ni PHP, ni ASP implican realizar compilaciones, ni los errores de programación provocarán la caída del servidor Web.

2.3 HTML

2.3.1 Introducción

Las páginas Web son los documentos que constituyen la World Wide Web (WWW). Una pagina Web es un documento multimedia, porque está constituida por texto, imágenes, sonido, animaciones, etc. Pero, también es un documento en el que se pueden añadir enlaces a otras páginas y secciones, convirtiéndose así en documentos hiper-media. Las páginas Web están escritas en un lenguaje de programación denominado HTML (HyperText Marckup Language). Para obtener el resultado final, es decir, visualizar la página Web, es necesario la existencia de un programa, denominado "navegador". Este programa es el encargado de recoger el documento en formato HTML e interpretarlo, para ofrecer al usuario el resultado final.

El lenguaje HTML, del inglés *Hypertext Markup Language* (Lenguaje de Marcación General de Hipertexto), fue desarrollado por Tim Barners-Lee junto a

un grupo de científicos del Centro Europeo de Investigación Nuclear (CERN) allá por el año 1992.

Los desarrolladores se apoyaron en el Standard Generalized Markup Language (SGML) que yaestaba en uso en parte del software de edición y publicación del momento. SGML fue la primera tecnología de la información estandarizada y estructurada de cierta importancia, que procedía de IBM y lo utilizaba para mantener y dar formato a sus documentos legales. Posteriormente se expandió y adaptó para ser usado en un amplio abanico de sectores como estándar ISO. Aunque SGML es extremadamente potente, es muy complejo y requiere software de gran complejidad. Debido a esta complejidad y a su elevado coste, SGML no supuso una alternativa clara al hipertexto en los primeros días de Internet.

Usando como plataforma al SGML, el grupo del CERN creó lo que hoy conocemos como HTML, un lenguaje capaz de conectar documentos mediante enlaces (conocidos como *links* en inglés), de muy fácil manejo, publicación y creación de documentos; además de unificar elementos como imágenes, sonido y texto que anteriormente se tenían que distribuir por separado.Las especificaciones del lenguaje HTML original creadas por el grupo del CERN constituyen el núcleo de las especificaciones y extensiones del HTML que han ido apareciendo a lo largo del tiempo.

El lenguaje HTML, es un lenguaje de marca, permitiendo indicar el aspecto que va a presentar un documento, mediante instrucciones incluidas en el propio documento. Un lenguaje de marcas, es por tanto, un conjunto de instrucciones que permiten especificar el formato deseado para el texto marcado. Ejemplo: indica que el texto "hola", debe ir en negrita:

```
<inicio_negrita> hola <fin_negrita>
```

El marcado puede ser físico o semántico. El marcado físico indica exactamente cómo se debe de representar el texto, mientras que el semántico indica el tipo de texto,dejando flexibilidad al visualizador para que aplique el estilo físico que corresponda a ese tipo.

El lenguaje HTML es un lenguaje de marcas orientado a la publicación de documentos en Internet. La mayoría de las marcas son semánticas, debido a la existencia de un gran número de dispositivos distintos que pueden mostrar la información. Los documentos están formados por una serie de bloques con entidad lógica(titulares, párrafos,listas,etc). La interpretación de estas entidades se deja al visualizador (navegador), lo que dan gran flexibilidad al documento, pudiendo, por ejemplo mostrarse en terminales gráficos o de texto.

2.3.2 Características de HTML

2.3.2.1 Definiciones

2.3.2.1.1 Elementos Un elemento de un documento HTML está formado por una marca de comienzo, un bloque de texto y una marca de fin. Todas las marcas en HTML, van encerradas entre los símbolos <>

```
<MARCA> bloque de texto </MARCA>
```

A este tipo de elementos se les denomina contenedores, porque encierran texto entre la marca de comienzo y la de fin.

Por otro lado, existen elementos que no afectan al texto. Se denominan elementos vacíos. Los elementos vacíos no necesitan marca de fin:

```
<MARCA>
```

2.3.2.1.2 Atributos Los atributos están asociados a los elementos, y definen propiedades de estos:

```
<MARCA ATRIBUTO=”VALOR”> bloque de texto </MARCA>
```

2.3.2.1.3 Comentarios Al igual que otros muchos lenguajes de programación, HTML permite el uso de comentarios. Los comentarios son trozos de texto que serán ignorados por el intérprete de HTML; los comentarios se añaden al código con el objeto de hacerlo más claro y comprensible. El comentario comienza con <!– (debe ir todo junto) y termina con –>

```
<!-- aquí va el comentario -- >
```

Como se puede observar, se puede incluir espacios entre – el >, para indicar el final del comentario. Además, cabe significar, que los comentarios se pueden expandir varias líneas (pude haber visualizadores que no sean capaces de interpretarlos de este modo).

2.3.2.1.4 Otras características HTML es un lenguaje estructurado. Existen unas reglas estructurales sobre dónde pueden y no pueden ir los elementos: en primer lugar, existen elementos que no pueden contener otros elementos; en segundo lugar, los elementos no se pueden solapar. Ejemplos:

```
<!-- Esto no es correcto -->
<H2> Titulo <H2> Titulo2 </H2></H2>
<!-- Esto tampoco es correcto. No se pueden sola-
par los elementos-->
<B><P> texto párrafo </B></P>
```

Los caracteres de espacio, tabulaciones y retorno de carro serán ignorados por el intérprete de HTML, siendo entendidos cómo un único espacio en blanco. Esta característica permite introducir, tantos espacios,tabulaciones o retornos de carro como se deseen, para estructurar el código y hacerlo más legible.

También debemos resaltar en HTML no se hace distinción entre minúsculas y mayúsculas, de forma que cuando queremos que se respete esta distinción(al hacer referencia a ficheros, por ejemplo), debemos encerrar el texto entre comillas dobles:

```
<IMG SRC=”Foto.jpg”>
```

2.3.2.2 Estructura de un documento HTML

Un documento HTML comienza y termina con la marca HTML. Esta marca indica que se trata de un documento HTML, para que no se confunda con otro tipo de lenguajes de marcas, que son similares.

Un documento HTML se consta de una cabecera(se usa la marca HEAD) y del cuerpo (se usa la marca BODY). Mientras que la cabecera incluye información asociada documento, el cuerpo incluye la información que constituye al propio documento.

```
<HTML>
<HEAD>
Cabecera del documento (titulo, meta-
información,estilos, etc.)
</HEAD>
<BODY>
Cuerpo del documento (Texto, imágenes, etc.)
</BODY>
</HTML>
```

La cabecera es el primer elemento de un documento. En ella se encuentran otros elementos, como el título del documento :

```
<TITLE> titulo documento </TITLE>
```

Otro elemento que podemos encontrar en la cabecera es el elemento META. Este elemento permite añadir meta-información del documento, como puede ser el autor, una breve descripción del documento, etc.

El elemento STYLE, que trataremos más adelante, si se usa dentro de la cabecera, permite asociar una hoja de estilo[10] al documento.

También, es muy útil la marca BASE, la cual va en la cabecera del documento. Esta marca permite definir una URL[11] de base para toda la página. Esto permite que se puedan usar URL relativos en los enlaces, tomando como base el especificado en el elemento BASE.

```
<BASE HREF="http://www.euler.es/wag">
<!-- Ahora una URL relativa del tipo "pagi-
na.html", es equiva-
lente a "http://www.euler.es/wag/pagina.html" -->
```

El cuerpo del documento, va a continuación de la cabecera, y contiene todo el material del documento que se quiere mostrar.

```
<BODY BGCOLOR=color BACKGROUND=imagen>
Cuerpo del documento ....
</BODY>
```

[10] fichero donde se especifica el aspecto para un documento HTML(style sheet)
[11] Nombre que identifica unívocamete un fichero en la red.

El atributo BGCOLOR permite especificar un color de fondo. El formato de los colores se puede especificar de dos formas distintas:mediante el nombre del color (aqua,black,blue,etc...) o mediante la intensidad de las componentes RGB del color (#RRGGBB: expresadas en hexadecimal, siendo 00 intensidad nula y ff la intensidad máxima).

```
<BODY BGCOLOR=#FF0000 >
<!-- El color de fondo será el rojo -->
</BODY>
```

El atributo BACKGROUND permite especificar una imagen de fondo, la cual se mostrará como fondo de la página Web.

2.3.2.3 Elementos básicos

- **Titulares.** Se pueden utilizar distintos niveles de titulares para las distintas secciones de un documento, mediante el elemento Hx, donde x es el nivel del titular. x puede tomar un valor desde 1 hasta 6, donde 1 es el titular de mayor importancia, y el 6, el de menor. El uso de titulares permite una mejor organización del texto documento.
- **Realzado de caracteres.** Existen dos tipos de realzado de caracteres: el realzado semántico (la mayoría), y el realzado físico. Entre el realzado semántico podemos resaltar los elementos EM (texto enfatizado), STRONG (texto muy enfatizado), PRE (texto pre-formateado). Entre el realzado físico, se pueden destacar los elementos B (texto en negrita), I (texto en itálica), U(texto subrayado) y TT (texto en fuente de máquina de escribir).
- **Párrafos.**Para indicar el comienzo de un párrafo, se usa la marca P.El final del párrafo se indica mediante la marca de fin de párrafo /P (la cual es opcional) o bien mediante comienza un nuevo bloque de texto, bien sea un nuevo párrafo, una lista (UL, OL,DIR,MENU,DL) o una anotación (BLOCKQUOTE).
- **Ruptura de línea**. La ruptura de línea o espacio vertical, se puede inducir mediante la marca BR.
- **Listas**. Existen en HTML varios tipos de listas: la lista de definiciones y las listas regulares. Las listas regulares a su vez, se pueden distinguir en listas numeradas y no numeradas.

    1. La lista de definiciones permite mostrar una lista de términos y su definición asociada. La lista de definiciones, debe ir encerrada entre las marcas DL y /DL. El término se define con la marca DT y la definición, con la marca DD.
    2. La lista no numerada representa una lista de elementos que no tienen que ir precedidos por un número. Se emplea la marca UL para el comienzo y /UL para el final. Los elementos de la lista se definen mediante la marca LI.

3. La lista numerada representa una lista de elementos en un orden determinado. Cada elemento va precedido por un numero secuencial. Se usan las marcas OL y /OL, para delimitar una lista numerada. Cada elemento se define mediante la marca LI.

- **Líneas horizontales.** Una línea horizontal se dibuja mediante el elemento vacío HR.

```
<HR SIZE=altura en pun-
tos WIDTH=anchura en puntos >
```

Las líneas horizontales permiten separar en distintas secciones un documento.

- **Imágenes.** Podemos insertar una imagen en un documento usando el elemento IMG, el cual inserta una imagen y adapta el texto que la rodea, para que no se solape con dicha imagen. La sintaxis de este elemento es la siguiente:

```
<IMG SRC=fichero_de_la_imagen WIDTH=anchura
HEIGHT=altura ALIGN=alineación>
```

El atributo SRC permite especificar el fichero que contiene la imagen, siendo por tanto un atributo de uso obligatorio. Por otra parte, los atributos WIDTH y HEIGHT indica la anchura y la altura de la imagen, respectivamente.Si no se especifica ninguno, se tomarán las dimensiones de la imagen real. Si se especifica sólo alguno de los dos, el otro se calculará de forma automática manteniendo las proporciones de la imagen original. La alineación que tendrá el texto con respecto a la imagen, se especifica mediante el atributo ALIGN. Este atributo puede tomar los valores: TOP (alineación superior), MIDDLE (alineación al medio), BOTTON (alineación inferior), LEFT (alineación a la izquierda) o RIGHT (alineación a la derecha).

- **Enlaces.** Los enlaces de hipertexto nos van a permitir enlazar unos documentos con otros, o bien, con otros objetos. Para ello, se utiliza el elemento ancla A, cuya sintaxis es:

```
<A HREF=url_del_documento>texto del en-
lace</A>
```

El texto que encierra el elemento A, será la parte activa del ancla, mostrándose este de forma especial (azul subrayado, en la mayoría de navegadores). Cuando se activa el ancla, se carga en el navegador el documento o recurso al cual hace referencia la URL del ancla. La URL se especifica mediante el atributo HREF.

No sólo texto se puede encerrar en el ancla, sino que también se puede usar una imagen.

2.3.2.4 Tablas

Las tablas son elementos muy utilizado en la creación de páginas Web, pues son muy versátiles. Una tabla, formada por un conjunto de celdas agrupadas por filas y columnas, se define mediante el elemento TABLE. Dentro de un elemento TABLE se puede encontrar un sólo elemento CAPTION (titilo de la tabla) y uno o más elementos TR(fila de la tabla). Un elemento TR, a su vez, sólo podrá contener en su interior elementos de dos tipos: TH (celda de encabezado) y TD (celda de datos).

Las tablas, por defecto aparecerán alineadas a la izquierda, interrumpiendo el flujo de texto, y sin bordes. Además, todas las filas deben tener el mismo numero de columnas y todas las columnas , a su vez, deben tener el mismo número de filas.

Los atributos de las tablas son:

- **BORDER:** especifica la anchura, en número de píxeles, del borde de la tabla. Su valor por defecto es 0(ausencia de borde). Si se coloca el atributo pero no el valor, tomará el valor 1.
- **CELLSPACING:** anchura de la línea divisoria entre celdas.
- **CELLPADDING:** espacio entre los bordes de celda y el contenido de esta.
- **BGCOLOR:** color de fondo de las celdas de la tabla. Su formato es BGCOLOR=color (expresado en formato #RRGGBB o mediante nombres).
- **ALIGN:** alineación de la tabla. Puede venir expresada de manera absoluta, en número de pixels, o en porcentaje respecta a la anchura total de la ventana del visualizador.
- **BACKGROUND:** imagen de fondo. Se coloca en forma de mosaico dentro de cada celda, como la imagen de fondo de una página.
- **BORDERCOLOR:** color del borde la de tabla.

Las celdas de las tablas se crean mediante los elementos TR,TD y TH (cuya marca final es opcional), y poseen las siguientes propiedades: tamaño (anchura y altura), alineación (horizontal y vertical) y color de fondo.

- **Tamaño de las celdas**

  El tamaño de las celdas es calculado automática. No obstante, es posible indicar el tamaño de las celdas mediante los atributos WIDTH Y HEIGHT, de los elementos TH y TD (y no de TR).

  WIDTH especifica la anchura de la celda (WIDTH=anchura). La anchura se pude expresar en valor absoluto de pixels (WIDTH=número), o bien, porcentaje con respecto al total de la tabla (WIDTH=número%), o bien, el valor relativo a la anchura de otras tablas (WIDTH=número*). Todas las celdas de una misma columna tendrán la misma anchura. En caso de existir celdas de una misma columna con distintas anchuras, se tomará la mayor.

HEIGH especifica la altura de una celda en pixels. Todas las celdas de una misma fila deben tener la misma altura. Si se especifican distintos valores de alturas, se toma el mayor de ellos.

- **Alineación del contenido de las celdas**

  El atributo ALIGN permite alinear horizontalmente el contenido de la celda. Los valores pueden ser left (izquierda),center (centro) y right (derecha). El valor por defecto es left.

  El atributo VALIGN permite alinear verticalmente el contenido de la celda. Los valores pueden ser top (arriba), middle (centro) y bottom (abajo). El valor Por defecto es middle.

  Ambos atributos pueden aparecer en los elementos TR,TH y TD. Si aparecen en TR especifican valores válidos para toda la fila, aunque siempre tendrán preferencia los que aparezcan en los elementos TH Y TD.

- **Color de fondo de las celdas**

  El atributo BGCOLOR de los elementos TR, TH y TD permite especificar un color de fondo para la celda. En el caso de TR, el color se aplicará a toda la fila.

- **Imagen de fondo de las celdas**

  El atributo BACKGROUND permite colocar una imagen de fondo en una fila de la tabla (si se aplica a TR) o en una celda (si se aplica a TH o TD).

Las celdas de una tabla se pueden extender en la fila o en la columna, dando lugar a las denominadas celdas multifila y multicolumna, respectivamente. Para ello, se utilizan los atributos ROWSPAN y COLSPAN de los elementos TH y TD:

- **ROWSPAN:** extiende una celda a varias filas. Formato:

  `ROWSPAN=número_filas`

- **COLSPAN:** extiende una celda a varias columnas.Formato:

  `COLSPAN=número_columnas`

Debemos saber, que dentro de una celda se puede insertar cualquier elemento: párrafos,titulares, e incluso tablas. Por tanto, las tablas son muy utilizadas para dar formato al contenido de una página. Por ejemplo, las tablas se utilizan entre otras muchas cosas para distribuir imágenes y texto, crear texto en varias columnas, colocar notas al margen y realizar menús.

También, debemos resaltar la existencia del modelo de tabla más reciente, que consiste en dividir la tabla en tres bloques: encabezado (elemento THEAD), cuerpo de la tabla (elemento TBODY) y pie de la tabla (elemento TFOOT). Todos los bloques contienen elementos TR, que a su vez contienen elementos TH y TD. El encabezado y el pie están pensados para actuar como partes fijas de la tabla, de forma que si ésta es mayor que la pantalla, el encabezado y el pie se mantienen, y se desplazan el resto de las filas. También se visualizarán líneas más gruesas entre los bloques. No obstante, esto aun no está contemplado por la mayoría de los visualizadores.

2.3.2.5 Frames

Con el uso de frames (también denominados marcos), se puede dividir la ventana del visualizador en viarias subventanas (o frames), pudiéndose, de este modo, mostrar en cada una de ellas un tipo distinto de información. (Un ejemplo típico, es un frame que contiene un menú, que será siempre visible aunque cambie el contenido del resto de la pantalla).

Los frames se definen mediante el elemento FRAMESET. Un elemento FRAMESET sustituye a elemento BODY de un documento HTML, permitiendo dividir la pantalla del visualizador. Dentro de un elemento FRAMESET, podemos incluir nuevos elementos FRAMESET, indicando así, que un frame se subdivide a su vez en otros frames. El elemento FRAMESET puede dividir la pantalla, bien sea horizontalmente, o bien, verticalmente. Los atributos del elemento FRAMESET son:

- **BORDER:** anchura en pixels del borde los frames. Su valor por defecto es 5, y sólo puede ser fijado por el FRAMESET más externo.
- **BORDERCOLOR:** color del borde de los frames. El valor especificado se sustituye por los correspondientes a los atributos BORDERCOLOR de los elementos FRAMESET o FRAME de su interior.
- **COLS:** anchura de los frames verticales en que se divide la ventana.
- **FRAMEBORDER=(yes/no):** especifica cómo se dibuja el borde. Si se especifica yes, se muestra un borde tridimensional, mientras que si se especifica no, se muestra un borde liso,. Si el valor del atributo BORDER es 0, no se dibujará el borde, ignorándose por tanto este atributo.
- **FRAMESPACING:** espacio a dejar entre los frames.
- **ROWS:** altura de los frames horizontales en que se divide la ventana.

Los atributos COLS Y ROWS tienen como valor una lista de números separados por comas. Dichos números indican la anchura y altura, respectivamente, de los frames en que se divide la ventana. Existen tres formas distintas de expresar estos valores:

1. **Número de pixels:** Indica el tamaño absoluto del frame en pixels. El formato es : número
2. **Porcentaje:** Tamaño de frame relativo al total de la ventana. Si la suma es mayor de 100, se reescalan todos los frames. Si es menor de 100 y no existen frames relativos, se reescalan todos los frames. Si la suma es menor de 100 y existen frames relativos, el espacio restante se asocia a estos frames.Formato: número%
3. **Tamaño relativo:** el espacio que queda después de asignar espacio a los frames con tamaño fijo o en porcentaje se divide entre todos los frames de tamaño relativo, de acuerdo con sus pesos correspondientes.

Uno de los dos atributos ROWS o COLS debe aparecer obligatoriamente, pero nunca ambos a la vez. Un FRAMESET divide la ventana en filas o en columnas; si se desea dividir en filas y columnas, hay que utilizar más de un elemento FRAMESET, unos dentro de otros.

Finalmente, cada frame posee unas propiedades posee unas propiedades, las cuales pueden establecerse mediante el elemento FRAME. Los atributos del elemento FRAME son:

- **BORDERCOLOR:** color del borde de los frames. Sustituye al color especificado por el elemento FRAMESET exterior. Si dos frames vecinos especifican colores distintos, el color queda indefinido.
- **FRAMEBORDER=(yes/no)**:especifica cómo se dibuja el borde. Si se especifica yes, se muestra un borde tridimensional, mientras que si se especifica no, se muestra un borde liso. La anchura del borde está especificada en el atributo BORDER del elemento FRAMESET exterior.
- **MARGINHEIGHT:** altura en pixels de los márgenes superior e inferior del frame. El valor por defecto suele ser 10, y no pude ser menor a 1.
- **MARGINWIDTH:** anchura en pixels de los márgenes derecho e izquierdo del frame. El valor por defecto suele ser 10, y no puede ser menor a 1.
- **NAME:** nombre del frame, para usar el atributo TARGET del elemento ancla de hipertexto A[12].
- **NORESIZE:** indica que el frame tiene un tamaño fijo que no puede ser modificado por el usuario. Si no aparece este atributo y el frame posee bordes, el usuario puede modificar su tamaño arrastrando los bordes con el ratón.
- **SCROLLING=(yes/no/auto):** especifica la existencia o no de barras de desplazamiento en el frame. Un valor "yes" indica que siempre las hay, mientras que un valor "no" indica que nunca las hay. El valor "auto" indica que las barras de desplazamiento aparecerán sólo en el caso de que sean necesarias.
- **SRC:** URL del documento a mostrar en el frame. Si no existe este atributo, el frame aparece inicialmente en blanco.

Cuando se activa un enlace, el documento enlazado se carga por defecto en la ventana donde se encuentra el ancla del enlace. Sin embargo, en el caso de los frames interesa con frecuencia que el documento se cargue en otro frame o bien ocupe la ventana completa del visualizador. Para ello existe el atributo TARGET del elemento ancla A, que puede tomar como valor el nombre de un frame, el cual viene dado por su atributo NAME. Si no existe el atributo TARGET, el documento se carga en el propio frame. Si existe, pero no existe el nombre indicado en él, abre una nueva ventana y se carga allí el documento.

[12]El atributo TARGET del elemento A permite que el documento al que hace referencia el enlace, se carge en el frame indicado por éste.

2.3.2.6 Hojas de estilos

Hasta la versión 4.0, los elementos de lo que se llama estilo (los que controlan la presentación de la información por parte del navegador) se encontraban diseminados en la especificación de HTML y cada diseñador de páginas recurría a ciertos "trucos personales"para obtener efectos atractivos. Con esta versión, aparece un lenguaje específico para definir el estilo (llamado CSS, *Cascading Style Sheets*) y elementos, tanto marcas como atributos, para incorporarlo en un documento.

Por un lado, CSS es un lenguaje completo para la definición de la apariencia que deben tener los elementos de un documento HTML. Está basado, al igual que en el caso de las marcas HTML, en un conjunto de atributos y valores que definen las propiedades de presentación. Una definición en CSS consta de una lista de pares atributo-valor separados por punto y coma (el carácter ;). El atributo se separa del valor por medio de dos puntos (el carácter :):

```
atributo1: valor1; atributo2: valor2; ...; atributoN: valorN
```

CSS ofrece atributos para prácticamente cualquier elemento de presentación del documento, como pueden ser:

- Fuentes, tanto en cuanto a su tipo como a su tamaño.
- Colores, de los caracteres, del fondo de la página, de los enlaces, etc.
- Alineación entre los diferentes elementos del documento: texto, imágenes, tablas, etc.
- Bordes y marcos para la representación de objetos.
- Sangrados e interlineados de párrafos.
- Tipos de presentación alternativos, de manera que puedan escogerse diferentes estilos según el equipo usado para acceder la información, e incluso opciones de accesibilidad para personas discapacitadas.

Cualquier marca HTML puede ir acompañada de una especificación de estilo usando CSS por medio del atributo `STYLE`. Por ejemplo, si queremos que el tamaño de la fuente de un párrafo sea de 12 puntos y que el color del texto sea rojo podemos usar la siguiente marca:

```
<P STYLE="font-size: 12pt; color: red">
```

Como vemos `STYLE` admite como parámetro una lista de especificaciones CSS.

La verdadera potencia de las especificaciones de estilo aparece cuando se usa la marca de estilo en la cabecera del documento. Entre <`STYLE`> y </`STYLE`> pueden incluirse:

- Especificaciones CSS que sean aplicables a determinados tipos de marcas. Por ejemplo, para utilizar el color verde en todas las cabeceras de nivel 1 de un documento incluiríamos en la cabecera del mismo las siguientes líneas:

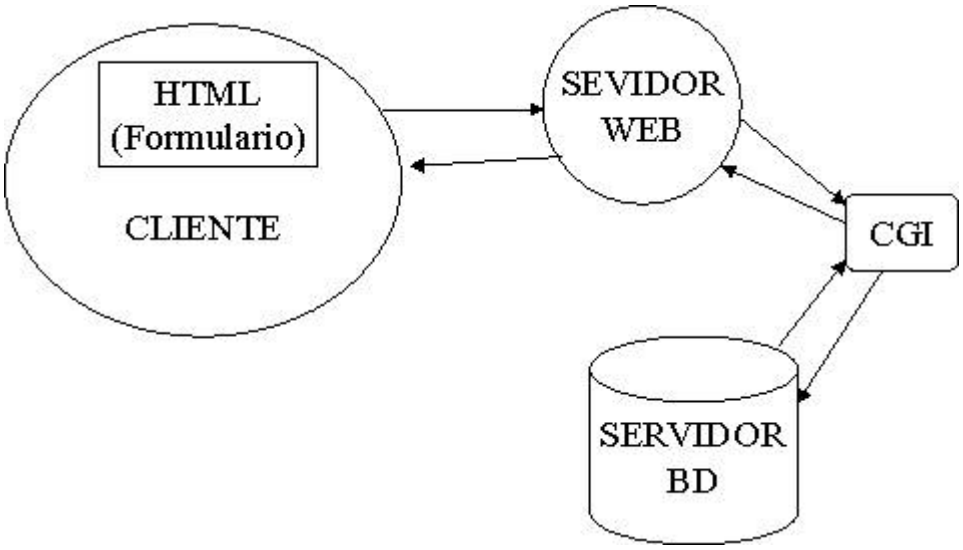

Figura 2.3: Envio y recepción de información a través de formularios

```
<STYLE TYPE="text/css">
H1{ color: green }
</STYLE>
```

- Referencias a especificaciones de estilo que se encuentren disponibles en la red. De esta forma, es posible definir un estilo común para un conjunto de páginas, sin necesidad de modificarlas todas si en cualquier momento es necesario cambiar el estilo.

2.3.2.7 Formularios

Un formulario permite el envío de información desde el cliente hacia el servidor WEB. A través de una serie de controles, el usuario puede introducir información que será enviada hacia el servidor. El servidor pasará la información a un CGI[13], el cual se encarga de procesarla y de emitir una respuesta hacia el servidor. Éste, a su vez, enviará una respuesta al cliente.

Los formularios son creados mediante el elemento FORM. Existe una serie de controles que permiten la recogida de información(cajas de texto,menús desplegables,botones de opciones,etc. La URL del CGI se indica mediante el atributo ACTION del elemento FORM. El CGI se encontrará en un directorio del servidor, de nombre /cgi-bin, y debe estar diseñado conforme al formulario.

[13]Programa que se ejecuta en el servidor Web. Habitualmete toma los datos enviados desde un formulario.

Por tanto, es necesario saber que un formulario está compuesto por dos partes bien diferenciadas: el elemento FORM (recoge los datos) y el programa CGI(procesa los datos).

El elemento FORM permite crear un formulario dentro de un documento HTML. Sus atributos son:

- ACTION: URL del CGI.
- METHOD: método utilizado para enviar los datos. Existen dos métodos: GET y POST.
- ENCTYPE: tipo MIME utilizado para codificar la información cuando se utiliza el método POST. El valor por defecto es "application/x-www-for-urlencoded", y el otro posible es "multipart/form-data".
- TARGET: nombre del frame donde se va a visualizar la información devuelta por el servidor. Su funcionamiento es igual al que tiene el elemento A (ancla de hipertexto).

Dentro de un elemento FORM, podemos insertar distintos elementos de entrada de datos. Cada elemento de entrada tiene un nombre, dado por el atributo NAME, y un valor, dado por el atributo VALUE. Si este último atributo no está presente, se tomará como valor del elemento, el introducido por el usuario. De la misma forma, dentro de un elemento FORM podemos encontrar otros elementos como: listas, encabezados, tablas,etc, para dar el formato deseado al formulario.

Los elementos de entrada datos que nos podemos encontrar dentro de un formulario son:

1. Elemento INPUT

   INPUT especifica varios tipos de controles para la entrada de datos. El tipo se especifica con el atributo TYPE, que puede tomar los siguientes valores:

   (a) BUTTON: botón. Se utiliza para invocar un script[14] contenido en el documento. El valor asociado a él no se enviará al servidor.

   (b) CHECKBOX (caja de verificación): recuadro que puede estar activado (on) o desactivado (off). Por defecto, está desactivado, a no ser que aparezca el atributo CHECKED (que no tiene valor asociado). El atributo VALUE establece el valor asignado a la caja de verificación cuando éste está activado; si no existe, se toma el valor por defecto on. Pueden existir varias cajas de verificación con el mismo nombre, con el cual se asignan distintos valores a una misma variable. Esto es es muy útil cuando se quieren seleccionar varias opciones que no son excluyentes.

[14] Progrmas que se ejecutan al cargarse la página o bien, se pueden asociar a ciertos eventos, de forma que se ejecutarán cuando estos se produzcan.

(c) FILE: lista de selección de ficheros, que permite al usuario seleccionar un fichero para enviar al servidor. Este tipo sólo se permite cuando el atributo ENCTYPE del elemento FORM tiene el valor "multipart/form-data".

(d) HIDDEN: el elemento de entrada no se muestra al usuario, aunque su contenido se envía al servidor. Es muy utilizado cuando se quiere pasar información al servidor desde el cliente. Un uso muy típico es para guardar el estado de la interacción cliente-servidor.

(e) IMAGE: imagen activa, similar al mapa sensible del lado del servidor.

(f) PASSWORD: línea de texto en la que, en lugar del texto tecleado, aparecen asteriscos. Se utiliza para introducir palabras clave (PASSWORD).

(g) RADIO: botón de radio. Los botones de radio tienen sentido cuando hay múltiples botones con el mismo nombre. Todos están enlazados, de manera que al activar uno de ellos los demás se desactivan. Solo un botón puede estar activado a la vez.

(h) RESET: botón de reset, que borra todos los datos introducidos en el formulario y restablece los valores por defecto.

(i) SUBMIT: botón de envío. Al ser pulsado, se envían los datos del formulario a la URL especificada en el atributo ACTION del elemento FORM.

(j) TEXT: línea de texto. El atributo SIZE, que es opcional , permite indicar su tamaño.

Otros atributos del elemento INPUT son los siguientes:

- NAME: nombre asociado al elemento de entrada. El par nombre/valor se enviará al servidor para su procesamiento.
- ALIGN: alineación de la imagen respecto al texto (sólo valido para TYPE="IMAGE"). Puede tomar los valores: bottom, left,middle,right y top.
- CHECKED:sólo válido para TYPE="CHECKBOX" y TYPE="RADIO", indica que una caja de verificación o un botón de radio están seleccionados inicialmente. En el caso de varios botones de radios enlazados, sólo uno de ellos puede tener este atributo.
- MAXLENGTH: sólo válido para TYPE="TEXT" y TYPE="PASSWORD", indica el número máxima de caracteres de la caja de texto. Puede ser mayor que el tamaño de la caja, en cuyo caso el texto se desplazará dentro de ella. Por defecto, este valor es ilimitado
- SIZE: sólo válido para TYPE="TEXT" y TYPE="PASSWORD", indica el tamaño de la caja que contiene al texto.
- SRC: sólo válido para TYPE="IMAGE", contiene el URL de la imagen a incluir.
- VALUE: valor inicial del elemento de entrada. Es obligatorio para TYPE="RADIO".

- HSPACE: válido para TYPE="IMAGE", especifica el espacio horizontal a izquierda y derecha de la imagen.
- VSPACE: válido para TYPE="IMAGE", especifica el espacio vertical encima y debajo de la imagen.

2. Elemento SELECT

   SELECT muestra una lista de valores, especificados mediante el elemento OPTION, y permite seleccionar uno de ellos. Tiene el aspecto de un menú desplegable y su sintaxis es la siguiente:

```
<SELECT NAME=nombre>
     <OPTION SELECTED>Opción 1
     <OPTION>Opción 2
     <OPTION>Opción 3
     <OPTION>Opción 4
</SELECT>
```

   Los atributos de SELECT son los siguientes:

   - MULTIPLE: permite seleccionar más de un elemento de la lista. Si no está presente este atributo, sólo se puede seleccionar un único valor.
   - NAME: nombre de variable asociado al elemento SELECT.
   - SIZE: número de líneas a mostrar en pantalla. Por defecto es 1, por lo que la lista se presenta como un menú desplegable. Si es mayor de 1, aparecerán usualmente barras de desplazamiento. Si está presente el atributo MULTIPLE, el valor por defecto es mayor a 1, para permitir la selección de varios elementos.

   El atributo SELECTED de OPTION indica el valor por defecto. Solo puede haber una, a no ser que el elemento SELECT tenga el atributo MULTIPLE. El usuario puede deshacer la selección por defecto. Otro atributo de OPTION es VALUE, el cual indica el valor asignado a la opción. Si no existe, se toma el propio contenido del elemento OPTION como valor.

```
<SELECT MULTIPLE SIZE=3 NAME=nombre>
    <OPTION VALUE=valor1 SELECTED>Opción 1
    <OPTION VALUE=valor2>Opción 2
    <OPTION VALUE=valor3>Opción 3
    <OPTION VALUE=valor4>Opción 4
</SELECT>
```

3. Elemento TEXTAREA

   Este elemento permite introducir un bloque de texto a través de una ventana de texto con barras de desplazamiento. El tamaño de la ventana se especifica con los atributos ROWS y COLS. El texto puede tener un tamaño ilimitado y no pasa a la siguiente línea al llegar al final de la ventana, siendo necesario para ello introducir un retorno de carro manual.

   Los atributos de este elemento son:

- COLS: número de columnas de texto de la ventana.
- NAME: nombre de variable asociado al elemento TEXTAREA
- ROWS: número de líneas de texto de la ventana.

```
<TEXTAREA COLS=50 ROWS=5 NAME=nombre>
  Este es el valor por defecto del elemen-
to TEXTAREA. Aquí
  pueden aparecer códi-
gos HTML,los cuales no serán interpreta-
dos. </TEXTAREA>
```

2.3.2.8 El programa CGI

Un programa CGI es un programa que se ejecuta en el servidor Web que, por regla general, toma como entrada los valores introducidos a través de un formulario.

El modo en el cual un programa CGI opera es el siguiente:

1. En primer lugar, el CGI recibe la información que le envía el cliente Web. Hay dos modos distintos de pasar esta información.
2. La información es tomada y procesada.
3. El CGI genera como resultado final una pagina HTML, la cual será enviada de vuelta por el servidor al cliente Web.

Cuando el usuario pulsa el botón 'enviar' de un formulario se manda al servidor una cadena de la forma

```
nombre1=valor1&nombre2=valor2&nombre3=valor3&nombre4=valor4
```

donde 'nombreX' es el valor del atributo NAME de cada uno de los elementos de entrada, y 'valorX' es el valor de los correspondientes atributos VALUE, o bien, los introducidos por el usuario. El programa CGI deberá decodificar esta cadena y descomponerla para averiguar el valor asociado a cada una de las variables.

La forma en que el programa CGI accede a los datos enviados desde el formulario depende del método utilizado para pasar los mismos. El método se especifica mediante el atributo METHOD del elemento FORM, pudiendo tomar los siguientes valores:

- Método GET: el CGI recibe los datos en la variable de entorno QUERY_STRING.
- Método POST: el programa CGI recibe los datos a través de la entrada estándar.

La página resultante, tras la ser ejecutado el CGI, será enviada a la salida estándar. En ella suele incluirse una cabecera para indicar el contenido del documento, que puede ser un documento HTML directamente, o bien una referencia a un documento HTML. El contenido de la cabecera es, respectivamente:

1. Documento HTML:
   Primera línea: Content-type: tipo/subtipo
   Segunda línea: En blanco
2. Referencia a un documento HTML:
   Primera línea: Location: URL_doc
   Segunda línea: En blanco

Hay que tener en cuenta que 'tipo/subtipo' es el tipo MIME del documento devuelto (por ejemplo, text/html o text/plain), y que URL_doc es la URL del documento ya existente. En este último caso, el CGI no construye una respuesta propia, sino que recupera un documento almacenado previamente.

2.3.3 Utilización de HTML en el WAG

Como ya sabemos, la finalidad del WAG es la de generar aplicaciones Web. Una aplicación Web, a su vez está formada por un conjunto de paginas Web, las cuales están construidas mediante HTML, de ahí, la importancia que tiene este lenguaje dentro del proyecto WAG.

Desde esta óptica, la función del WAG es la de generar de forma dinámica documentos HTML, de modo que estos serán enviados por el servidor de Web al cliente. Por su parte el cliente también enviará información al WAG, fundamentalmente, mediante los formularios incluidos en los documentos HTML.

Como podemos intuir, gran parte de la implementación del WAG estará basada en un conjunto de programas CGI, que se ejecutan en el servidor Web.

2.4 PHP

2.4.1 Introducción

2.4.1.1 ¿Qué es PHP?

PHP es un lenguaje interpretado, que se introduce dentro de las páginas HTML y que es ejecutado en servidor de Web.

La diferencia más importante entre PHP y un script-CGI escrito en Perl o C para generar como salida un fichero HTML, es que PHP va introducido entre el código HTML, encerrado entre unas marcas especiales, para que se pueda diferenciar. PHP, también se diferencia de Javascript, puesto que este código es ejecutado en el cliente, a diferencia de PHP, que es ejecutado por el servidor. De este modo, las paginas HTML llegarán al cliente, sin saber qué código las ha generado.

2.4.1.2 Breve historia

La primera versión fue creada por Rasmus Lerdorf a finales de 1994. Esta primera versión, denominada " Herramientas para paginas Web personales" , estaba compuesta por un analizador sintáctico y un reducido conjunto de macros, que le permitían a su creador utilizarlas como herramientas comunes en sus páginas Web (contadores de visitas, libro de visitas, etc.).

A mediados de 1995 aparición la versión PHP/FI, que estaba constituida por el conjunto de herramientas para las paginas Web y por un interprete de formularios (FI). Además esta versión daba soporte a MySQL. Esta versión se expandió rápidamente, de modo que a mediados de 1997, ya PHP/FI era utilizado por más de 50.000 servidores.

Fue a mediados de 1997, cuando comenzaron a colaborar con Rasmus, un grupo de usuarios, pasando a ser PHP/FI un proyecto de mayor embergadura. El analizador sintáctico fue reescrito por Zeev Suraski y Andi Gutmans. Fue así como se gestó la versión PHP3, que posee código de la versión PHP/FI y código escrito de nuevo.

En la actualidad, se utiliza la versión PHP4, en la cual se ha revisado y reprogramando el analizador sintáctico. Con ella se ha pretendido conseguir mayores prestaciones y una amplia gama de nuevas características.

2.4.1.3 Para qué se utiliza PHP

Cómo ya hemos comentado, PHP es un lenguaje interpretado, que es ejecutado por el servidor. Es utilizado para la creación de páginas Web dinámicas para comercio electrónico y otro tipo de aplicaciones Web. Las páginas Web dinámicas, son aquellas que interactúan con el usuario, de modo que cada usuario ve las páginas con la información personalizada. Las aplicaciones Web dinámicas son muy comunes en servidores de comercio electrónico,donde gran parte de la información que aparece en las páginas proviene bases de datos o de otras fuente externas, como ficheros, sockets, etc.

Lo que PHP nos va a permitir, una solución universal para programar de forma fácil y cómoda páginas Web dinámicas. Los comandos PHP son incrustados dentro del fichero HTML, encerrados entre unas marcas especiales. La sintaxis de PHP es muy similar a otros lenguajes como C o Perl, de modo que permite que cualquier persona con conocimientos básicos de programación pueda utilizarlo.

PHP, se adapta perfectamente a servidores Web Apache, bajo Linux, aunque, es trabaja igualmente bien en otras plataformas como Unix o Windows con servidores Web tales como Netscape o Microsoft (IIS), por lo que permite la utilización de aplicaciones ASAPI y componentes ActiveX , que son propios de esta plataforma.

PHP permiten al usuario de forma fácil y cómoda conectarse con una amplia gama de servidores de bases de datos comunes, tales como MySQL, PostgreSQL, ORACLE, Sybase, ODBC, etc). Además, PHP, está dotado de un gran

numero de librerías externas, las cuales permiten desde generar un fichero PDF hasta actuar como parser de XML.

Las mayores ventajas que tiene PHP con respecto a otros lenguajes como ASP o ColdFusion, es que posee un código abierto y es multiplataforma. Además, debemos resaltar que son ya más de 6 millones de servidores los que utilizan PHP.

2.4.2 Uso de PHP dentro del WAG

PHP se adapta perfectamente a las necesidades del WAG, puesto que el objetivo del WAG es la generación de aplicaciones Web. Todas las aplicaciones Web necesitan acceder a una base de datos, para obtener los datos que serán mostrados en las paginas Web a los usuarios.

Por tanto, toda la lógica relacionada con la generación de las páginas Web, va a estar escrita en PHP. No obstante, esta lógica será detallada más adelante.

Una razón de peso por la que se ha elegido PHP, es porque se integra perfectamente con el servidores Web Apache, que es de libre distribución , al igual que PHP, así como por su capacidad para poder adaptarse a cualquier plataforma con otro tipo de servidores Web.

www.php.net

www.zend.co

2.5 XML.DOM

2.5.1 XML

Como hemos dicho en el apartado anterior, una de las tecnologías de cierta importancia para la estandarización y normalización de la información estructurada fue SGML, que procedía de IBM. A partir de él se creó el primer formato estándar en la web, HTML. HTML tuvo mucho éxito pero pasado algún tiempo se hizo patente la necesidad de algo más que imágenes estáticas y texto. Lo que siguió fue una infinita sucesión de tecnologías tendentes a añadir interactividad a la Web. Algunas de estas tecnologías, como Java, han permanecido y florecido, mientras que otras se han quedado en la estacada. Aparte de las tecnologías interactivas como Java, el propio HTML atravesó una rápida serie de mejoras para ensancharse y adaptarse a hacer cosas para las que en un principio no estaba destinado.

Agregar interactividad a HTML, esperando que representara información dinámica, supuso un cambio en su propia naturaleza. Aún más importante que el cambio de información estática a dinámica fue el hecho de que se esperaba que HTML sirviera como forma de almacenar tipos específicos de datos, como transacciones financieras, catálogos de productos y resúmenes. Los desarrolladores web pronto se dieron cuenta de las ventajas de conectar páginas a bases de datos de servidores, en cuyo caso se utilizaba HTML como interface a bases

de datos. HTML estaba cambiando de lenguaje de información basado en presentaciones a una tecnología de software. Ahora hay aplicaciones enteras que se implementan como HTML, junto con una serie de scripts, applets de Java, plug-ins y otras tecnologías relacionadas.

A pesar de todo esto los miembros del Consorcio de la World Wide Web (W3C) se dieron cuenta de que sería necesaria una alternativa mucho más amplia a HTML para satisfacer las necesidades futuras de la Web. Quizás la limitación más importante de HTML sea su conjunto finito de etiquetas. Es imposible que los desarrolladores Web añadan etiquetas personalizadas con HTML, lo cual sería infinitamente más útil al tratar con datos pertenecientes a una aplicación o sector específicos. Por ejemplo, un editor de libros utilizaría mucho más etiquetas como <título> y <autor> que una etiqueta de párrafo genérica <p>.

En 1996, el W3C, se propuso introducir el poder y la flexibilidad de SGML en el dominio de la Web. SGML ofrecía tres ventajas importantes que faltaban en HTML: extensibilidad, estructura y validación. Técnicamente, hay estructura en HTML, pero la mayoría de los navegadores obvian si se adhiere a ella o no. SGML está mucho más estructurado que HTML, mientras que el software SGML es mucho menos compasivo que los navegadores web a la hora de aventurarse fuera de la estructura permitida. Así, por razones puramente prácticas, HTML es un lenguaje poco estructurado.

El W3C creó un equipo de expertos en SGML cuyo objetivo era el de crear una nueva tecnología de marcado con las ventajas nucleares de SGML y con la relativa simplicidad de HTML. Por lo que en febrero de 1998 se lanzó la especificación XML 1.0, que significa *"eXtensible Markup Language"* (Lenguaje de Marcado eXtensible)[2].

XML es un subconjunto simplificado de SGML que incorpora muchas de sus características, entre las que se incluyen las tres más importantes: extensibilidad, estructura y validación. En cierta forma, XML representa una nueva época para la Web en la medida que establece una forma de transmitir datos estructurados. XML, en realidad, no es más que un formato de texto estandarizado que sirve para representar información estructurada en la Web. XML está intencionalmente diseñado para que parezca algo sencillo, ya que algo tan aparentemente sencillo e insignificante como un formato de texto puede tener repercusiones vitales en el mundo del software.

XML no sólo es una versión extensible de HTML. De hecho XML no es en absoluto una versión de HTML. Mientras que HTML es un lenguaje de marcado específico que se utiliza para codificar información para presentarla en navegadores web, XML es un **metalenguaje**. Esto implica que HTML, que fue escrito como aplicación específica de SGML, podría rediseñarse como aplicación de XML. De hecho ya está siendo diseñando como aplicación XML, llamado XHTML[1].

Para los diseñadores web, que ven la web como un escaparate para el diseño gráfico, inicialmente XML podría no resultar muy útil. Aunque la presentación juega un papel muy importante en cualquier sitio web, las aplicaciones web modernas son conducidas por datos de tipos muy específicos. Como es fácil deducir, HTML es un lenguaje de marcado muy pobre para representar tales

datos. Con su soporte para vocabularios de marcado personalizados, XML abre la puerta a la descripción cuidadosa de los datos y las relaciones que hay entre las distintas piezas de datos. De hecho, uno de los principales objetivos de XML consiste en separar el contenido (los datos) de la presentación (cómo se ven los datos) en los documentos web.

2.5.1.1 Los metadatos

Al centrarse en el contenido, XML proporciona una forma de incluir metadatos en documentos web. A continuación vamos a intentar esclarecer este concepto y descubrir su utilidad en Internet.

Teniendo en cuenta tanto su crecimiento (que no ha alcanzado ni remotamente su potencial) como su modelo completamente descentralizado e informal de desarrollo, uno de los problemas fundamentales de la información ofrecida por Internet es su accesibilidad. Es evidente que no nos referimos aquí a una cuestión de disponibilidad física ni de anchos de banda, sino a la posibilidad de que la información en cuestión sea encontrada por aquellas personas que, en un momento determinado, pudieran necesitarla.

Una de las respuestas más evidentes a este problema es la creación de índices que, de un modo u otro, permitan dirigir a un usuario a las piezas de información que necesita. Sin embargo, la enorme cantidad de datos disponible hace que estos índices no puedan ser construidos a partir de la información en crudo, tal como está disponible en Internet, so pena de ofrecer respuestas incoherentes o, en el mejor de los casos, irrelevantes a las consultas de los usuarios. Conseguir una lista de cientos, incluso miles, de referencias sin interés como respuesta a una consulta es una experiencia, no menos frustrante por lo común, que han sufrido y sufren muchos usuarios de Internet. Obviar este problema y ser capaces de ofrecer a los usuarios respuestas con sentido a sus búsquedas requiere el empleo de índices basados en información sobre la información, o metainformación. Esta metainformación contiene datos como la fecha en que la información fue producida, su autor, o un conjunto de palabras clave que describen su contenido.

Ahora bien, conviene tener en cuenta que la mayor parte de la información contenida en los servidores Internet no ofrece esta metainformación junto con los datos y que, aún en el caso de que así fuera, su adaptación a medida que la información cambia supone una carga adicional que ni los operadores de los servidores ni los autores de la información suelen asumir. Es más, es importante también considerar que la creación y gestión de la metainformación es un trabajo delicado (por ejemplo, la metainformación suele ser empleada para clasificar el recurso al que hace referencia) y en el que deben emplearse conocimientos específicos.

A la hora de extraer la metainformación acerca de un recurso disponible a través de Internet, hay que tener en cuenta que la información se encuentra fuertemente estructurada dentro de cada uno de los recursos concretos. Se emplean lenguajes con más o menos formalismo, pero con una estructura determinada que facilita la tarea de detectar los datos que resultan relevantes, usando

las regularidades y frecuencias detectadas dentro de esa estructura. Por otro lado, es deseable que el sistema sea lo suficientemente flexible como para poder ser configurado de manera simple e intuitiva, sin necesidad de conocimientos específicos.

Como ya se ha comentado anteriormente, un fichero XML describe la estructura y el significado que posee pero no el formato que va tener, con lo que la extracción o asignación de metadatos a su contenido se simplifica enormemente. Las etiquetas que contienen el documento dicen qué información hay en éste, pero no como debe ser presentado. Mientras en HTML se mezcla en el mismo documento el formato, la estructura y la semántica de la página. Por ejemplo, la etiqueta <B> simboliza formato (pone en negrita parte del texto), la etiqueta <STRONG> simboliza semántica porque está enfatizando la parte de texto que contiene y la etiqueta <TD>, que simboliza la celda de una tabla, es un elemento claramente estructural. Veamos un ejemplo para ver las diferencias entre una página HTML y otra XML, en ambos casos describiremos una canción. En HTML tendría el siguiente aspecto:

```
<dt> Hot Cop
<dd> Jacques Morali, Henri Belolo, and Victor Willis
<ul>
<li> Productor: Jacques Morali
<li> Discográfica: PolyGram Records
<li> Duración: 6:20
<li> Escrita en: 1978
<li> Artista: Village People
</ul>
```

Y en XML tendríamos:

```
<CANCION>
<TITULO> Hot Cop </TITULO>
<COMPOSITOR> Jacques Morali </COMPOSITOR>
<COMPOSITOR> Henri Belolo </COMPOSITOR>
<COMPOSITOR> Victor Willis </COMPOSITOR>
<PRODUCTOR> Jacques Morali </PRODUCTOR>
<DISCOGRAFICA> PolyGram Records </DISCOGRAFICA>
<DURACION> 6:20 </DURACION>
<AÑO> 1978 </AÑO>
<ARTISTA> Village People </ARTISTA>
</CANCION>
```

Como se puede ver las etiquetas utilizadas con XML pueden hacerse autodescriptivas, por lo que la lectura de dicho fichero resulta más fácil. O mejor aún, si en un fichero existiesen multitud de canciones siguiendo este formato, tanto para una máquina como para un humano detectar las canciones en un fichero escrito en XML será mucho menos tedioso que en uno escrito en HTML. Además cualquier persona que no conozca XML, si lee el trozo de código anterior sabría distinguir perfectamente que se trata de una canción y extraería

sin problemas la información almacenada sobre ella como el título, la fecha, los compositores, able.

XML es ideal para grandes y complejos documentos porque los datos se encuentran muy estructurados. Y no sólo nos permite especificar un vocabulario que define los elementos del documento, sino que además especifica las relaciones entre elementos.

2.5.1.2 Comentarios

Los comentarios de XML, como los comentarios de la mayoría de los lenguages de programación, están pensados para anotaciones que no se quiere que tengan influencia sobre los subsiguientes procesamientos. Muchos procesadores descartan los comentarios al analizar los documentos, en cambio DOM, como veremos más adelante, lo considera como un nodo más del árbol que representa al fichero XML. Un comentario es de la siguiente forma:

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

Los comentarios en XML no se pueden anidar.

2.5.1.3 Estructura de un documento XML

La mayoría de los documentos XML comienzan con un prólogo. A veces el prólogo puede ser bastante extenso, aunque en general suele componerse de sólo dos cosas: la Declaración XML y la Declaración de Tipo de Documento (DTD).

La Declaración XML, que debe ser lo primero que aparezca en el archivo XML tiene tres propósitos:

- Identifica la versión de XML a la que se ajusta el documento. De momento, la versión debe ser la 1.0. El número de versión es obligatorio.
- Puede identificar la página de códigos utilizada en el documento. Aunque XML utiliza Unicode, hay muchas maneras distintas de representar los caracteres de un documento (ISO Latin 1, KOI8-R, Big5, etc..). Si no hay declaración, los sistemas XML intentan averiguar el tipo de codificación (a partir de las cabeceras MIME o el sistema de ficheros). Los procesadores de XML sólo están obligados a soportar las codificaciones UTF-8 y UTF-16, pero en la práctica, la mayoría soportan muchas más.
- Puede identificar al documento como independiente (*standalone*). Los documentos independientes informan de que no incluyen declaraciones externas que afecten a la información que pasan a la aplicación de proceso.

El formato de una Declaración XML es:

```
<?xml version='1.0' encoding='US-ASCII' standalone='no'?>
```

El otro elemento común en el prólogo es la de Declaración de Tipo de Documento:

```
<!DOCTYPE nombre-elemento
  PUBLIC "-//Ejemplo//Ejemplo de identificador público de DTD //EN//XML"
         "http://www.ejemplo.com/dtds/ejemplo.dtd">
```

Esta declaración tiene dos propósitos: identifica el elemento raíz del documento (<nombre-elemento>) y asocia al documento un conjunto de declaraciones externas, la DTD (Declaración de Tipo de Documento). La DTD se utiliza para ayudar al analizador (parser) a validar el documento XML.

Una de las funciones más importantes que incorporan los analizadores de XML es la capacidad de validar un documento. Una DTD contiene un conjunto de declaraciones que identifican las reglas adicionales y las restricciones que debe satisfacer un documento para que sea considerado una instancia válida de un tipo de documentos asociados a una DTD.

Por ejemplo, una DTD podría afirmar que un elemento <dirección> debe tener un elemento <nombre>, otro <calle> y otro <telefono>. Puede llegar incluso a especificar los tipos de los datos que pueden contener esos elementos. Un documento es válido si y sólo si satisface todas las reglas de la DTD a la que está asociado.

2.5.1.4 Elementos

Los elementos están delimitados por los signos mayor y menor. Los elementos con contenido aparecen con el formato <nombre-elemento>, seguido del contenido opcional, seguido de </nombre-elemento>. La primera aparición es la etiqueta de inicio, la segunda es la etiqueta de final. En XML, todo elemento no vacío debe tener explícitamente etiquetas de inicio y de final, y deben estar anidadas apropiadamente. A diferencia de HTML, no se puede omitir etiquetas finales nunca.

Los elementos vacíos tienen el formato <nombre-elemento/>. La barra final antes del símbolo mayor indica que el elemento no tiene contenido y en consecuencia no se permite una etiqueta de final.

Los nombres de los elementos deben comenzar con un carácter alfabético y pueden contener cualquier número de letras, dígitos, guiones, subrayados y puntos.

2.5.1.5 Atributos

Las etiquetas de inicio de los elementos pueden tener además atributos (las del final no). En el siguiente ejemplo, el tipo de teléfono de Juan está identificado con un atributo:

```
<direccion>
<nombre> Juan </nombre>
<ciudad> Córdoba </ciudad>
<telefono tipo="trabajo"> 957-334411 </telefono>
</direccion>
```

Los atributos son parejas de nombre-valor. Como los nombres de los elementos, los nombres de los atributos deben ajustarse a la notación de nombres de XML. Todos los valores de los atributos deben ir entrecomillados, con comillas dobles o simples.

2.5.1.6 Entidades de Carácter

XML se reserva una pequeña cantidad de caracteres, como <, para identificar el etiquetado del documento. Para poder utilizar estos caracteres literalmente en un documento se utilizan varios mecanismos. Uno de los más generales es la referencia numérica del carácter. Cualquier carácter Unicode[12] puede ser referenciado en el documento utilizando una referencia numérica al carácter, independientemente de la página de códigos del propio documento. Dicha referencia tiene el formato `&#nnnn;`, donde nnnn es el número del carácter Unicode en decimal. Si se quiere utilizar hexadecimal, el formato a seguir es `&#xhhhh;`.

Los caracteres de etiquetado comunes (<, >, &, ', y ") son tan utilizados literalmente en los documentos XML que se han definido referencias mnemotécnicas para ellos, llamadas entidades y que se resumen a continuación:

- `<` para <.
- `>` para >.
- `&` para &.
- `&apos;` para '.
- `&quot;` para .

2.5.1.7 Instrucciones de Procesamiento

Algunas veces se le quiere dar información adicional a una aplicación de procesamiento específica. Por ejemplo, se le puede quere decir a la aplicación qué nombre de fichero utilizar para enviar de información de salida.

Las instrucciones de procesamiento son la forma de pasar esta información a la aplicación de procesamiento y tienen el siguiente formato:

```
<?destino la-información-que-se-quiere-suministrar?>
```

Todas las instrucciones de procesamiento deben comenzar con un destino. El destino es simplemente un nombre. Todo lo que vaya después del nombre, hasta el ?> de cierre, es parte de la instrucción de procesamiento. Aunque no es obligatorio hacerlo, se ha convertido en tradición utilizar una sintaxis similar a la de los atributos en las instrucciones de procesamiento, como la siguiente:

```
<?destino nombre-dato="valor"?>
```

Los nombres de destinos que empiecen con `xml` (en cualquiera de las posibles combinaciones de mayúsculas y minúsculas) están reservados.

2.5.1.8 Secciones CDATA

En las secciones `CDATA` todo el texto es considerado como caracteres de información. Lo que en él pueda parecer una etiqueta o una instrucción de procesamiento, sólo es el texto de la etiqueta o de la instrucción de procesamiento, y el procesador de XML no intentará interpretarlo de ninuna forma.

Las secciones `CDATA` son utilizadas cuando se quiere que el texto sea considerado como puros caracteres de información y no como etiquetas. Estas son muy útiles cuando tenemos un gran bloque de texto que contiene un montón de caracteres `<`, `>`, `&` o `"`, pero no etiquetas. También resultan muy útiles cuando escribimos algo sobre XML en XML.

Por ejemplo, los dos siguientes bloques de código XML serían considerados como información por el procesador de XML, y no serían ni interpretados ni procesados por éste.

```
&lt;?xml version="1.0" standalone="yes"?&gt;

&lt;SALUDO&gt;

Hola XML!

&lt;/SALUDO&gt;
```

Ahora utilizando las secciones CDATA:

```
<![CDATA[
<?xml version="1.0" standalone="yes"?>
<SALUDO>
Hola XML!
</SALUDO>
]]>
```

El único texto que no está permitido dentro de una sección CDATA es el delimitador de cierre `]]>`.

2.5.1.9 Documentos bien formados

Un documento XML bien formado es aquel que cumple los siguientes requerimientos:

- Las etiquetas de inicio y fin de todos los elementos están anidadas adecuadamente.
- Los valores de los atributos no contienen caracteres `<` (la entidad `<` sí se puede utilizar).
- Todas las referencias numéricas a caracteres se refieren a caracteres Unicode que están permitidos en documentos XML.
- Las entidades con nombre se utilizan de forma correcta (están declaradas, están estructuradas y no son recursivas).

2.5.2 DOM

Los programadores y diseñadores de sitios web persiguen acceder y manipular los contenidos de los documentos web de diferentes formas. Muchas veces se utilizan scripts de una forma u otra para interaccionar con el usuario; o bien para validar formularios, o bien para crear efectos dinámicos visuales. Pero en definitiva, el script accede y manipula las partes relevantes del documento web y el usuario ve los resultados. La gran barrera con la que se encontraban los desarrolladores de estos scripts era que no había un interface estándar para manipular el documento. Esto significa, por ejemplo, que los diseñadores del sitio web deben aprender las diferentes formas en la que los navegadores manipulan la parte relevante del documento para conseguir que el efecto deseado se aprecie para cualquier usuario.

Este interface para el documento es denominado *Document Object Model* (DOM), y se basan en una aproximación en la que el documento web está modelado por medio de objetos. Las partes del documento (elementos, atributos, comentarios ..) son tratados como objetos, los cuales pueden ser accedidos y manipulados. El documento puede ser procesado al completo o por partes.

El W3C ha desarrollado un interface para documentos HTML y XML. El grupo de trabajo del DOM de W3C ha completado la primera especificación. El DOM Level 1 define un API neutral para acceder, manipular y navegar documentos HTML y XML. Un API se compone de una serie de funciones de las cuales sólo conocemos su interface, la cual utilizamos para realizar las llamadas, pero no conocemos cual es su implementación. Define funciones primitivas, y otras funciones que pueden ser combinación de las primitivas y que se utilizan para tareas muy comunes pero que generalmente no forman parte del interface del nivel 1 de DOM. Existen implementaciones de DOM que ofrecen, además de las fuciones básicas del DOM, funciones aún más complejas para otras tareas no directamente especificadas en el DOM.

Uno de los objetivos del DOM es evitar implementaciones forzadas, por lo que sólo se especifica el **interface** que debe tener el modelo. Los interfaces son un concepto a menudo utilizado en Java[8] y otros lenguajes de programación orientados a objetos. El que una clase implemente ciertos interfaces, significa que la clase contendrá las funciones, métodos y propiedades especificadas en el interface. La clase puede implementar los interfaces como el implementador lo desee. Los interfaces pueden ser heredadas, y la herencia múltiple es posible. Incluso con la herencia no hay intención de compartir código u otros detalles de implementación. La clase puede además implementar otras funciones, métodos o propiedades que no estén especificadas en el interface.

El DOM Level 1 define cómo manipular objetos encontrados en documentos HTML o XML. Por lo tanto, esto es importante para saber cuales son las interfaces de esos objetos. La figura siguiente muestra el árbol de herencia del núcleo de las interfaces del DOM Level1, que son derivadas desde la clase principal:

Todas las partes de un documento heredan de `Node`. Los objetos más comúnmente utilizados en el DOM son `Node, Element, Attr, Document y Text`.

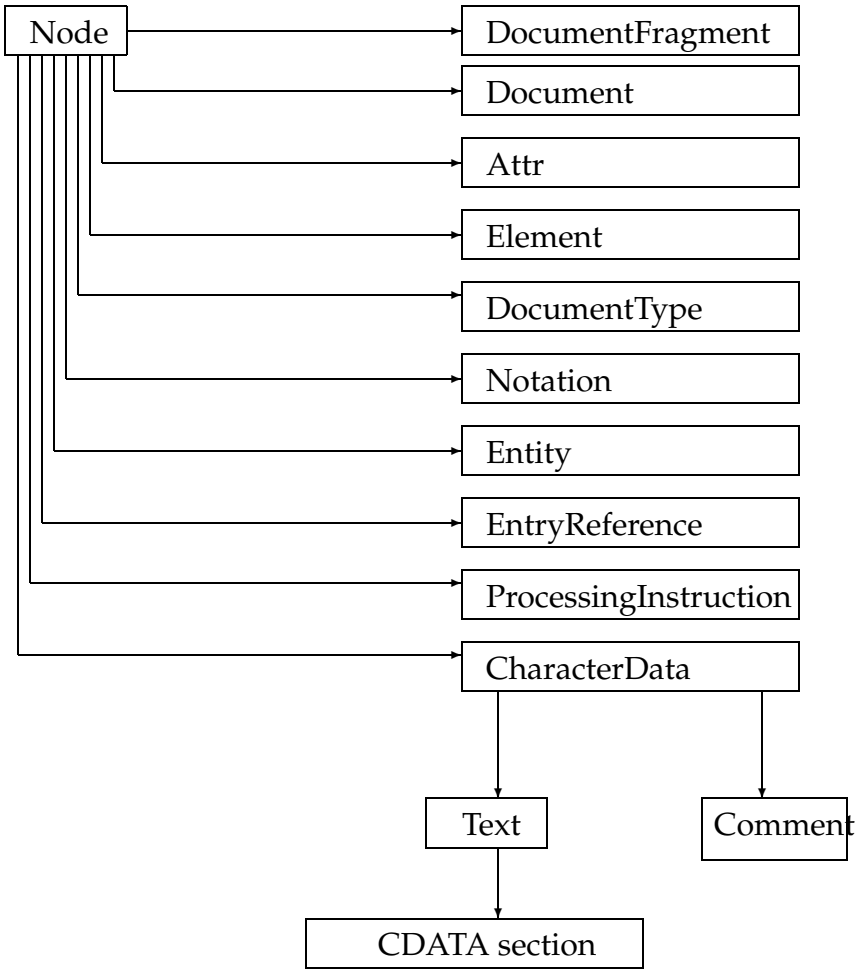

Figura 2.4: Árbol de herencia de DOM Level 1

El interface básico es `Node`. Tiene métodos para borrar, sustituir e insertar, los cuales están disponibles para todas las interfaces que lo heredan. Los métodos básicos para navegar están definidos en Node, tales como las funciones básicas para recorrer un árbol: `ParentNode` (obtiene el padre de un nodo), `firstChild` (mediante este método obtenemos el primer hijo de un nodo padre), `nextSibling` (con este método podemos recorrer los nodos hijos de un nodo, ya que con él se obtiene el siguiente nodo hijo con respecto en el que estamos procesando actualmente) y `previousSibling` (este otro método también nos permite recorrer los nodos hijos pero en sentido contrario a la anterior, ya que nos devuelve el nodo hermano anterior al que estamos procesando actualmente). Algunos usuarios del DOM prefieren una especie de matriz para navegar por el documento, por este motivo, la interface `Node` contiene una forma de obtener una lista de nodos de todos los hijos de una determinada instancia de `Node`. Este nodo puede ser el documento completo o un elemento en particular.

`Element` representa los elementos del documento. Es derivado de `Node` por lo que tendrá todos los métodos y propiedades de él. La mayoría de los documentos tiene más `Element` y `Text` que de cualquier otro tipo de objeto.

`Attr` representa un atributo asociado con un elemento. Los nodos `Attr` no son nodos hijo de `Element`, por lo que los atributos de `Node`, `ParentNode`, `PreviousSibling` o `NextSibling` tienen un valor nulo para los objetos atributo.

`Text` representa parte del contenido textual en un documento. Cuando un documento es parseado al principio dentro del modelo de la estructura DOM, hay un solo nodo de texto por cada bloque de texto existente en el documento. Como el usuario de DOM manipula el documento se pueden ir creando múltiples nodos de texto adyacentes. Puesto que no hay manera de especificar los límites de estos nodos de texto en HML o XML, el método `normalize()` sobre un `Element` nos va permitir unir todos los nodos de texto adyacentes en uno único; cosa que es aconsejable realizar antes de aplicar algunas operaciones que dependan de una estructura en particular del documento.

`Document` representa al documento completo. Es conceptualmente la raíz del árbol del documento, y suministra acceso a todos los datos que hay dentro del documento. Todos los nodos de un documento en particular poseen exactamente el mismo valor para la propiedad `ownerDocument`, la cual apunta al documento con el que está asociado.

El programa que incorpora la implementación DOM, por ejemplo un navegador o un editor, tiene que leer el documento y cambiarlo a una representación interna que permita acceder por medio de las interfaces DOM. Este proceso de lectura depende de la implementación y se muestra en la figura.

El procesador HTML/XML lee el documento o cadena de datos en una representación interna. En la especificación de DOM, a esto se le conoce como representación interna del modelo, y a menudo es implementada como un árbol. La aplicación del cliente (a menudo un script o applet) accede al documento parseado por medio de las funciones y métodos definidos en el API DOM.

La especificación de DOM Level 1 se organiza en tres partes:

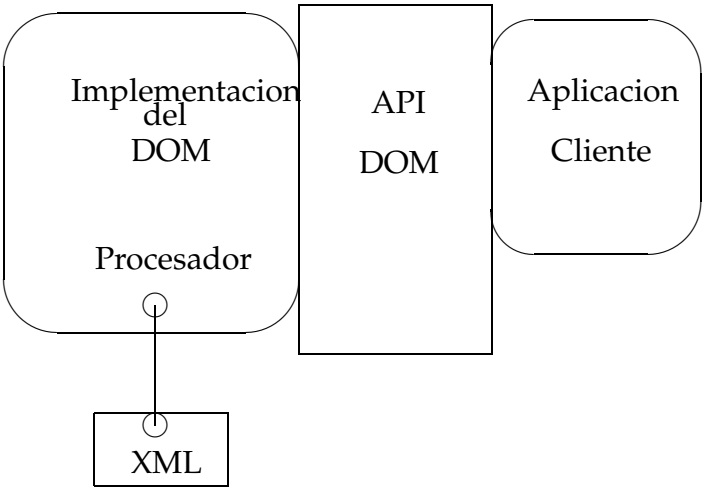

Figura 2.5: Arquitectura de un procesador HTML/XML que implementa el interface DOM

- Interfaces fundamentales, que son las interfaces a estos objetos que son comunes a los documentos HTML y XML, tales como atributos, elementos, comentarios, texto.
- Interfaces extendidas, que son para objetos XML, tales como `Entity` y `CDATA Sections`.
- Los interfaces HTML, que suministran interfaces especiales para el conjunto de marcas de HTML.

2.6 Perl

2.6.1 Introducción

Perl es un lenguaje de alto nivel. Este lenguaje permite hacer fácilmente las tareas cotidianas como son el manejo de números y texto, ficheros y directorios, computadoras y redes de trabajo, y especialmente programas. Esto quiere decir que Perl permite hacer fácilmente aquellas cosas comunes, pero no haciendo imposible aquellas tareas complejas. Esta es la característica principal de Perl, puesto que la mayoría de los lenguajes de ordenador permiten hacer cualquier aplicación, pero no siempre estos, permiten hacer ciertos trabajos de forma fácil o cómoda, sin que se vean reducidas las posibilidades de realización de otras tareas más complicadas.

Otra gran ventaja de Perl, es que es potable a la mayoría de los sistemas operativos actuales, como VMS, OS/2, Windows, MS-DOS, etc, además de, por supuesto, para los sistemas UNIX y derivados (como Linux), para los cuales fue inicialmente diseñado. Además esta portabilidad.

Perl es especialmente conocido entre programadores de sistemas y desarrolladores de Web. Pero, Perl no sirve únicamente sirve para procesar texto, sino que se está convirtiendo en un lenguaje de programación de propósito general, utilizándose en cualquier campo de la ciencia imaginable, desde la ingeniería aéreo-espacial hasta la biología molecular, desde la manipulación de bases de datos hasta gestores de redes cliente-servidor, etc.

Existen varias razones para explicar el auge que Perl está experimentando. Por una lado, podemos destacar que Perl es gratuito, de modo que cualquier persona puede tener acceso a él. Pero, no sólo nos podemos quedar con esto, puesto que no sólo por el hecho de ser gratuito, tiene que tener éxito, como ampliamente nos demuestra la experiencia. Además, Perl proporciona al usuario una gran libertad de expresión, permitiendo la optimización en varias direcciones: velocidad de ejecución o de programación, de fácil lectura o de mantener, portabilidad, etc. En pocas palabras, Perl es, al mismo tiempo, un lenguaje simple y rico.

Decimos que Perl es un lenguaje simple porque no es necesario tener muchos conocimientos para compilar un programa, puesto que este se puede ejecutar del mismo modo que un shell script. La estructura de datos que utiliza es muy fácil de entender, y no se imponen restricciones arbitrarias a los datos (por ejemplo los arrays y las cadenas pueden crecer en su tamaño de forma dinámica, mientras haya memoria disponible en la máquina). Perl no adopta una sintaxis y una semántica nueva, sino que deriva de lenguajes ampliamente conocidos por todo el mundo, como C. Además, un usuario que quiera escribir una aplicación, no tiene por qué conocer todos los entresijos del lenguaje, para poder construir aplicaciones útiles (se puede ir de menos a más).

Por otro lado, decimos que Perl es un lenguaje rico, porque, no sólo permite de forma cómoda procesar texto, crear y obtener datos dinámicos e informes con formato que accedan a estos datos, sino que también permite la manipulación de ficheros y directorios, el desarrollo de procesos de gestión, administradores de bases de datos, aplicaciones cliente-servidor, programas de seguridad, aplicaciones Web, etc. Perl es, incluso, un lenguaje orientado a objetos, además de permitir la programación funcional. Está diseñado, desde el principio, para que sea extensible de forma modular. Existe la posibilidad de incrustar código Perl en otros lenguajes y viceversa, así como, usar módulos externos como si fuesen construidos en Perl, gracias al mecanismo de importación.

Perl, no es estrictamente hablando, un lenguaje interpretado, tal como puede ocurrir con el shell script. Perl compila previamente todo el programa en un formato intermedio. Al igual que otros compiladores, se llevan a cabo un conjunto de optimizaciones, así como la generación de los mensajes de errores sintácticos y semánticos existentes. Si un programa Perl, pasa correctamente todas las fases de compilación, se pasa el código intermedio al interprete, para que pueda ser ejecutado. También posible transformar este código intermedio en código C o código binario. Aunque pueda parecer, todo esto un poco complejo, la verdad es que todos estos pasos se llevan a cabo de forma muy eficiente, por lo que suele tardar pocos segundos una compilación-ejecución típica.

2.6.2 Características de Perl

Nos vamos a detener un poco más en ver algunas de las características propias de Perl, las cuales han sido aprovechadas directamente para la realización del WAG, entre ellas: los tipos de datos, expresiones regulares , entrada/salida, funciones, paquetes, módulos y clases de objetos.

2.6.2.1 Tipos de datos

Perl posee tres tipos de datos básicos: escalares, arrays de escalares y hashes de escalares. Este último tipo, se conoce también como arrays asociativos. No obstante, Perl permite definir nuevos tipos de datos más complejos.

Los escalares pueden almacenar un único dato, fundamentalmente un número, una cadena o una referencia a otro elemento). Un array es una lista ordenada de escalares, los cuales pueden ser accedidos mediante una referencia numérica. Por último, un array asociativo (hashed) es un conjunto no ordenado de pares clave/valor, que puede ser accedido utilizando una cadena (la clave) como referencia, para obtener finalmente el valor asociado a una clave dada.

Como es lógico, las variables serán de un tipo u otro según el tipo de datos que almacenen. Podemos distinguir entre los distintos tipos de variables:

2.6.2.1.1 ***Escalares*** A estas variables se le puede asignar cualquier valor escalar ,es decir, un valor único, que puede ser un número, una cadena o una referencia a otro dato. Las variables escalares no poseen tipo, puesto que no es posible declarar un escalar para que sea de tipo numérico o de tipo cadena. Los escalares se comportarán en Perl como números o como cadena , de forma transparente y dependiendo del contexto en que se utilicen. El contexto viene impuesto por los operadores, los cuales requieren un escalar numérico o bien de tipo cadena. Por otro lado, a diferencia de los números y cadenas, las referencias a datos, están fuertemente tipadas, de forma que no es posible, por ejemplo, transformar una referencia a un array en una referencia a un hashed. Estas referencias a objetos se utilizarán para crear tipos más complejos, incluyendo objetos definidos por el usuario. Las variables escalares deben precedidas del signo $.

```
$proyecto=”WAG”; # Escalar de tipo cadena
$numero=”3”; # Escalar de tipo numérico entero
$pi=3.1416; # Escalar de tipo numérico pun-
to flotante
```

Veamos ahora un ejemplo de como se puede usar un escalar en distintos contextos.

```
print “Versión “,$numero;
# Esto sacará por pantalla la cadena “Versión 3”
$numero=$numero+1;
print “Versión “,$numero;
#Esto sacará por pantalla la cadena “Versión 4”
```

2.6.2.1.2 Arrays Un array es una lista ordenada de escalares, los cuales son indexados por la posición del escalar dentro de la lista. La lista puede contener, números, cadenas o ambas cosas; incluso puede contener una referencia a otra lista. Para asignar una lista de valores a un array, separaremos los datos mediante comas e irán encerrados entre paréntesis.

```
@amigos=(“Juan”,”Pedro”,”Antonio”,”Miguel”);
```

Si queremos acceder a alguno de los elementos del array, deberemos indexarlo mediante un número entero comprendido entre 0 y n, donde n es el número de elementos que contiene el array menos 1 (igual que ocurre con los array en C). Como el elemento al que estamos accediendo es un escalar, debe de ir precedido del signo $.

```
$amigos[0]=”Juan”;
$amigos[1]=”Pedro”;
$amigos[2]=”Antonio”;
$amigos[3]=”Miguel”;
# Este código da el mismo resultado que la asig-
nación anterior
# utilizando una lista de elementos
```

Puesto que los arrays son listas ordenadas de elementos, es posible utilizar dos funciones muy útiles, como son la función push y la función pop. Mediante el uso de estas dos funciones podemos utilizar los arrays como si de una pila se tratase. La función push añade un elemento en la cima de la pila, y mediante pop, obtenemos el elemento de la cima.

2.6.2.1.3 Hashes Un hash o array asociativo es un conjunto no ordenado de escalares, los cuales son accedidos mediante una cadena, que actúa como clave, y que está asociada a un escalar.Un hash no tiene principio ni final, a diferencia de los arrays.Los arrays asociativos son extremadamente útiles.

Se puede asignar una lista a un hash, del mismo modo que a un array, pero los elementos de la lista son interpretados como pares clave/valor. Veamos un ejemplo:

```
%altura=(“Juan”, 1.78, ”Pedro”, 1.63, ”Anto-
nio”, 1.91, ”Miquel”, 1.87);
```

Para una mayor claridad, se puede utilizar el operador ’=>’ para establecer la correspondencia entre clave y valor.

```
%altura=(“Juan”=>1.78,
         “Pedro”=>1.63,
         “Antonio”=>1.91,
         “Miguel”=>1.87);
```

Si nosotros usamos un hash en un contexto de lista, nos devolverá una lista de pares clave/dato, en cualquier orden. Mediante el uso de la función key, podemos extraer una lista con todas las claves que forman parte del hash.

Se puede acceder a un elemento del hash, encerrando entre llaves la clave asociada a dicho elemento. Por ejemplo, si queremos acceder a la altura de "Juan", procederemos como sigue:

```
$altura{"Juan"}=1.76; # Permite modificar la al-
tura asociada
                      # a la clave "Juan"
```

2.6.2.1.4 Tipo comodín (entrada a la tabla de nombres)

2.6.2.2 Expresiones regulares.

Las expresiones regulares son utilizadas en muchos programas de Unix, tales como grep,sed,awk y editores como vi o emacs. Una expresión regular permite describir a un conjunto de cadenas, sin la necesidad de tener que listar todas las cadenas en su conjunto.

Perl utiliza las expresiones regulares de varios modos. Por un lado se utilizarán en condiciones, para determinar si una cadena encaja dentro de un patrón. Para ello se utiliza los operadores de pattern-matching. Por otro lado, se pueden utilizar las expresiones regulares, para encontrar subcadenas dentro de una cadena, permitiendo sustituir dicha subcadena por otra. Para esta tarea se utilizará el operador de sustitución.

También se puede utilizar estas expresiones regulares para determinar que partes de una cadena no encajan dentro de un patrón, permitiendo así, por ejemplo, determinar fronteras dentro de una cadena. Para esta tarea se utiliza el operador split.

La sintaxis de Pattern-matching son:

- Búsqueda

  ```
  m/expresión-regular/modificador o /expresión-
  regular/modificador
  ```

- sustitución

  ```
  s/expresión-regular/sustitución/modificador
  ```

- partición

  ```
  split /expresión-
  regular/modificador, expresión, limite
  ```

No merece la pena entrar en detalle de como funciona y se utiliza las expresiones regulares, pero si podemos ver algunos ejemplos para mostrar qué cosas se pueden hacer.

```
if $cadena=~/hola/ {
     código....
}
# Permite ejecutar el código, si el escalar $ca-
dena almacena una
# cadena que contenga la subcadena “hola”
$cadena=~s/hola/adiós/g;
# Sustituye todas las ocurrencias de la subcade-
na “hola” por
# la subcadena “adiós”
@numeros=split(/-/,“1-2-3-4-5”;
# Divide la cadena en una lista : (1,2,3,4,5)
```

2.6.2.3 Entrada / salida.Filehandle

En la mayoría de las aplicaciones, es necesario comunicarse con el mundo exterior. Perl permite comunicarse con el exterior mediante los filehandle(descriptor de ficheros, en castellano). Un descriptor de ficheros, no es más que asociar un nombre a un fichero, dispositivo, socket, pipe, etc, para ayudarnos a saber con cual de ellos estamos tratando; además nos ocultará las características complejas de estos elementos.

Los descriptores de ficheros, facilitan el envío y recogida de datos sobre diferentes fuentes. Se puede asociar un nombre a un fichero mediante la función open. A esta función le proporcionamos el descriptor del fichero y el nombre del fichero al cual se quiere asociar. Existen descriptores de ficheros predefinidos por Perl, tales como STDIN(entrada estándar del sistema),STDOUT (salida estándar del sistema) y STDERR(salida estándar de error). Al abrir un fichero, también es necesario especificar de qué modo queremos utilizarlo: entrada, salida, tubería.

```
open (FICHERO,”nombre-del-
fichero”);# Abre un fichero existente
                                        # y los aso-
cia al descriptor
```

Una vez que un descriptor es asociado a un fichero, este permanecerá asociado a este hasta que se use la función close o bien se asocie a un fichero nuevo.

```
close(FICHERO); # Desvincula el descrip-
tor del fichero
```

Podremos leer lineas desde un fichero utilizando el operador <>. Podemos escribir en un fichero mediante la función print.

```
print STDOUT “Introduzca un número:“; # Pi-
de un numero
$numero=<STDIN>; # recoge el número
$print STDOUT “El número es: $numero\n”; # Mues-
tra el número
$linea=<FICHERO>; # recoge la primera lin-
ea del fichero
@resto=<FICHERO>; # recoge el resto de las líneas del fichero
```

Otro modo importante de entrada de datos es median el uso del operador de entrada de comandos. Este comando permite que la cadena que encierra sea interpretada como un comando del shell, de modo que el resultado será la salida que genera dicho comando. En contexto de escalar, todo la salida generada se devuelve como una cadena, mientras que en un contexto de lista, se devuelve una lista de valores, una por cada línea de salida. (Se puede modificar el valor de la variable $/ para establecer diferentes tipos de terminadores de línea.)

2.6.2.4 Funciones (o subrutinas)

Perl, al igual que otros lenguajes, permite la definición de subrutinas por parte del usuario. Las subrutinas pueden ser definidas en cualquier parte del programa principal, cargadas desde otro fichero, mediante las palabras reservadas do, requiere o use. También se pueden crear subrutinas anónimas, pudiendo accederse a ellas únicamente a través de referencias. Así, se puede invocar indirectamente una subrutina a través de una variable, la cual contenga el nombre de una función, o bien, una referencia a dicha función.

Declaración de una subrutina:

```
sub NOMBRE;
sub NOMBRE (PROTOTIPO); # Declaración con pro-
totipo
```

Declaración y definición de una subrutina:

```
sub NOMBRE BLOQUE-CÓDIGO
sub NOMBRE (PROTOTIPO) BLOQUE-
CÓDIGO # Declaración y definición con prototipo
```

Importación de subrutinas definidas en otros paquetes:

```
use PAQUETE qw(NOMBRE1 NOMBRE2 NOMBRE3 ...);
```

Llamada a una subrutina de forma directa:

```
NOMBRE(LISTA); # & es opcional, si se us-
an los paréntesis
NOMBRE LISTA; # paréntesis op-
cionales si fue declarada o importada
&NOMBRE; # pasa como lista el contenido actu-
al de @_
```

Llamada a una subrutina de forma indirecta:

```
&$subref(LISTA); # & no es opcional en lla-
madas indirectas
&$subref; # se pasa el contenido actual de @_
```

En cuanto al paso de parámetros, tenemos que tener en cuenta lo siguiente:

- Los parámetros se pasan como una lista de escalares. Dentro de la subrutina, se puede acceder a los parámetros mediante el array especial @_ .
- Si pasamos varios arrays o varios arrays asociativos, se fundirán en un lista, de forma, perdiendo así su identidad. No obstante, la mayoría de las ocasiones, esta transformación es útil. Existe un mecanismo para evitar que esto ocurra.
- Los parámetros de salida se devuelven en forma de lista.
- Tanto la lista de parámetros de entrada, como la de salida, pueden tener un tamaño variable. Esto siempre es cierto, a menos que se utilice un prototipo a la hora de la declaración.
- El valor de una subrutina es el resultado de la última expresión. Es posible devolver valores explícitamente, mediante el uso de la sentencia return (además fuerza la salida de la rutina, independientemente del lugar dónde se encuentre.)

Hasta ahora, sólo hemos visto cómo pasar parámetros por valor, de modo que sólo se comportarán como parámetros de entrada. Existen otros dos tipos de paso de parámetros que debemos resaltar. Por un lado, tenemos el paso de entradas a la tabla de nombres; este mecanismo, permite modificar el valor del parámetro que se pasa (parámetro de entrada/salida). También, permite que al pasar varios arrays como parámetros de entrada, estos se puedan recuperar, impidiendo que estos se fundan. Por otro lado, el paso de referencias es similar al concepto de puntero en el lenguaje C . Con este método se pueden conseguir las mismas ventajas que con el método de paso de entradas a la tabla de nombres.

2.6.2.5 Paquetes y módulos

Un paquete es una sección de código, el cual tiene asociado su propia tabla de símbolos. De esta forma se protege al paquete de las posibles colisiones entre variables de otros paquetes. El código se compila siempre en base a un paquete activo. El paquete activo inicial, se le denomina paquete principal. Se puede cambiar de paquete activo mediante la declaración 'package'. El paquete activo, por tanto, determina solamente cual el la tabla de símbolos activa en cada momento (esto es muy importante). Un paquete comprende todo el código encerrado entre llaves, tras una declaración 'package' (o bien, todo el código comprendido hasta que se encuentre la siguiente declaración 'package'). Los paquetes pueden estar anidados. Veamos ahora algunos ejemplos de acceso a variables, según al paquete al que pertenezcan:

```
$Paquete::Variable # Este será la estructura gen-
eral de acceso a una variable
$Variable # Accede al paquete activo
$NODO::atrib # Accede a la variable cuyo nom-
bre es 'atrib' del paquete 'NODO'
$::apellido # Accede a la variable de nom-
bre 'apellido' perteneciente al paquete principal
$main::apellido # Igual que el caso anterior
$NODO::ATRIBUTO::var # Accede a la vari-
able de nombre 'var' del paquete 'ATRIBUTO'
# que está incluido dentro del paquete 'NODO'
```

Un módulo es un paquete que está definido en un fichero de librería cuyo nombre coincide con el nombre del paquete (terminado en .pm). Un módulo puede incluir un método para exportar símbolos para cualquier otro módulo que lo utilice. También puede funcionar como una definición de clase, haciendo que sus operaciones estén disponibles de forma, mediante llamadas a métodos de clase y de los objetos asociados a la clase, sin necesidad de exportar explícitamente ningún símbolo. Los módulos se incluyen en un programa con 'use':

```
use Modulo;
```

o

```
use Modulo Lista;
```

Si no especificamos lista, todos los símbolos exportados en el modulo serán importados por el paquete que lo utiliza. Si por el contrario la especificamos, sólo se podrán usar aquellos que se incluyan en la lista.

2.6.2.6 Orientación a objetos en Perl

En Perl, una clase no es más que un módulo que contiene una serie de subrutinas(esperan un objeto o el nombre de una clase), las cuales serán utilizadas como métodos.

En perl no existe una sintaxis expresamente creada para la definición de clases, ni para los constructores. Tampoco existe el concepto de métodos privados y públicos, a diferencia de lo que ocurre en lenguajes como C++, Eiffel, etc.

No obstante, Perl posee mecanismos para poder actuar como un lenguaje orientado a objetos típico.No es objetivo de este trabajo ver en detalle todos los mecanismos existentes en Perl.Veamos un ejemplo de cómo se implementa una clase (clase 'polígono' en Perl):

```
package Poligono;
# Método constructor;
sub new {
  return bless {};

}
# Método 'dibujar'
sub dibujar {
...
   CODIGO DEL MÉTODO
...
}
```

Si queremos crear una subclase:

```
package Rectangulo;
@ISA=(Poligono);
# Método constructor;
sub new {
  return bless {};

}
```

El array @ISA, nos permite implementar el concepto de herencia, puesto que permite indicar en qué paquete se deben buscar los métodos que no se encuentren en el paquete actual.

Existen dos alternativas para invocar un método:

- Forma de objeto indirecto:

  ```
  METODO CLASE_O_INSTANCIA LISTA;
  ```

- Sintáxis orientada a objetos:

  ```
  CLASE_O_INSTANCIA->METODO(LISTA);
  ```

Veamos, pues, algunos ejemplos de cómo se invoca un método:

```
use Polígono;
$var=new Poligono; # Crea una instan-
cia de la clase 'Polígono';
$var=Poligono-
>new(); # Es lo mismo que lo anterior;
$var->dibujar(); # Invoca al método de instan-
cia 'dibujar'
Polígono-
>dibujar(); # Invoca al método de clase 'dibujar'
```

2.6.3 Uso de Perl dentro del WAG

Parte del Proyecto WAG ha sido implementado en Perl, concretamente, la parte relacionada con el parser del lenguaje 'XML-DB' . Las razones principales de la elección de Perl, para llevar a cabo esta función de perseador, ha sido:

- Es un lenguaje que facilita el tratamiento de texto, la creación dinámica de dato y el tratamiento de ficheros y direcctorios.
- Es mucho más portable que otros lenguajes.
- Posee una sintaxis muy parecida a lenguajes muy extendidos como C.
- Permite la utilización del módulo Perl DOM, para el tratamiento de ficheros con formato XML.
- Es un lenguaje que permite la programación Orientada a Objetos.

Capítulo 3

Diseño del sistema

En esta sección vamos a ver con más detalle el funcionamiento del WAG, cómo está estructurado y cómo se relacionan entre si, los distintos subsistemas que lo constituyen.

El objetivo de WAG, no es otro que facilitar la creación de una Aplicación Web que interaccione con una base de datos. La Aplicación Web permitirá al usuario final hacer consultas a la base de datos, así como hacer un mantenimiento de los datos que la base de datos contiene.

Como podemos observar, existe un pilar básico, sobre el cual se apoya el WAG, que es sin duda el 'Sistema Gestor de Base de Datos' (SGBD) ,que como ya sabemos, en este caso esta basado en PostgreSQL.

Debido a que el lenguaje XML es utilizado para la especificación de la estructura de la base de datos, el WAG necesita de un sistema externo que permita tratar ficheros, con formato XML, de forma cómoda (en nuestro caso usamos las librerías DOM).

Otro sistema auxiliar para el WAG es el servidor Web, que será el encargado de recoger las páginas Web que conforman la aplicación final, y la hará llegar a su destino; pero si vamos un poco más allá, comprobaremos que existe un intermediario entre el WAG y el servidor Web.El intermediario al que hacemos referencia es el intérprete de ficheros(PHP), el cual generará definitivamente la página Web que será luego suministrada por el servidor Web.

Si echamos un vistazo dentro del WAG, se pueden diferenciar claramente tres partes. Por un lado tenemos el subsistema encargado de interpretar los ficheros de entrada (Parser), los cuales están escritos en XML. Otro subsistema será el encargado de interaccionar con la base de datos(Envoltorio del Servidor de Base de datos) y por último, el subsistema encargado de generar el conjunto de páginas Web, que conforman la aplicación Web (Generador de Aplicaciones).

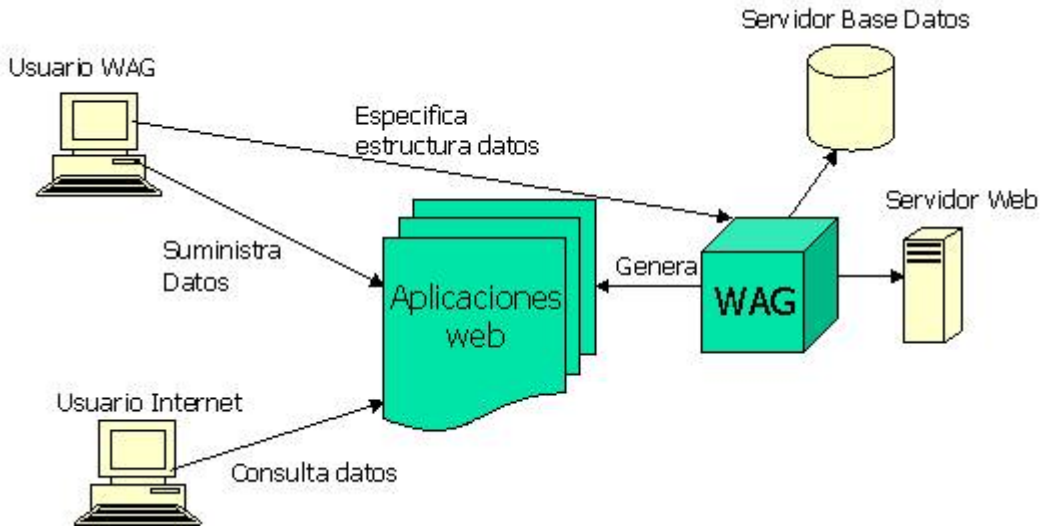

Figura 3.1: Generador de Aplicaciones Web (WAG)

3.1 División en subsistemas

3.1.1 Gestor del WAG

La descripción o especificación de una base de datos la realiza el usuario, mediante un fichero escrito en lenguaje de especificación de base de datos XML-DB (será estudiado con detalle en el Manual de Usuario).

Este subsistema es el encargado de recoger un fichero de entrada escrito en XML-DB, y de procesar dicho fichero, dando como resultado una respuesta sobre el SGBD, de modo que la estructura de la base de datos que se ha especificado en el fichero de entrada, quede almacenada físicamente en el SGBD. Otra posibilidad es la de transformar el fichero de especificación de la estructura de la base de datos, en un fichero script de SQL-PostgreSQL, de modo que este puede ser almacenado y utilizado posteriormente.

Otra tarea que realiza el 'Parser' es la de registrar cuál es la estructura de cada una de las bases de datos especificadas por los usuarios, a qué usuario está asociada y algún otro parámetro de interés. Esta información, que es muy útil, se le denomina meta-información del WAG. Será el subsistema 'Generador de aplicaciones', el que haga uso de esta meta-información, como más adelante podremos ver.

Este subsistema queda encapsulado en una herramienta, denominada 'wag-manager, que será utilizada por el administrador del sistema.

3.1.2 Generador de Aplicaciones

Una aplicación Web, será un conjunto de páginas Web que van a permitir a un usuario final, el acceso a los datos que una base de datos contiene. Permite la consulta de estos datos, así como el mantenimiento de los mismos (inserción, modificación y eliminación de dichos datos).

Es este el subsistema que se encarga de generar la Aplicación Web, puesto que es aquí donde se generarán las páginas que conforman la Aplicación Web. Por tanto, el generador de aplicaciones tiene entre otras, que conocer la estructura de la base de datos (tablas, campos, tipos de los campos,índices, claves primarias, etc.), para poder generar páginas que sean acordes con la base de datos. Para conocer la estructura de la base de datos se accederá a la meta-información del WAG, la cual se encuentra alojada en una base de datos especial, que será creada y mantenida por el 'Parser'.

Otra de las tareas principales del generador de aplicaciones será acceder a los datos que contiene una base de datos definida por un usuario, realizando inserciones, consultas, actualizaciones y eliminaciones, razón por la que, nuevamente, tendrá que interaccionar con la base de datos.

El 'Generador de aplicaciones' tiene la posibilidad de generar páginas Web con un formato estándar, el cual será el utilizado por defecto, a menos que el usuario desarrolle el suyo propio. Para dar un propio formato (aspecto) particular a las páginas, que el generador de aplicaciones debe generar, es necesario que tome como entrada unos ficheros escritos en HTML-DB (lenguaje que será descrito con detalle en el manual de usuario). Es por esta razón, por la que el 'Generador de aplicaciones' debe de interpretar estos ficheros y dar como salida las páginas Web que constituyen la aplicación.

3.1.3 Interface Base de Datos

A diferencia de los dos subsistemas citados anteriormente, este subsistema no se puede distinguir tan claramente, puesto que forma parte de ambos. Sin embargo, tiene una función muy importante, y es obvio que debe tratarse de como un subsistema con entidad propia.

Su primera función es la de ocultar aquellas características especificas del SGBD que se está utilizando. Esto permite independizar el sistema WAG, de un SGBD determinado. Esto permitirá, que el sistema WAG pueda trabajar con cualquier otro SGBD, simplemente reemplazando dicho envoltorio.

Otra función de este subsistema es centralizar y controlar todos los accesos a las bases de datos.

Como anteriormente se ha citado, este subsistema se encuentra subdividido en dos:

1. Interface del SGDB para el Parser.
2. Interface del SGDB para el Generador de Aplicaciones.

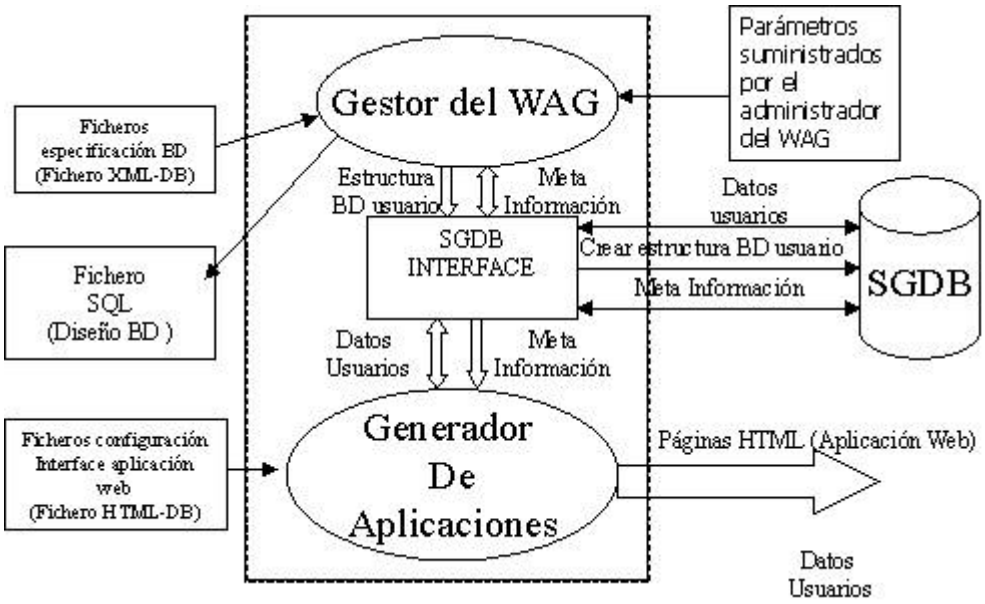

Figura 3.2: División en subsistemas del WAG

Debemos resaltar que esta subdivisión se debe a necesidades técnicas de implementación, puesto que cumplen las mismas funciones y poseen idéntica lógica de funcionamiento .

3.2 Bases de datos

La idea fundamental del WAG es que un usuario pueda publicar un conjunto de datos a través de una aplicación Web. Es por tanto, sin duda necesario un lugar donde estos datos puedan residir. Será una base de datos, el lugar donde dichos datos van a estar alojados.

Una base de datos debe estar asociada a un único usuario. Será el propio usuario el que decida qué estructura debe tener, de forma que se adapte a los datos que este quiere publicar. También será el propio usuario el que debe suministrar y mantener los datos que se almacenan en la base de datos.

Para el WAG, una base de datos es un conjunto de tablas de datos. Serán finalmente estas tablas las que almacenan los datos en forma de registros.

Todas las bases de datos mantenidas por el WAG son gestionadas por un SGBD. El SGBD es un sistema externo al WAG. El WAG debe de interaccionar con el SGBD, para almacenar y acceder a los datos. Esta función de interacción con el SGBD es la que lleva a cabo el Envoltorio de base de datos.

3.3 Funcionamiento del Parser

El Parser permite crear una base de datos con la estructura que un usuario especifica. Para ello se le proporcionará un fichero escrito en lenguaje XML-DB.En este fichero, básicamente, se especifica el nombre de la base de datos, el conjunto de tablas y los campos que estas deben tener. . A partir de esta información, el Parser transforma el fichero de entrada en un conjunto de sentencias SQL, que serán interpretadas por el SGBD para crear la base de datos. Este conjunto de sentencias SQL serán enviadas al SGBD a través del Envoltorio de Base de Datos. El conjunto de sentencias SQL, también pueden ser almacenadas en un fichero (script de Sql), para una posterior utilización.

El proceso de traducción de XML-DB a SQL, es realizado construyendo, recorriendo e interpretando una estructura jerárquica en forma de árbol sintáctico, con la colaboración del módulo DOM (Document Object Model) . Este módulo permite transformar cualquier documento XML en un arbol.

Una vez que la secuencia de sentencias SQL es enviada al SGBD y es ejecutada por este, la base de datos ,especificada por el usuario, queda creada físicamente.

Posteriormente a la creación de la base de datos, el 'Parser' procede a registrar cuál es la estructura de la base de datos especificada en fichero de entrada. Esta información es proporcionada por el SGBD. A esta información se le denomina meta-información de la base de datos.

Ahora, el Parser transforma la meta-información de la base de datos y la transforma en meta-información del WAG. La meta-información del WAG, esta compuesta por la meta-información de la Base de Datos, más información adicional de interés para el WAG (configuración del WAG, a qué usuario pertenece cada base de datos, etc). Esta información adicional se le suministrará al Parser en forma de parámetros y/o de ficheros de configuración. Estos ficheros de configuración pueden ser creados por el Administrador del sistema o , incluso, por el propio usuario. Es, por consiguiente, el Parser, el encargado del mantenimiento de la meta-información del WAG.

La pregunta siguiente, que nos hacemos es, ¿dónde se almacenará la meta-información del WAG?. Pues, bien, se utilizará una base de datos creada y mantenida por el WAG, de forma transparente a los usuarios, para llevar a cabo esta tarea.

3.4 Funcionamiento del Generador de Aplicaciones (AG)

Para entender mejor cuál es el funcionamiento del AG, es necesario conocer cómo está construido y cómo interacciona con el Servidor de Web:

- El AG es un conjunto de ficheros escritos en PHP, e interpretados por un interprete de PHP.

- El conjunto de ficheros PHP, contienen la lógica necesaria para generar los ficheros HTML, correspondientes a cada una de las páginas Web que constituyen la Aplicación Web.
- El interprete de PHP, transforma los ficheros PHP en ficheros HTML, los cuales son enviados al Servidor de Web.
- Finalmente, el fichero HTML es enviado al cliente Web

Nos centraremos ahora, cuál es la lógica encerrada en el conjunto de ficheros PHP, que constituyen al AG. Veamos, pues, el modo de operar.

En primer lugar, el AG busca los ficheros HTML-DB (HTML extendido, que permite al usuario dar un formato personalizado a las páginas Web de la Aplicación) asociados a la Aplicación Web.

Seguidamente, se accede a la meta-información del WAG, la cual proporciona la estructura de la base de datos que se desea consultar y otros datos de configuración.

Una vez que la estructura de la base de datos es conocida, se accederán a los datos almacenados en ella, como respuesta a la petición hecha por un usuario.

Finalmente, la información proporcionada por los ficheros HTML-DB, junto con la meta-información del WAG y los datos recogidos de la base de datos, van a permitir la creación de un fichero HTML. Este fichero HTML se va a corresponder con una página Web de la Aplicación Web.

Por tanto, como podemos ver, el AG va generando las páginas Web de forma dinámica, es decir, a medida que el usuario va navegando por la Aplicación Web.

Capítulo 4

Implementación del sistema

4.1 Implementación del 'Gestor del WAG'

4.1.1 Introducción

Este subsistema permite al usuario la configuración del sistema WAG, así como llevar a cabo el mantenimiento del mismo. Esencialmente, el subsistema ofrece al administrador del WAG la posibilidad de dar de alta usuarios en el sistema, definir y crear bases de datos asociadas a estos usuarios, así como establecer algunos parámetros de configuración sobre las aplicaciones que serán generadas por el sistema. De este modo, este subsistema es el encargado de interpretar los ficheros de especificación de bases de datos (formato XML-DB) e interpretarlos. Tras ser interpretado un fichero de especificación, todo estará listo para que sea dada de alta en el servidor de base de datos, la base de datos especificada. También, es posible la obtención de un fichero SQL equivalente al fichero de especificación suministrado.

El subsistema, deberá de almacenar información relacionada con la estructuras de las bases de datos, usuarios existentes, bases de datos asociadas a dichos usuario. Además deberá quedar registrado cuál es el directorio de trabajo de un usuario, los ficheros 'HTML-DB' que dicho usuario ha proporcionado y que tratamiento dará a dichos ficheros el AG. Toda esta información relacionada con el funcionamiento del propio sistema se considera la meta-información del WAG.

Por tanto, el subsistema, tendrá cuatro tareas básicas:

- Recoger la información suministrada por el usuario administrador.
- Por otro lado, crear una base de datos sobre el SGBD, tomando como base el fichero de especificación XML-DB.
- Compilar un fichero XML-DB y transformarlo en un fichero SQL equivalente.
- Gestionar toda la información de interés para el sistema.

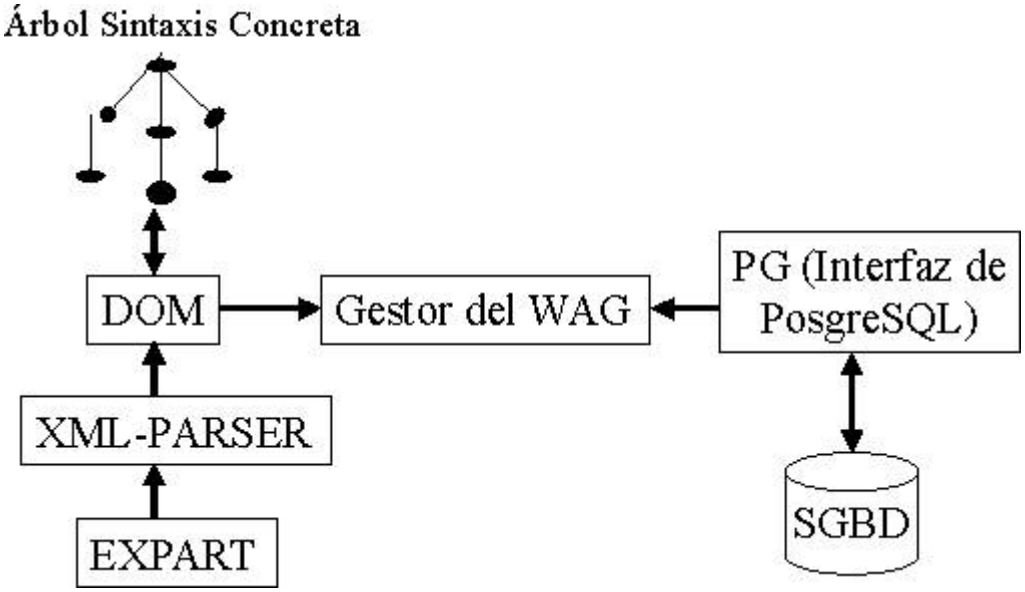

Figura 4.1: Visión global del Gestor del WAG

4.1.2 Estructura

El 'Gestor del WAG' está implementado haciendo uso del lenguaje Perl. Esto quiere decir que será necesario un interprete de Perl, para que pueda ser ejecutado.

El subsistema se apoya para su correcto funcionamiento sobre dos módulos Perl : DOM y Pg. El primero, nos ofrece una interfaz que permitirá manipular un fichero XML. El segundo paquete, permite el acceso a la interfaz libpq del servidor de base de datos PostgreSQL.Veamos cómo interacciona el subsistema con dichos módulos:

Como se puede observar, DOM recoge un fichero de texto XML y construye un árbol sintáctico. El subsistema, recorrerá este árbol haciendo uso de la interfaz que le ofrece el módulo DOM. Como resultado, el 'Parser' generará un fichero de texto SQL. Por otro lado, se produce una interacción con el servidor de bases de datos, haciendo uso de la interfaz suministrada por el módulo Pg. La interacción con el servidor de base de datos es necesaria para gestionar la meta-información y para gestionar las bases de datos definidas por los usuarios.

El subsistema 'Gestor del WAG' , a su vez, se encuentra subdivido en varios módulos relacionados del siguiente modo:

Analicemos detenidamente cada uno de estos módulos:

1. **Interfaz del usuario:** toma la información suministrada por el usuario y la analiza. Tras ser procesada esta información, se distribuye hacia el

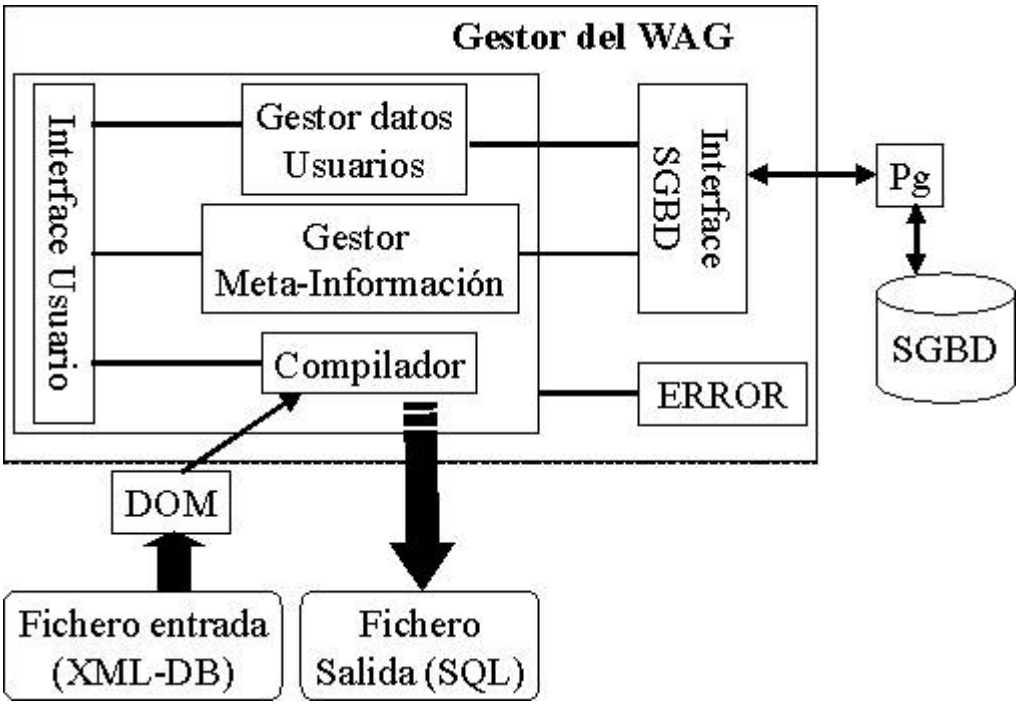

Figura 4.2: División modular del subsistema 'Gestor del WAG'

'Gestor de meta-información' y el 'Compilador'. Estos módulos generan mensajes informativos para el usuario, los cuales son enviados a la salida estándar.

2. **Compilador**: Este módulo recorre el arbol sintáctico construido por el módulo DOM, a partir del fichero XML de entrada, haciendo uso de las funciones que dicho módulo le proporciona. Durante el recorrido del árbol sintáctico, se realiza la validación de éste y se construye el árbol de sintaxis abstracta .Por último, y partiendo del árbol de sintaxis abstracta, se lleva a cabo la generación de código SQL.

3. **Gestor de los datos de los usuario:** Es el encargado de crear y mantener las bases de datos definidas por los usuarios, por tanto, es necesario interaccionar con el servidor de base de datos a través de la interfaz de BD.

4. **Gestor de meta-información:** Permita mantener actualizada toda la información relacionada con el propio sistema. Esta información reside en una base de datos, creada y mantenida por el propio sistema. Es necesaria, por tanto, la interacción con el servidor de base de datos, haciendo uso del la 'Interfaz de SGBD'.

5. **Interfaz de SGBD:** Nos permite interactuar con un servidor de base de datos, aislando al resto del sistema de las características propias de dicho servidor (en este caso con PostgreSQL). De este modo, cambiar el

servidor de base de datos por otro distinto, sólo implica reemplazar este módulo por otro.

Todos los ficheros que componen el subsistema 'Gestor del Wag' se encuentran ubicados en el directorio 'manager/'

4.1.3 Interfaz del usuario

Este módulo se encuentra implementada mediante el fichero 'wag_manager.pl'. Es el encargado de recoger información proporcionada por el usuario administrador en forma de parámetros , para ser posteriormente analizados y ejecutados. Por otro lado, se ofrecen al administrador un juego de mensajes, que le permitirán conocer cuál es el resultado obtenido tras la interacción con el sistema.

4.1.4 Compilador

Un proceso de compilación consta de las siguientes fases: análisis léxico, análisis sintáctico, análisis semántico y generación de código. En nuestro caso, DOM hace por nosotros las dos primeras fases (análisis léxico y sintáctico). El resto de las fases deben ser implementadas.

Tras la etapa de análisis léxico, DOM nos proporciona un árbol sintáctico, que puede ser recorrido gracias a las funciones que proporciona. En este punto es dónde entra en juego el compilador del WAG.

- El árbol de sintáctico proporcionado por DOM debe ser validado, es decir debe cumplir con las restricciones impuestas sobre el lenguaje XML-DB. Por restricciones del lenguaje se entiende: estructura del documento XML-DB (un elemento sólo podrá tener un determinado tipo de elementos), número máximo de apariciones de un elemento, obligatoriedad de un elemento, unicidad de un elemento, atributos que posee un elemento y utilización de los mismos.

  Para especificar las restricciones del lenguaje se utiliza una estructura jerárquica en perl, que denominaremos 'estructura del documento'. Por tanto, la validación se realiza contrastando el árbol sintáctico contra la 'estructura del documento'.

- Seguidamente, se construye el árbol de sintaxis abstracta tomando como base el árbol sintáctico proporcionado por DOM y la 'estructura del documento'. El árbol de sintaxis abstracta tendrá como nodos objetos de tipo ELEMENT. Un objeto ELEMENT simula un elemento del lenguaje XML-DB . Un objeto ELEMENT posee el valor de todos los atributos asociado al elemento del lenguaje al cual represente y la lógica necesaria para generar el código asociado a dicho elemento.

  Una función muy importante de la 'estructura del documento' es la de servir de nexo de unión entre la sintaxis concreta del lenguaje XML-DB

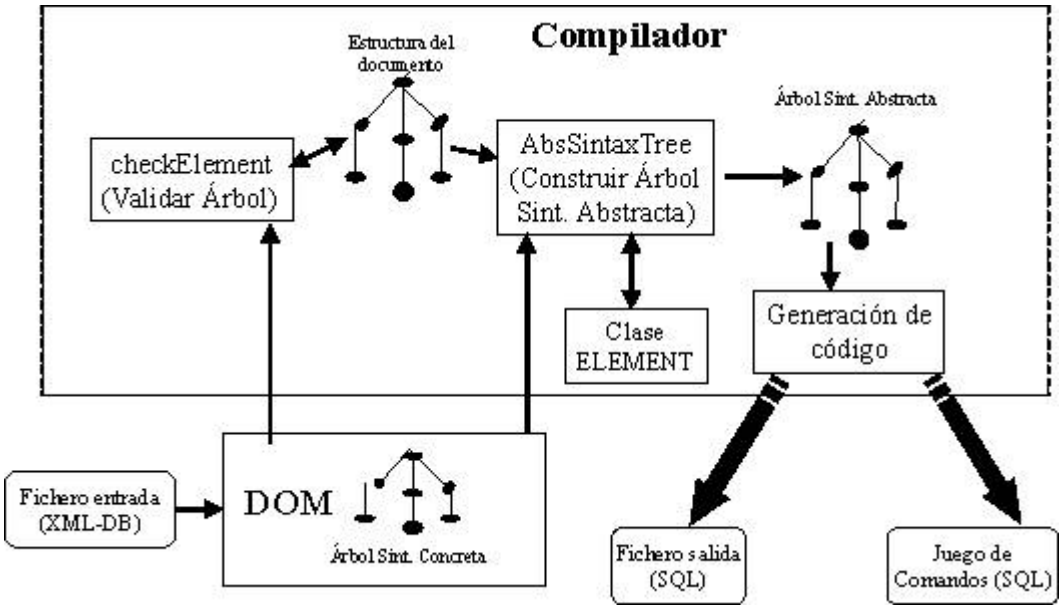

Figura 4.3: Diseño del compilador

y la sintaxis abstracta, es decir, por un lado nos encontramos os identificadores concretos de los elementos y de los atributos del lenguaje y , por otro lado, nos encontraremos los identificadores abstractos de los elementos y atributos. La 'estructura del documento' relaciona identificadores abstractos con los concretos del lenguaje.

- Finalmente, se lleva a cabo la generación de código, recorriendo cada uno de los objetos ELEMENT del árbol de sintaxis abstracta y ejecutando el método de generación de cada uno dichos objetos.

El compilador de XML-DB está implementado mediante dos módulos perl:

4.1.4.1 Módulo COMPILER

Este módulo se encuentra implementado en el fichero 'COMPILER.pm'. En el se encuentra implementada la 'estructura del documento'. Es la encargado de coordinar todo el proceso de compilación. Además se encuentran implementadas las tareas de verificación del árbol sintáctico y la construcción del árbol de sintaxis abstracta.

Vemos con más detalle cada una de las funciones que componen el módulo:

- **CompileFile:** Recibe como entrada el nombre del fichero a compilar (fichero XML-DB) y una estructura donde se almacenarán la configuración de

paginas Web asociada a cada una de las tablas de la base de datos. Es la encargada de coordinar todas las fases de compilación . Como resultado devolverá el nombre de la base de datos asociada al fichero XML-DB y el código resultante de la compilación

- **AbsSintaxisTree:** Esta función construye el árbol de sintaxis abstracta asociado a un elemento del lenguaje y funciona de forma recursiva. Para ello, toma como entrada una referencia a un elemento del árbol sintáctico de DOM y una referencia al nivel de la estructura jerárquica asociada al documento, al cuál pertenece el elemento referenciado. Como resultado devuelve la el árbol de sintaxis abstracta asociado a un elemento, por tanto, si pasamos el elemento raíz del documento, obtendremos el árbol de sintaxis abstracta de todo el documento.

- **CheckElement:** Siguiendo la misma idea de recursividad de la función anterior, en este caso lo que se lleva a cabo es la validación de los elementos del árbol sintáctico. Como entrada, se pasa la referencia a un elemento del árbol sintáctico y una referencia a la 'estructura del documento'. Como resultado, nos informará de todos los errores detectados en cuanto a incumplimiento de las restricciones del lenguaje XML-DB se refiere.

Además existe un conjunto de funciones auxiliares, que son utilizadas por las funciones anteriormente citadas:

- **GetElementObject:** Esta función crea un objeto ELEMENT de un tipo determinado, en función del elemento del documento.Este objeto se utilizará para construir el árbol de sintaxis abstracta.

- **GetElementStr:** Busca dentro de la estructura del documento un elemento del lenguaje y devuelve una referencia en ese punto sobre dicha estructura. Esta función permite recorrer la estructura del documento ir comprobando las restricciones impuestas sobre cada uno de los elementos.

- **GetChildElements:** Proporciona una lista con todos los elementos hijos del elemento pasado como parámetro.

- **CheckAttribs:** Realiza el análisis de los atributos de un elemento del lenguaje XML-DB, comprobando que no falte ninguno que sea obligatorio.

- **CheckElementStr:** Función que cheque si un elemento cumple las restricciones especificadas en la 'estructura del documento' , entre ellas,:si falta un elemento obligatorio o si aparecen más de un elemento único.

4.1.4.2 Clase ELEMENT

Aquí se encierra toda la lógica de la generación de código. (fichero ELEMENT.pm). Los nodos del árbol de sintaxis abstracta son objetos ELEMENT. Un objeto ELEMENT está constituido por un conjunto de atributos abstractos, que representan los atributos concretos del elemento del lenguaje. Además, UN objeto

ELEMENT posee un método que permite la generación de código asociado a un determinado elemento, en función del valor de sus atributos de los objetos hijos ELEMENT que tenga.

El fichero se encuentra estructurado de la siguiente forma:

- **Clase ELEMENT:** Esta es la clase principal, y que permite representar los elementos que intervienen en un documento XML-DB de forma abstracta. En esta clase se encuentran implementados los siguientes métodos, los cuales serán heredados por el resto de las subclases:

  1. ***new:*** Crea un objeto de la clase 'ELEMENT'. Como entrada, debemos suministrar una referencia del elemento del árbol sintáctico que es representado y la referencia a la 'estructura del documento' donde aparezca dicho elemento. El objeto 'ELEMENT', finalmente, será devuelto con el valor de todos los atributos actualizados.
  2. ***getChildsByName:*** Construye un array con todos los elementos hijos de uno dato, pero incluyendo sólo aquellos que tengan una etiqueta determinada.

- **Subclase ELEMENT::DataBase:** Esta clase representa a un elemento de base de datos 'DataBase', dentro del lenguaje XML-DB. Posee un único atributo: 'DataBaseName' , el nombre de la base de datos. En esta clase se implementa un método específico ' getDataBaseName' que devuelve el nombre de la base de datos.

- **Subclase ELEMENT::DataTable:** Clase que representa un un elemento de tabla de datos 'DataTable'. Posee un único atributo que indica el nombre de la tabla de datos ('DataTableName'). Un objeto 'ELEMENT::DataTable' puede tener como objetos hijo 'ELEMENT::Field', 'ELEMENT::PrimaryKey', 'ELEMENT::Unique' o 'ELEMENT::Wag'. Esta clase implementa dos métodos:

  1. ***generateCode:*** Este objeto, implica generación de código, por lo que se implementa este método 'generateCode'. En él, además de generar el código relacionado con el elemento al cual representa, se recorren todos los objetos hijos del objeto 'ELEMENT::DataTable', para obtener el código asociado a los mismos. Además de para generar código, este método permite actualizar una estructura perl que almacena los ficheros de configuración de las secciones de una aplicación Web, asociada a la tabla de datos representada; actualización que se lleva a cabo invocando el método 'getWagConfFiles' del objeto hijo 'ELEMENT::Wag', que además, será único.
  2. ***isPKeyTable:*** Esta función nos devuelve si se ha especificado clave primaria sobre una tabla en cuestión. Para ello, a la función actúa sobre el propio objeto 'ELEMENT::DataTable', el cual representa de forma abstracta la definición de la tabla de datos.

- ***Subclase ELEMENT::Index:*** Un objeto 'ELEMENT::Index' conlleva generación de código. Además, como posee hijos, necesita recorrer cada

uno de estos, para completar dicha generación de código. Los hijos sólo pueden ser objetos 'ELEMENT::IndexField'. Esta clase tiene como atributos: 'IndexName' (nombre del índice), 'TableName' (nombre de la tabla a la cual va asociada el índice), 'IsUnique' (indica tomará valores únicos) y 'AccessType' (tipo de acceso, en la implementación del índice).

- **Subclase ELEMENT::Field:** Esta clase posee los siguientes atributos: 'FieldName' (nombre del campo) , 'FieldType' (tipo de dato asociado al campo) , 'FieldValue' (valor por defecto), 'IsKey' (indica si es campo clave o no) , 'IsUnique' (indica si es campo único o no) e 'IsNotNull ' (indica si el campo puede tomar el valor nulo o si, por el contrario, está prohibido). Esta clase posee un método para la generación de código (método 'generateCode'). En este método no se recorrerán ningún objeto hijo, puesto que es un objeto 'hoja' (no tiene ningún hijo), dentro del árbol de sintaxis abstracta.

- **Subclase ELEMENT::PrimaryKey:** Clase sin atributos asociados. Un objeto perteneciente a esta clase implica generar código para él y para todos sus objetos hijos asociados. Estos objetos hijos pertenecerán a la subclase 'ELEMENT::PrimaryKeyField'. Por esta razón, la clase implementa el método 'generateCode'.

- **Subclase ELEMENT::Unique:** Clase que tampoco tiene atributos asociados. Los objetos de esta clase implican generación de código, y puesto que tienen objetos hijos (sólo pertenecientes a la clase 'ELEMENT::UniqueField'), deberán generar código para estos objetos también. Para todo ello, la clase implementa el método 'generateCode'.

- **Subclase ELEMENT::Wag:** Esta clase representa la configuración de una aplicación Web asociada a una tabla, de modo que permite establecer ficheros de configuración de páginas Web (para cada una de las secciones de que dispone la aplicación). Esta clase no posee atributos. Cada objeto de tipo 'ELEMENT::Wag' debe tener tantos objeto hijo de tipo 'ELEMENT::WagSection' como secciones se vayan a configurar para una tabla determinada. Esta clase no implica generación de código de ningún tipo. Como alternativa, tiene un método 'getWagConfFiles', que permite obtener una estructura hash anónima, que describe el fichero asociado a cada una de las secciones de una aplicación Web. además añade un entrada más donde se almacena el nombre de la tabla de datos asociada a esa aplicación Web.

- **Subclase ELEMENT::IndexField:** En esta clase, sí hay generación de código asociada al elemento. Para ello dispone de la implementación del método 'generateCode'. Posee un único atributo, que es el nombre del campo

- **Subclase ELEMENT::UniqueField:** Si implica generación de código, por lo que implementa su propio método 'generateCode'. Su único atributo es 'UniqueFieldName', que hace referencia al nombre de un objeto ELEMENT ::Field.

- **Subclase ELEMENT::PrimaryKeyField:** Esta clase redefine el método 'generateCode' y lo particulariza, teniendo en cuenta el elemento al que

representa. Posee un sólo atributo, que hace referencia al nombre de un campo de la tabla , nombre de un objeto ELEMENT::Field.

- **Subclase ELEMENT::WagSection:** Este clase no posee ningún método adicional al proporcionado por la superclase ELEMENT. Además, tampoco interviene en la generación de código. Los atributos son: 'SectionType' (tipo de sección) y 'SectionFile' (fichero asociado a la sección). Un objeto ELEMENT:WagSection permite asociar un fichero de configuración de página Web (.hdb) a una determinada sección de una aplicación Web, y todo esto, referido a una tabla de datos.

4.1.5 Gestor de datos de los usuario

El fichero USERDATAMANAGER.pm contiene toda la lógica para realizar el mantenimiento de las bases de datos definidas por los usuarios a través de los ficheros de especificación de base de datos (XML-DB). El mantenimiento de las bases de datos de los usuarios implica la creación, la eliminación de las bases de datos, así como la carga de los ficheros de especificación (XML-DB) en la base de datos. No obstante, se llevan a cabo algunas actualizaciones de la meta-información del sistema WAG (alta y bajas de bases de datos y altas de los usuarios). Veamos con más detalle las funciones de las que consta este módulo:

Constantes:

- DB_DROP_OK
- DB_DROP_ERROR
- DB_CREATE_OK
- DB_CREATE_ERROR
- DB_LOAD_OK
- DB_LOAD_ERROR

Funciones:

- createDb
- loadDb
- dropDb

4.1.6 Gestor de meta-información

Este módulo (fichero METADATA.pm) contiene un conjunto de funciones cuyo objetivo es, exclusivamente, gestionar gran parte de la meta-información del sistema WAG. Vamos a ver cual es la misión de cada una de ellas, y cómo se encuentran implementadas:

Constantes:

- WAG_DB_USER
- WAG_DB_PSSWD
- WAG_DB_NAME
- WAG_DB_META_INF_OK
- WAG_DB_META_INF_ERROR
- UPDATE_META_INF_OK
- UPDATE_META_INF_ERROR

Funciones:

- updatePageConf
- updateWagMetaInf
- updateWagTable
- updateWagField
- isPKeyField
- setUserPath
- getDbMetaInf

4.2 Implementación interface con SGBD

La interface con el sistema gestor de base de datos, que en este caso particular se trata de un servidor de base de datos PostgreSQL, está implementada en el fichero 'DB.pm' mediante dos clases: la clase DB y la clase DB::RESULT.

El objetivo de esta interface es el de aislar el resto del sistema WAG de un SGBD concreto, sirviendo, por tanto esta interface, de intermediario entre el resto del sistema y la interface proporcionada por el SGBD concreto. De este modo, cambiar el SGBD, sólo implicaría sustituir esté módulo por otro, que esté capacitado para interactuar con la interface del nuevo SGBD, sin afectar al resto del sistema.

Seguidamente vamos a ver cual es la lógica de este módulo perl:

4.2.1 La clase DB

Cada vez que necesitemos realizar una conexión con una determinada base de datos, deberemos crear una objeto 'DB'. Posteriormente, y una vez que la conexión se haya realizado con éxito, utilizaremos este objeto para ejecutar comandos SQL, obteniendo como resultado un objeto 'DB::RESULT', en el cual vendrán incluidos los datos resultantes de la consulta, si que es que los hubiera.

Constantes para definir el estado de la conexión con una base de datos o de la ejecución de un comando SQL:

- DB_CONNECT_OK
- DB_CONNECT_ERROR
- DB_EXEC_OK
- DB_EXEC_ERROR

Los atributos asociados a la clase DB son:

- dbParameters:
- dbConnection:

Los métodos de la clase DB son:

- dbExec:
- dbConnect:
- dbDisconect:

4.2.2 La clase DB::RESULT

Esta clase proporciona el resultado ante la ejecución de un comando SQL sobre una objeto de base de datos. Ahí que tener en cuenta que existen dos tipos de comandos SQL, en cuanto a la devolución de información se refiere. Los comandos SQL de tipo 'SELECT' suelen devolver un conjunto de una o más tuplas de datos, mientras que comandos del tipo 'UPDATE', 'INSERT' , 'DELETE', etc devuelven datos, tan sólo utilizaremos el objeto 'DB::RESULT' para saber si el comando se ejecutó de forma correcta, o si por el contrario hubo algún error.

Relación de atributos que pertenecen a la clase DB::RESULT:

- nRows:
- nCols:
- rowToGet:
- pgResult:
- data:

Métodos asociados a la clase DB:

- new:
- status:
- getData:
- nRows:
- getRow:
- updateData:

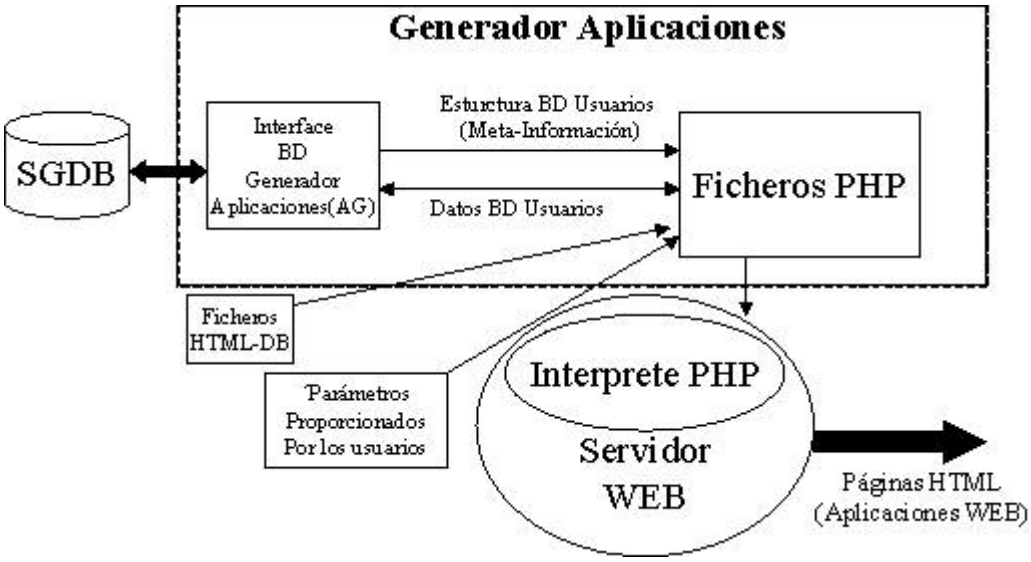

Figura 4.4: Funcionamiento de Generador de Aplicaciones (AG)

4.3 Implementación del AG

La implementación del generador de aplicaciones está basada en un conjunto de ficheros PHP, que se ejecutan en el dado del servidor de Web y que permite generar en tiempo de ejecución las páginas Web que conforman las aplicaciones Web.

Si nos fijamos en cual sería la estructura de una aplicación Web concreta, básicamente, ésta constará de formularios, encargados de la entrada y salida de datos, y de un programa CGI, que se encarga de interaccionar con la base de datos. En este tipo de aplicaciones Web, los datos de entrada salida son bien conocidos, de modo que tanto los formularios como el CGI, estarán diseñados conforme a dichos datos.

La idea que subyace en el generador de Web es el desarrollo de una aplicación Web, pero de tipo genérico, es decir, donde los datos de entrada/salida no son conocidos. De este modo, la aplicación Web será útil, sean cuales sean los datos que vayan a intervenir. Para ello, el diseño por el que se ha optado es el siguiente:

Como podemos ver, existe un módulo encargada de generar los distintas páginas Web que componen la aplicación Web, el 'Form Generator'. Estas páginas incorporan en su interior los formularios que recogen y muestran los datos que se almacenan en la base de datos. Esta capa necesita conocer cuál es la estructura de la tabla de datos, con la cual está ligada la aplicación Web. Esta información, junto con algunos otros parámetros de configuración del sistema o del usuario, es suministrada por el módulo Gestor de meta-información'

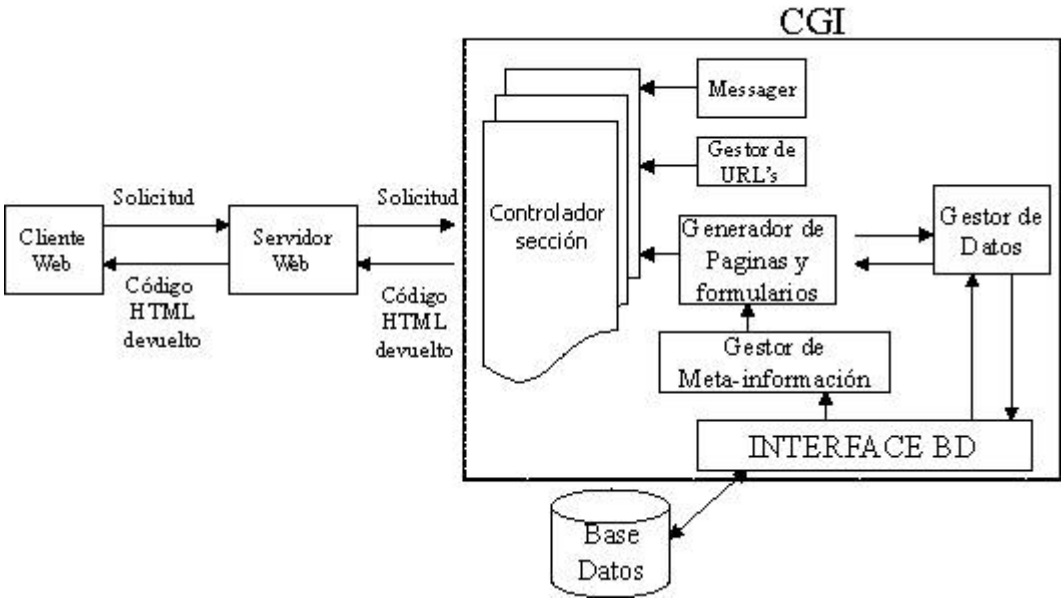

Figura 4.5: Diseño del Generador de Aplicaciones (AG)

Por otro lado, tenemos un modulo, denominado 'Gestor de datos', que actúa como el CGI típico de una aplicación, sólo que actúa de un modo mucho más genérico que el de una aplicación concreta, puesto que no se conocen , a priori, los datos que intervienen en los formularios. Este módulo es el encargado de interactuar con la base de datos concreta, para obtener y almacenar los datos, que el usuario proporciona. Al igual que el módulo anterior, éste también necesita la información proporcionada por el 'Gestor de meta-información'.

El módulo 'Meta-information Manager' es el encargado de obtener cuál es la estructura de una base de datos concreta, y mucho más concretamente, el de una tabla de una base de datos. Como sabemos , una aplicación Web está ligada a una tabla. También se acceden a otros parámetros de configuración del sistema o del usuario al cual pertenece la base de datos. La meta-información se almacena en el una base de datos, que gestiona el propio sistema. Por esta razón, esté módulo interactúa con el servidor de base de datos.

Por último, entre el 'Gestor de meta-información' y el 'Gestor de datos', nos encontramos con un módulo que actúa de interfaz entre el servidor de base de datos y el AG (generador de aplicaciones). Este módulo lo vamos a denominar 'DB-Interface', y su función fundamental es la de aislar el AG, de un servidor de base de datos concreto.

4.3.1 Estructura de una aplicación Web

Una aplicación Web genérica está estructurada en cuatro secciones: Inserción (INS), consulta (CONS), actualización (UPD) y eliminación (DEL). Cada una

estas secciones se componen a su vez de un conjunto de páginas, enlazadas entre sí, que se pueden recorrer linealmente en ambos sentidos.

- Sección INS. Permite la introducción de datos por parte del usuario. Se puede añadir registros manualmente, o bien, se puede cargar un fichero de texto (estructurado en lineas y columnas). Consta de una página:

  1. Página de entrada de datos (INS1): Esta página está dividía en dos marcos horizontales. En el marca superior, aparecerá un formulario con los campos de la tabla, para la captura de los datos, que serán posteriormente enviados a la base de datos. En el marco inferior, aparecen unos controles que nos permiten cargar directamente los datos desde un fichero de texto en la tabla.

- Sección CONS. Permite la consulta de datos almacenados en una tabla. Consta de tres páginas Web:

  1. La primera página (SCONS1) permite que el usuario indique los registros de la tabla que quiere consultar. De este modo, se mostrarán aquellos registros cuyos campos posean los valores introducidos en esta página.
  2. En esta página (SCONS2) se mostrará un listado de todas los registros que cumplen la condición de consulta. Los registros serán identificados por su clave primaria. Para cada uno de los registros, existirá un enlace junto a la clave primaria, el cual nos llevará a la siguiente página
  3. En esta página (SCONS3) se muestra el registro que se quiere consultar. Llegaremos a esta página si pinchamos en un enlace de la lista de registros, o bien , desde la página primera, si sólo existiera un único registro que cumpliera las condiciones exigidas.

- Sección UPD. Esta sección permite la actualización de los datos que hay almacenados en una tabla de datos. Esta sección posee también tres páginas:

  1. Esta página (SUPD1) es análoga a la de la sección CONS.
  2. Esta página (SUPD2) es análoga a la de la sección CONS.
  3. En está página se muestra el resultado del registro seleccionado, sólo que con la opción a modificar el valor de cualquiera de los campos.

- Sección DEL. Permite la eliminación de datos que están almacenados en una tabla. Al igual que la sección CONS y UPD, esta sección consta de tres páginas:

  1. Esta página (SDEL1) es análoga a la de la sección CONS.
  2. Esta página (SDEL2) es análoga a la de la sección CONS.
  3. Aquí (SDEL3) se muestra la información del registro solicitado, de forma análoga a lo que ocurre en la sección CONS, sólo que además tenemos la opción que nos permite eliminar dicho registro de la tabla.

La aplicación Web genérica, quedará instanciada en una aplicación Web concreta, pasándole cómo información básica: la tabla de datos a la que estará asociada, y por otro lado, la base de datos a la que pertenece dicha tabla.

4.3.2 Estructura del AG

El AG está dividido principalmente en un conjunto de ficheros 'PHP' que representan a cada una de las secciones que componen una aplicación Web. Este conjunto de fichero se encuentran dentro del directorio 'web/':

- ins.php (asociado a la sección INS)
- cons.php (asociado a la sección CONS)
- upd.php (asociado a la sección UPD)
- del.php (asociado a la sección DEL)

Cada uno de estos ficheros dará lugar a las distintas páginas que puedan formar parte de la sección a la que están asociados. Por ejemplo, el fichero 'cons.php', dará lugar a las páginas que componen la sección 'CONS', es decir, a las páginas 'SCONS1', 'SCONS2' y 'SCONS3'.

A parte de estos fichero 'PHP', tenemos otro fichero:

messager.php

Este fichero permite generar los mensajes de error y de información al usuario.

Seguidamente, pasaremos a ver cuál es la lógica que encierra cada uno de estos ficheros.

4.3.2.1 Fichero 'cons.php'

Este fichero actúa como una máquina de estados, de forma que, en función del estado en que nos encontremos, se realizará una función u otra. Cada estado, dará como resultado final una página Web.

Cada estado tendrá como parámetros de entrada: el código de identificación de la tabla de datos y el de la base de datos a la que pertenece ésta. Dando distintos valores a estas dos parámetros se obtendrán las distintas instancias posibles de la aplicación Web genérica.

Los estados en los que nos podemos encontrar son:

- Estado 'INPUT' . Este estado representa lo que será la entrada de datos del usuario, dentro de la sección de consulta (CONS). Será el estado al que debemos pasar cuando invoquemos dicha sección de la aplicación. En este estado se toman los valores de los campos, para realizar la consulta. Los parámetros de entrada a destacar en este estado son: el identificador de la base de datos y el de la tabla.

  A continuación vamos a ver los pasos que se dan en el estado 'input':

1. Se recoge el nombre del fichero 'HTML-DB' asociado a la página 'SCONS1' y la ruta asociada al usuario propietario de la base de datos que se está consultando.
2. Se recoge la URL base, donde está ubicado la herramienta WAG.
3. Se construye la URL de la página anterior ('menu.php', en este caso)
4. Se construye la URL de la página destino, que en este caso, será 'cons.php' sólo que pasando al estado 'search' (estado de búsqueda de registros).
5. Si existe un fichero 'HTML-DB', es posible generar la página 'SCONS1', que es la encarga de la entrada de datos, para generar el resultado de la consulta. Esta página, contendrá un formulario con un conjunto de elementos de entrada asociados a campos de la tabla activa en este momento. Para generar la página (en concreto el formulario), es necesario acceder a la meta-información de wag, para saber qué campos posee la tabla en cuestión y cuál es la estructura de dichos campos. Posteriormente, hacemos una llamada al procedimiento 'generate_form(parámetros), que dará lugar a la página 'SCONS1' de la aplicación Web ligada a una tabla concreta de una base de datos concreta.

- Estado 'search'. Este es el estado de búsqueda de los registros que cumplen las condiciones impuestas en el estado 'input'. Como resultado, se mostrará: o bien una lista de todos los registros que cumplen las condiciones impuestas (página SCONS2), o bien directamente el registro solicitado (página SCONS3, caso de haber un único que cumpla las condiciones). Tenemos en este estado como parámetros de entrada destacados: El identificador de la base de datos, el de la tabla y los datos introducidos por el usuario como restricciones. Los pasos que se encuentran implementados en este estado son, por tanto:

  1. Se recoge la URL base del WAG. (acceso a la meta-información del sistema)
  2. Se crea la URL de vuelta atrás. En este caso pasaremos a la página SCONS1, por lo que debemos pasar al estado de entrada de datos 'input'.
  3. Se obtiene una tabla con las claves primarias. En esta tabla se almacenan los valores que toman dichos campos, para todos aquellos registros que cumplen las condiciones especificadas en el estado de entrada de datos. Esta tarea la realiza la función 'get_key_list'. Aquí será necesario, por un lado, acceder a la meta-información del sistema para obtener cuales son los campos clave de la tabla, y por otro lado, será necesario hacer una consulta a la tabla de datos.
  4. En el caso de que sólo se devuelva un registro: Mostraremos los datos del registro solicitado (página CONS3):
     (a) Se recoge el fichero HTML-DB asociado a la página SCONS3 y el directorio de trabajo del usuario propietario de la base de datos activa.

(b) Comprobamos que el fichero exista, y en este caso recogemos los valores asociados para todos los campos del registro que cumplió las condiciones impuestas por el usuario (acceso a la tabla de datos). Finalmente, se genera la página CONS3 , que incluye un formulario para mostrar los datos del registro.

5. En el caso de que haya más de un registro:
   (a) Por cada uno de los registros que cumple las condiciones impuestas, se crea una entrada en la tabla de enlaces. Cada elemento de la tabla, por tanto,es un enlace a un registro. Un enlace a un registro, nos llevará al estado 'register', donde se genera una página de resultado para un registro concreto (página SCONS3).
   (b) Se consulta cuál es el fichero HTML-DB asociado a la página SCONS2 y el directorio de trabajo del usuario propietario de la base de datos (acceso a la meta-información).
   (c) Si el fichero existe, se genera la página con la lista de registros. Existirá un enlace para poder consultar cada uno de los registros.

- Estado 'register'. En este estado se muestra el contenido de un registro de la tabla, solicitado por el usuario. Por tanto, en este estado, lo que se generará finalmente será una página Web con un formulario que incluya los campos del registro y sus valores asociados a estos. Aquí, como parámetros de entrada importantes, tenemos: el identificador de la base de datos, el de la tabla, y los valores para los campos que actúan como clave primaria de la tabla. Veamos, cómo está implementado este estado:

  1. Se recoge el fichero HTML-DB asociado a la página SCONS3 y el directorio de trabajo, en la que se encuentra dicho fichero. (acceso a la meta-información)
  2. Se toma la URL base del WAG (acceso a la meta-información)
  3. Se crea la dirección de vuelta atrás. En este caso, pasaremos al estado anterior, es decir, el estado 'search'.
  4. Si existe el fichero, se accede a los datos del registro, mediante una consulta sobre la tabla de datos y, seguidamente, se genera la página Web que muestra los datos asociados al registro solicitado por el usuario.

4.3.2.2 Fichero 'upd.php'

Este fichero, al igual que el fichero 'cons.php', también se basa en una máquina de estados que generan las distintas páginas que forman parte de la sección UPD. El acceso a este fichero, debe de estar protegido a todos los usuarios, excepto al propietario de la base de datos en cuestión. Los estados existentes en esta máquina de estados son: INPUT, SEARCH, REGISTER y UPDATE.

Los tres estados primero, tienen la misma lógica que la explicada para estos estados dentro del fichero 'cons.php', sólo que teniendo en cuenta, que aquí, lo

que se buscarán son los ficheros HTML-DB relacionados con la sección UPD. Al igual que en el caso de la sección de consulta, estos estado implementan la entrada de datos, generación de la lista de registros y generación del registro resultado, respectivamente.

No obstante, aparece un nuevo estado 'update', que permite la actualización del registro seleccionado, tras ser éste modificado por el usuario. Veamos cuál es la lógica que encierra este estado:

1. Se recoge la URL base del WAG (acceso a la meta-información)
2. Se crea la URL para acceder a la página de actualización (SUPD3) , que posteriormente será utilizada para generar la página Web destino. Por esta razón, debemos pasar al estado 'register' de la máquina de estados.
3. Se construye la URL para generar los mensajes resultantes de las acciones realizadas sobre la tabla. Estos mensajes indicarán al usuario si las transacciones se han realizado correctamente o si por el contrario hubo algún problema.
4. Se lleva a cabo la actualización del registro, mediante la función 'update_register'
5. Finalmente, se genera una página que muestra el resultado tras la actualización. Esta página estará formada por dos marcos. Un marco ,contendrá una página de actualización de registro (SUPD3), para permitir al usuario realizar una nueva modificación sobre este registro. En el otro marco se mostrará un mensaje que indica al usuario cuál ha sido el resultado de la modificación del registro.

4.3.2.3 Fichero 'del.php'

Este fichero también implementa una máquina de estados. En este caso, lo que se genera es las páginas asociadas a la sección DEL. Los estados que contiene esta máquina de estados son: INPUT, SEARCH, REGISTER, DELETE y DELETEALL.

Del mismo modo que en los dos fichero anteriores, la lógica de los tres estados primeros es la misma, teniendo en cuenta, que en esta ocasión se buscarán ficheros HTML-DB asociados a la sección de eliminación (DEL).

Los otros dos estados, permiten la eliminación del registro seleccionado y la eliminación de todos los registros pertenecientes a la lista, respectivamente. Veamos cómo están implementados ambos estados:

- Estado 'delete': En este estado se lleva a cabo la eliminación del registro seleccionado. Como resultado se generará una página página dividida en dos marcos. Un marco (el marco 'DATA'), que contendrá la lista de registros resultante de la selección, o bien, el registro registro resultante, en el caso de que quede sólo un registro tras la eliminación. Otro marco (el marco 'MESSEGER') que mostrará un comentario para que el usuario

sepa cuál ha sido el resultado de la eliminación, es decir, si esta se realizó satisfactoriamente o no.

Los pasos que se dan en este estado son:

1. Se recoge la URL base del WAG.
2. Se construye la URL para el marco 'DATA' (en este caso, pasamos al estado 'search' de la máquina de estados).
3. Se lleva a cabo la eliminación del registro seleccionado. Para ello se hace uso de la función 'delete_register'.
4. Generamos la página que contendrá los dos marcos mencionados anteriormente:uno con los registros (o registro resultante) y otro con el mensaje informativo para el usuario.

- Estado 'deleteall': Dentro de este estado se realiza la eliminación de todos los registros seleccionados por el usuario. Esto permite al usuario la eliminación, en una sola operación, de un conjunto de registro, evitando así tener que eliminarlos de uno en uno. Como resultado, obtendremos un página dividida en dos marcos: uno, que contiene la página 'SDEL1' (marco 'DATA') y otro ,que contiene el mensaje para el usuario (marco 'MESSEGER'). La lógica encerrada en este estado es la siguiente:

  1. Se recoge la URL base del WAG.
  2. Se construye la URL para el marco 'DATA' (En esta ocasión, se requiere pasar al estado 'input', para volver a la página de entrada de datos de la sección DEL).
  3. Se realiza la eliminación de todos los registros seleccionados. Esta tarea se realiza mediante la función 'delete_all_registers'.
  4. Se genera la página formada por los marcos 'DATA' y 'MESSEGER', previamente comentados.

4.3.2.4 Fichero 'ins_upload.php'

Este fichero es el que da lugar a la página que permite la entrada de datos en la tabla. Esta página está dividida en dos marcos. Un marco (INS), que permitirá insertar insertar un registro dentro de la tabla, dando valores a los campos. Por tanto, será este marco el que contenga la página SINS1. Por otro lado, en otro marco (marco UPLOAD), se permite la carga de un conjunto de registros desde un fichero de texto.

4.3.2.5 Fichero 'ins.php'

El fichero 'ins.php', asociado a la sección de inserción de datos en una tabla (INS), implementa una máquina de estados. En este caso, los estados que intervienen son: INPUT e INSERT. Por un lado, se permite al usuario suministrar los datos asociados a un un registro, mediante una página alojada en el marco 'INS' dentro de la página 'ins_upload.php', por otro lado, se realiza el proceso de inserción de los datos del registro en la tabla.

La implementación de los estados es la siguiente:

- Estado 'input'. En este estado se genera el marco de entrada de datos del registro. Los pasos de los que consta este estado son:

  1. Cogemos el fichero HTML-DB asociado a la página SINS1 y el directorio de trabajo del propietario del fichero.
  2. Creamos la URL para la vuelta a la página anterior, que en este caso es el menú; por esta razón debemos ir al fichero 'menu.php'.
  3. Se construye la URL destino de la página, que será la el fichero 'ins.php', pero pasando al estado 'insert', puesto que debemos de insertar el registro y emitir el mensaje de resultado correspondiente.
  4. Comprobamos que exista el fichero HTML-DB, y si es así, se accede a la meta-información para saber cuales son los campos de los que constan los registros de la tabla. Posteriormente, se genera la página Web, que incluirá un formulario en el que aparecerán los campos de la tabla.

- Estado 'insert'.Aquí es donde tiene lugar la inserción del registro dentro de la tabla de datos. Veamos cuáles son los pasos para llevar a cabo esta tarea:

  1. Se coge la URL base del WAG (acceso a la meta-información)
  2. Creamos la dirección del marco destino DATA, que en este caso es la página de entrada de datos SINS, dividida en los dos marcos citados anteriormente (fichero 'ins_upload.php')
  3. Se realiza la inserción de los datos del registro en la tabla.
  4. Se crea la dirección del marco MESSAGE, para mostrar al usuario el resultado de la inserción del registro en la tabla.
  5. Finalmente, se genera la página destino, formada por los marcos DATA y MESSEGE.

4.3.2.6 Fichero 'upload.php'

Este fichero permite la inserción en una tabla de un conjunto de datos a partir de un fichero de texto, que será proporcionado por el usuario. También en este caso, se ha utilizado una máquina de estados para su implementación. Esta máquina consta de dos estados, que a continuación detallamos:

- Estado 'insert': Permite extraer el conjunto de registros almacenados en el fichero de texto proporcionado por el usuario e insertarlos en la tabla. El algoritmo de implementación del estado es:

  1. Obtenemos la URL de la herramienta WAG.
  2. Se construye la URL para el del marco DATA. En este marco aparecerá la página página de entrada de datos SINS1.

3. Extraemos el conjunto de registros desde el fichero proporcionado por el usuario. Además se extraen los nombre de las columnas en las que está estructurado el fichero, para que se puede llevar a cabo la correspondencia entre los nombres y el campo al que debe de ir asociado.
4. Se insertan cada uno de los registros extraídos del fichero de texto. Esta tarea se realiza gracias al procedimiento 'get_register_from_file'.
5. Construimos la URL para generar el mensaje informativo para el usuario. Este mensaje se insertará en el marco MESSEGE.
6. Generamos la página compuesta por dos marcos (marco DATA y MESSEGE), cuyas URL se han construido previamente.

El procedimiento 'get_register_from_file', se encuentra implementada dentro de este mismo fichero, como una función auxiliar. Este procedimiento tiene como parámetros de entrada el nombre del fichero original, el nombre del archivo que se almacena en el servidor y el indicador sobre la inclusión de nombre de columnas en la primera línea del fichero. Como parámetros de salida tenemos dos tablas de datos: la tabla de nombres de columnas y la tabla de registros. Su implementación queda como sigue:

1. Abrimos el fichero proporcionado por el usuario, comprobando previamente que no hay ningún problema, como que no se haya indicado ningún fichero, o que no pueda abrirse, por cualquier motivo.
2. Comprobamos si incluye o no nombres de columna en la primera línea del fichero. En caso que se incluyan los nombres de las columnas, estas serán extraídas del fichero y almacenadas en la tabla ($columns_name).
3. Se extraen los registros del fichero texto. Un fichero por cada línea. Para ello, se utiliza la función de PHP 'fgetcsv'. Esta función de PHP permite extrae un conjunto de datos desde una línea de un fichero de texto, siempre que se indique cuál es el separador de dichos datos dentro de la línea (NOTA: se ha optado por el separador punto y coma ';'). El conjunto de registros se devuelve dentro de la tabla de registros ($registers).

- Estado 'input': En este estado, se genera la página donde el usuario puede proporcionar al sistema el fichero que actuará como fuente de datos. En dicho fichero se encuentran los registros que se quieren cargar en la tabla. La lógica del estado es la siguiente:

  1. Obtenemos la URL base del WAG.
  2. Creamos la URL para pasar al siguiente estado 'insert', donde se insertarán los registros incluidos en el fichero facilitado por el usuario.
  3. Finalmente, se genera el marco que formará parte de la página SINS1 de la aplicación Web. Dentro de este marco se podrá especificar el fichero que contiene los registros, y además podemos indicar si se incluyen los nombres de las columnas o no.

4.3.2.7 Fichero 'messager.php'

Este fichero tiene como objetivo mostrar un mensaje al usuario. Este mensaje estará alojado dentro de un frame, de modo que será siempre invocado desde un elemento FRAME. Como parámetro externo recibirá el mensaje que debe de ser mostrado.

4.3.3 Generación de páginas Web y formularios

Este módulo es el encargado de generar las páginas Web, así como todos los elementos que la componen. La mayoría de las páginas incluirán un formulario donde el usuario podrá insertar u obtener información. Desde un punto de vista abstracta, este módulo actuará como haría un formularios dentro de una aplicación Web, es decir, recogiendo los datos, para luego ser procesados por un CGI. Para la generación de las páginas y de sus elementos se necesita, por un lado un fichero HTML-DB de partida, y por otro lado, acceder a la meta-información del sistema.

Vamos ahora a detallar detenidamente cuáles son las funciones que componen este módulo y cómo se encuentran implementadas. Este módulo se encuentra implementado como un fichero para php (form.php) ubicado dentro del subdirectorio 'web/lib/'.

- Procedimiento 'html_wag_parser'

  Este procedimiento actúa como parser de ficheros HTML-DB, extrayendo los comandos HTML-DB existentes en el fichero que se pasa como entrada, a la vez que se extraen los bloques de código HTML 'puro', que éste pueda contener.

- Procedimiento 'generate_list'

  Genera una página Web que contiene una lista con las claves primarias de todos los registros de determinada tabla de datos, los cuales cumplen una condición. Cómo parámetros de entrada tenemos el fichero HTML-DB, que se utiliza como base, una tabla con las claves primarias de los registros a mostrar en la lista, una lista con todos los enlaces URL hacia cada uno de los registros y un conjunto de URLs auxiliares (para volver a la página anterior, etc.)

- Procedimiento 'generate_register

  Este procedimiento lleva a cabo la generación automática de un formulario dentro de la página Web. Este formulario permite tener acceso a un registro de la tabla de datos activa en ese momento. De modo que este formulario contendrá tantos elementos de entrada/salida de datos (elementos INPUT) como columnas tenga la tabla. Este formulario permitirá la visualización de los datos de un registro y/o la modificación de estos, según la sección a la que pertenezca la página Web, en la que va insertado el registro.

- Procedimiento 'show_list'

  Permite la generación de la lista de claves primarias de los registros seleccionados, asociándole un enlace a cada uno de los registros para permitir el acceso a los datos dicho registro. La lista se mostrará cómo una tabla con una fila por cada registro existente. La tabla, tendrá varias columnas: una columna contendrá los enlaces a los registros, y el resto, contendrá el valor para los distintos campos clave de la tabla de datos en cuestión. Este procedimiento será activado cuando encontramos un elemento HTML-DB que los indique.

- Procedimiento 'show_upload'

  Este procedimiento muestra la pagina Web alojada dentro del marco 'UPLOAD' perteneciente a la página SINS1. La página, aquí generada, permite al usuario indicar cuál es el fichero de texto que contiene los datos a cargar en la tabla de datos. Además se permitirá indicar si se incluyen los nombres de los campos de la tabla en la primera línea del fichero o no.

- Procedimiento 'show_field_textarea'

  Este procedimiento inserta un elemento TEXTAREA, asociándolo a una columna determinada de la tabla de datos que esté activa en ese momento. De esta forma, este elemento podrá mostrar y/o recoger un valor para el campo de un registro. El procedimiento tiene como parámetros de entrada: el nombre del campo, el valor asociado (si lo hay), y un conjunto de valores ligados a atributos del elemento TEXTAREA.

- Procedimiento 'show_field_input'

  Inserta un elemento de tipo INPUT dentro de la página Web, asociándolo a una determinada columna de la tabla de datos. Se procede, por tanto, como en el caso del elemento TEXTAREA. No obstante, en este caso, es necesario determinar qué tipo de elemento INPUT se trata (TEXT, HIDDEN, RESET, SUBMIT, RATIO, etc). Como parámetros de entrada, en este caso nos encontramos: el tipo de elemento INPUT, el nombre del campo, el valor asociado(si lo hay) y un conjunto de valores asociados a los atributos del elemento INPUT.

- Procedimiento 'show_back_link'

  Muestra un enlace de vuelta a la página anterior. Se pasa la URL de la página anterior como parámetro de entrada.

- Procedimiento 'show_title'

  Inserta un título dentro de pagina que indica dentro de qué sección nos encontramos. Como parámetro de entrada, tomará el nombre de la página en la que nos encontramos, cuando es invocado el procedimiento. En función de la sección a la que pertenezca la página, se generará un título u otro. Otros parámetros de entrada son el color de fondo del título, así como el color del texto que conforma el título.

- Procedimiento 'generate_form'

  Este procedimiento es el encargado de generar un formulario dentro de una pagina Web. Como partida se toma un fichero formato HTML-DB,

en el cual se encuentra el contenido de la página Web, así con una serie de comandos HTML-DB que indican dónde debe de ir colocado el formulario, así como cuáles y dónde deben de aparecer los campos de la tabla de datos a la que dicho formulario irá ligado. Por tanto, la tarea fundamental que se lleva a cabo es la extracción de los comandos HTML-DB y la interpretación de estos. Los parámetros de entrada son: el nombre del fichero HTML-DB, los campos que forman parte de la tabla de datos, el identificador de la tabla de datos y de la base de datos a la cual pertenece, un conjunto de URLs (destino del formulario, vuelta a la pagina anterior, etc.). Por último, también se pasa como parámetro de entrada el nombre de la página en la que el formulario será insertado.

- Función 'get_wag_command_parameters'

  Esta función permite extraer el valor de los parámetros asociados a un comando HTML-DB. Como entrada tenemos el comando HTML-DB, el cual será analizado. Como resultado obtendremos una tabla con todos los valores asociados a los parámetros que dicho posea. Esta función devuelve el tipo de comando HTML-DB pasado como parámetro.

- Procedimiento 'exec_wag_command'

  Lleva a cabo la ejecución de un comando HTML-DB, es decir, realiza las operaciones asociadas a la semántica de un comando. Para ello, el procedimiento debe conocer de qué tipo de comando se trata, el valor de los parámetros asociados a éste, así como los campos asociados a la tabla de datos.

- Procedimiento 'exec_start'

  Realiza las tareas relacionadas con la inicialización de un formularios. Como parámetro de entrada necesita la URL que se colocará en el atributo 'acction' del formulario. Esta URL será el destino donde se procesarán todos los datos provenientes del formulario. Este procedimiento será invocado siempre justo antes que cualquier comando HTML-DB tales como WAG-INPUT o WAG-TEXTAREA.

- Procedimiento 'exec_end'

  Aquí se lleva a cabo la finalización de un formulario. Será ejecutado justo después de todos los comandos HTML-DB de tipo WAG-INPUT y WAG-TEXTAREA existentes en el fichero.

Funciones auxiliares:

- Función 'exists_field'

  Esta función devolverá un valor lógico (cierto) cuando el campo al que hace referencia un elemento WAG-INPUT o WAG-TEXTAREA, existe verdaderamente en la tabla de datos. En caso contrario, se devolverá el valor lógico (falso)

- Procedimiento 'set_base_url'

  Permite establecer la URL que actuará como base en la página Web. Esto permitirá que la página se construya con URL relativas al directorio de

trabajo del usuario, de forma totalmente transparente. Como parámetros de entrada nos encontramos con el fichero HTML-DB y con la URL que actuará como base del documento. Esta URL estará formada por la URL donde se encuentra el sistema WAG, concatenado con la ruta relativa a dicha URL, donde se encuentra el directorio de trabajo del usuario propietario de la tabla de datos activa en ese momento.

- Procedimiento 'print_error'

  Este procedimiento genera un mensaje de error, avisando al usuario cualquier tipo de incidencia.

4.3.4 Gestor de meta-información

Este módulo encierra toda la lógica que permite el acceso a la meta-información del sistema WAG. La meta-información es necesaria para el resto de módulos, esencialmente para el 'Gestor de datos' y para el 'Generador de Web y de Formularios'. El Gestor de meta-información, por tanto, proporciona unos servicios al resto del AG, que hacen transparente el modo en que la meta-información se encuentra almacenada dentro del WAG, así como la implementación del acceso a dicha información.

Los servicios que se encuentran implementados en este módulo son :

- Función 'get_field_meta_inf'

  Esta función nos ofrece todos los campos que componen una tabla de datos determinada. Además nos facilita el tipo asociado a cada campo, el tamaño (para la interfaz), y la longitud máxima en número de caracteres. Toda esta información, se obtiene mediante un acceso a la meta-información del sistema.

- Función 'get_key_field_name'

  Permite obtener una tabla con todos los nombres de los campos que actúan como clave primaria en la tabla de datos, cuyo identificador se pasa como parámetro de entrada. Para obtener esta información, también tenemos que realizar un acceso a la meta-información del sistema WAG.

- Procedimiento 'get_db_names'

  Realiza un acceso a la meta-información del sistema para recoger el nombre de todas las bases de datos existentes en el WAG, es decir, que han sido definidas por el usuario. El conjunto de nombres de todas las bases de datos se inserta dentro de un array que se pasa como parámetro de entrada/salida.

- Función 'get_table_names'

  Aquí también se realiza un acceso a la meta-información del sistemas, aunque en este caso para obtener cuales son todos los nombres de las tablas de datos asociadas a una base de datos dada, la cuál se pasa como parámetro de entrada.

- Función 'get_base_url'

  Recoge la URL base del sistema WAG, mediante consulta sobre la meta-información. Esta URL será establecida por el manager del WAG e indica donde se encuentra situado el AG, dentro del servidor de Web.

- Función 'get_user_page_and_path'

  Mediante el uso de esta función podemos obtener cual será el fichero HTML-DB asociado a una página de una aplicación Web concreta. Para ello, se debe conocer a qué tabla de datos está asociada la aplicación Web, y de que página dentro de la aplicación se trata. Además , también será devuelta la ruta relativa, respecto del directorio 'web/', donde se encuentra dicho fichero. Este fichero puede ser uno definido por el usuario, o bien, un fichero tomado por defecto y definido por el manager del WAG.

- Función 'get_db_parameters'

  Esta función ofrece los parámetros necesarios para conectar con una base de datos definida por un usuario. Como parámetros de entrada, se pasan el identificador de la tabla a la que está asociada la aplicación, y el de la base de datos a la cual pertenece dicha tabla. Estos parámetros serán muy utilizados por el módulo 'Gestor de datos' , debido a que en él se realizan accesos a los datos almacenados en las bases de datos definidas por los usuarios. Los parámetros devueltos son: el nombre del usuario, el password de dicho usuario, el nombre de la tabla y el nombre de la base de datos. Si no se obtiene ningún parámetro, se devuelve el valor lógico 'falso' (cero) o en caso contrario el valor lógico 'cierto' (uno).

Ahora vamos a ver algunas de las funciones auxiliares de este módulo. Algunas de ellas son utilizadas por el módulo Gestor de datos'.

- Función 'add_field_in_where'
- Función 'add_field_in_select'
- Función 'select_not_empty_fields'
- Función 'get_max_length'

4.3.5 Gestor de datos

En cualquier estructura típica de una aplicación Web, siempre existe un CGI encargado de procesar toda la información proporcionada por el formulario insertado en la página Web. Este módulo es el encargado de hacer las funciones de procesado de los datos que provienen de los formularios generados por el módulo 'Generador de páginas Web y Formularios'. Este módulo, por tanto, implementa servicios para llevar a cabo la inserción de los datos, modificación y eliminación de los datos procedentes de los formularios en las tablas de datos . También implementa la posibilidad de obtener información desde las tablas de datos, y que esta sea presentada mediante un formulario. Todas estas funciones realizan una conexión con el servidor de base de datos, para

acceder a la base de datos a la cual pertenece la tabla de datos sobre la que se realizarán los accesos.

Los procedimientos y funciones, mediante los cuales se proporcionan los servicios anteriormente citados, son los siguientes:

- Función 'insert_all_registers'

  Permite la inserción de un conjunto de registros dentro de una tabla de datos dada. A esta función se le pasa como parámetro de entrada el código de la base de datos a la que pertenece la tabla de datos y el código de la tabla. Además se le proporciona una lista con el nombre de todos los campos de la tabla para los cuales se van a proporcionar valores y otra tabla con el conjunto de registros que se quieren insertar en la tabla de datos. La función nos dará como resultado un valor lógico 'cierto' si se produce la inserción correctamente, o bien, un valor lógico 'falso' en caso contrario. Para la inserción de los registros.

- Función 'insert_register'

  Esta función realiza la inserción de un registro dentro de la tabla de datos que se pasa como parámetro de entrada. Como parámetro de entrada, también se pasará una tabla de campos compuesta por el nombre de campos de dicha tabla, junto con el valor asociado a dichos campos. La función devolverá un valor lógico indicando si la inserción del registro se realizó o no con éxito.

- Función 'get_register'

  La función 'get_register' es una función que devuelve un registro de una tabla de datos, cuya clave primaria se pasa como parámetro de entrada. El registro se obtendrá en una tabla de campos, formada por el nombre de los campos de la tabla y el valor del registro que posee en cada uno de estos campos. Si no existe ningún registro que posea como clave primaria la pasada como parámetro, o bien hubiese algún problema en el acceso a la base de datos, se devolverá un array vacío (nulo).

- Función 'get_key_list'

  Esta función nos devolverá un array con un conjunto de claves primarias de todos aquellos registros de una tabla de datos que cumplen una cierta condición. En este caso, la condición que deben de cumplir un registro para que su clave primaria sea devuelta dentro del array, será que todos los campos del registro tengan como valor asociado el mismo que se pasa como parámetro de entrada en la función. Como parámetro de entrada, por tanto, se pasa el identificador de la base de datos y de la tabla de datos, la restricción impuesta a los registros (par campo-valor), y como parámetro de entrada/salida, el array con todas las claves primarias de los registros que cumplen la restricción impuesta. La función nos devolverá un valor lógico que nos indicará si todo ha ido bien o no.

- Función 'update_register'

  Con esta función se puede modificar el valor de los campos de un registro de terminado, identificado por su clave primaria. De este modo tendremos cómo parámetros de entrada un conjunto de pares campo-valor,

la clave primaria del registro que se desea modificar, y como siempre, el identificador de la base de datos y de la tabla de datos. La función nos devolverá el valor lógico que nos indicará si la operación de modificación se realizó de forma correcta, o por el contrario hubo algún problema.

- Función 'delete_register'

  Realiza la eliminación de un registro de la tabla de datos, cuya clave primaria se pasa como parámetro de la función. Otros parámetros de entrada de la función serán los identificadores de la base de datos y de la tabla de la cual queremos eliminar el registro. Esta función, también nos devolverá un valor lógico indicándonos si la eliminación se realizó de forma correcta o no.

- Función 'delete_all_registers'

  Mediante el uso de esta función se puede realizar la eliminación de todos los registros de una tabla que cumplan una condición. La condición impuesta se pasa como parámetro de entrada, como un array de pares campo-valor. En este caso, la restricción que deberán cumplir los registros que serán eliminados será poseer como valor de sus campos los indicados en los pares campo-valor de la condición. La función nos indicará si la eliminación se realizó con éxito o si ocurrió algún error.

Función auxiliar:

- Función 'add_fields_in_set'

  Permite construir un comando SQL (UPDATE) para la modificación de los registros de una tabla de datos. Toma como entrada el conjunto de campos cuyos valores se pretenden modificar. Toma la cadena de que representa el comando SQL, y le añade la clausula SET junto con el nombre de todos los campos que se van a modificar separados por comas.

4.3.6 Gestión de URL's

El papel de este módulo dentro del AG consiste en proporcionar un mecanismo para crear URLs. Esta necesidad surge porque la gran mayoría de estas URLs, llevan asociadas una serie de parámetros que permiten, por una parte, el correcto funcionamiento de las distintas máquinas de estados implementan el AG, y por otra parte, el intercambio de información entre los distintos estados en los que se encuentran las máquinas de estados. Otra función importante de este módulo es la de hacer los más transparente posible, la posibilidad de que se produzca un cambio en la URL donde se esté ejecutando la herramienta WAG.

Seguidamente veremos cuales son los servicios proporcionados por este módulo:

- Función 'create_url'

Nos va a permitir la creación de una URL interna del AG, con la posibilidad de asociarle un parámetro y el valor asociado a éste. A la función se le proporcionará el nombre del fichero, junto con el identificador del parámetro y el valor asociado a este. También tendremos que proporcionarle la URL base, donde se está situado el WAG.

- Procedimiento 'add_parameter_to_url'

  Añade un parámetro y el valor asociado a él a una URL dada. La URL será pasada como parámetro de entrada/salida del procedimiento, el parámetro de la URL y su valor asociado será pasado como entrada.

Capítulo 5

Uso del sistema

En este capítulo vamos a tratar todos los aspectos relacionados con el manejo del WAG, tanto desde el punto de vista del administrador, como del usuario final. Dado que el WAG es un servicio proporcionado por el servidor, es necesario que exista un administrador que realice la instalación y el mantenimiento del sistema. Pero, por supuesto, también es fundamental que los usuarios, conozcan las posibilidades que WAG les brinda y sean capaces de adaptarlos a sus necesidades.

5.1 Diferentes tipos de usuarios

5.1.1 El administrador

El administrador es un usuario especial del sistema WAG, cuyo papel más importante es el de llevar a cabo el mantenimiento del sistema, para su correcto funcionamiento. Para el mantenimiento del sistema WAG, el administrador debe hacer uso de la herramienta 'wag-manager', la cual está diseñada para hacer esta tarea lo más sencilla y cómoda posible. El mantenimiento del sistema implica tareas como la gestión los usuarios, administrar las bases de datos definidas por los usuarios, diseñar y definir la interfaz que van a tener por defecto las aplicaciones Web.

5.1.2 El usuario

Los usuarios de WAG son aquellas personas que estén habilitadas para hacer uso del sistema. El servicio que proporciona WAG a un usuario es el poder hacer públicas sus bases de datos a través en Internet. De este modo, los datos que el usuario tenga almacenados en sus bases de datos, podrán ser consultados por cualquier internauta que lo desee.

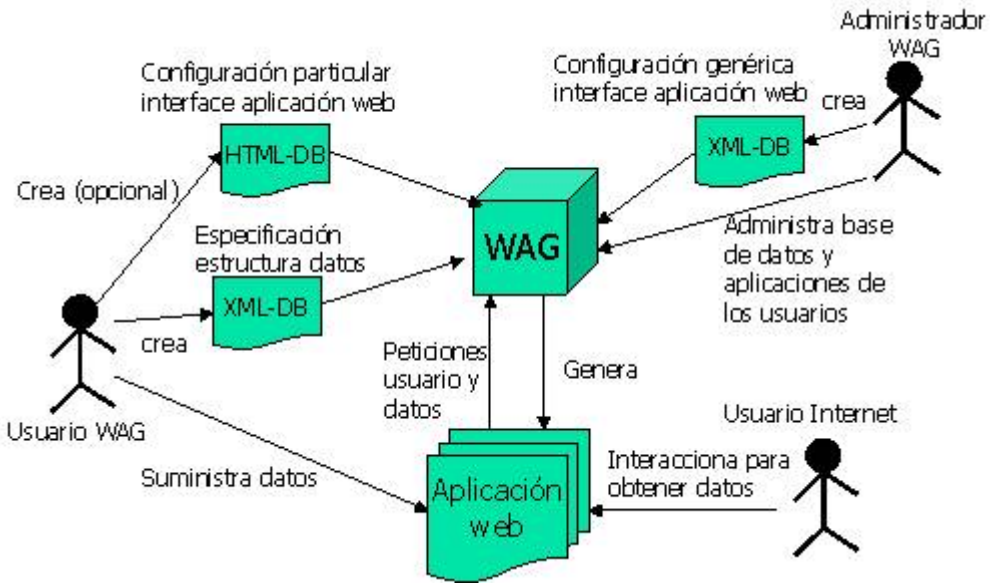

Figura 5.1: Utilización sistema WAG

Un usuario que quiera, por tanto, hacer pública una base de datos a través del servicio WAG, sólo tendrá que hacer una definición de la estructura de dicha base de datos, mediante un lenguaje muy sencillo y flexible que denominaremos XML-WAG. A partir de esta definición, se creará una base de datos en el lado del servidor, de modo que mediante una aplicación Web, el usuario podrá actualizarla de forma permanentemente, introduciendo datos, modificándolos o eliminándolos. Todas estas tareas de mantenimiento de los datos de la base de datos, las puede hacer un usuario cómodamente con su navegador desde su terminal de Internet.

5.2 Manual del administrador

El contenido de esta sección se va a centrar en conocer cuáles son las tareas que debe desempeñar el administrador del WAG y cuáles son los mecanismos existentes para llevarlas a cabo.

Por un lado, el administrador del sistema será el encargado de instalar el WAG sobre el servidor de Web y de ponerlo en marcha. El administrador, por tanto, debe de estar familiarizado con servidores Web, tales como Apache. Además , debe de ser capaz de configurar un servidor de base de datos PostgreSQL. La aplicación WAG se apoya tanto en un servidor de bases de datos (en este caso, PostgreSQL), como en un servidor Web (Apache, en nuestro caso). Para facilitar la instalación, WAG se puede obtener como un paquete (RPM). También se puede obtener en un fichero comprimido (GZIP o BZIP). Será, sin duda, , la tarea de coordinar WAG con los servidores Web y de bases de datos, lo más complicado en la fase de instalación del sistema WAG.

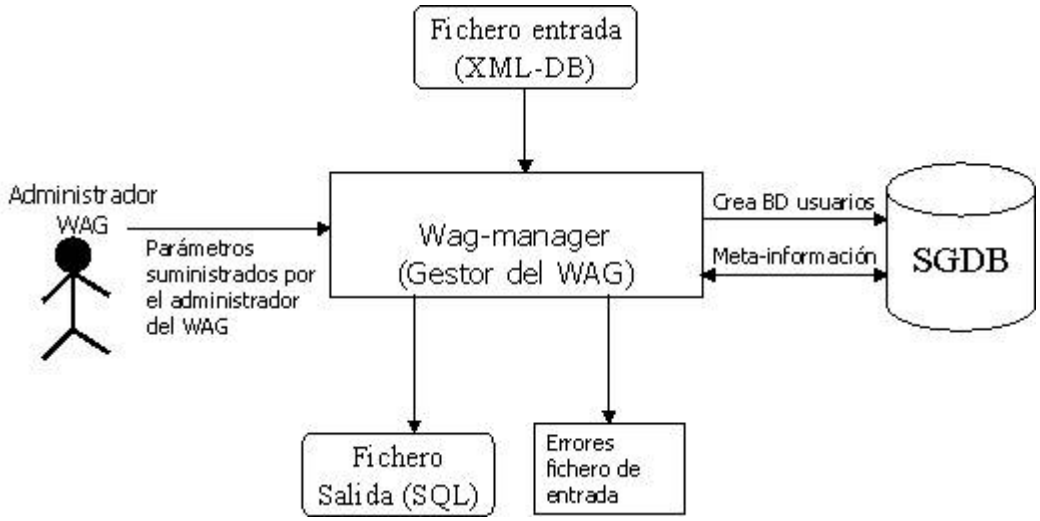

Figura 5.2: Utilización del gestor del WAG

Por otro lado, el administrador también es el encargado de realizar el mantenimiento del sistema. El mantenimiento del sistema implica: la gestión de los usuarios que de él harán uso (altas, bajas), asignación del directorio de trabajo a los usuario, diseño y configuración de la interfaz (por defecto) de las aplicaciones Web y del control de los scripts SQL de las bases de datos especificadas por los usuarios. Para facilitar el mantenimiento del WAG, se proporciona la herramienta 'wag-manager', mientras que para el diseño de la interfaz gráfica (por defecto), se utilizarán páginas HTML-DB (HTML enriquecido con elementos que indican dónde deben ir los datos de las bases de datos).

5.2.1 Herramienta wag-manager

Esta herramienta es la que va a permitir el mantenimiento del WAG al administrador. Por eso, esta herramienta únicamente será utilizada por él.

La herramienta 'wag-manager' es un script de Perl, por lo que será necesario que se encuentre instalado un interprete de Perl en el sistema, para que pueda funcionar correctamente. Además, esta herramienta interacciona con el servidor de base de datos PostgreSQL, por lo que será imprescindible, que dicho servidor esté configurado adecuadamente y se encuentre en ejecución.

La sintaxis de la herramienta 'wag-manager' es la que sigue:

```
wag-manager <input_file_name> [-o [-
f <output_file>] [-a] ]
            [-e] [-c <db_name> -u <user_name>]
            [--host=<ip_address>][--
port=<port_number>]
```

```
wag-manager -d <db_name>[--host=<ip_address>]
            [--port=<port_number>]
wag-manager -u <user_name>[-p <work_directory>]
            [--host:<ip_address>][--
port:<port_number>]
xml-manager -h | --help
```

- ***<input_name_file>***

  Nombre del fichero que queremos parsear. Normalmente, este fichero tendrá extensión 'xdb' .

- ***-o | –out***

  Esta opción le dice al parser que genere como salida un script 'sql'. Este fichero contiene el conjunto de sentencias 'sql' que definen base de datos especificada en el fichero que se pasa como entrada. Si no se especifica el nombre del fichero de salida, se generará un fichero denominado 'out.sql'.

- ***-a | –add***

  Con esta opción se puede añadir el resultado a un fichero, ya existente, indicado mediante la opción '-f'. Debemos usar esta opción combinada con el comando '-o'.Si el fichero no existiera entonces se crearía uno nuevo.

- ***-c | –create=<db_name>***

  Esta opción le indica al parser que cree la estructura de la base de datos físicamente en la servidor base de datos PostgreSQL. El nombre de la base de datos , debemos suministrarlo al utilizar esta opción . El uso de esta opción, implica el uso de la opción '-u', que se describe más abajo. Nota: Esta opción implica conectarse con el servidor de base de datos.

- ***-d | –drop=<db_name>***

  Esta opción permite la eliminación física de una base de datos. Nota: El uso de esta opción también implica la conexión con el servidor de base de datos.

- ***-e | –exec***

  Permite cargar el resultado de parsear el fichero de entrada en el servidor de base de datos. Esta opción se puede utilizar combinada con la opción '-c', que además implica el uso de la opción '–user'. Nota:El uso de esta opción requiere conectarse con el servidor de base de datos.

- ***-u | –user=<user_name>***

  Usa esta opción se usa junto con la opciones '-c' para indicar a qué usuario se le va a asociar la base de datos que se esta parseando. Úsala combinada con el comandos -p (directorio de trabajo) , para indicar a qué usuario queremos cambiarle el directorio de trabajo.

- ***-p | –path=<work_directory>***

  Esta opción permite establecer el directorio de trabajo para un determinado usuario (mediante -u), es decir, el directorio donde se van a ubicar todos las páginas Web enriquecidas(.hdb) y ficheros de configuración.

- ***–host:<host>***

  Mediante el uso de esta opción, puedes indicar el nombre(o dirección IP) de la máquina en donde el servidor de bases de datos se está ejecutando.

- ***–port:<port>***

  Con esta opción puedes indicar el número de puerto en el cual se encuentra el servidor de bases de datos.

- ***-h –help***

  Muestra la ayuda

Pasaremos ahora a ver cómo se utilizan todas las opciones del 'wag-manager' y cuál es su semántica. Además veremos ejemplos concretos:

1. **Cómo asociar una base de datos a un usuario**

   Supongamos, ahora, que tenemos un fichero de definición estructural de una base de datos (ejemplo.xdb), que ha sido creado por un usuario (usuario_ejemplo), al cual le vamos a asociar la base de datos. Un fichero de definición estructural sólo puede contener la definición de una única base de datos. Lo que tenemos que hacer es crear la base de datos, por lo que usamos la opción -c o –create. También queremos asociar esta base de datos al usuario, y para ello, usaremos la opción –user.

   ```
   wag-manager ejemplo.xdb -c nombre_db -
   u usuario_ejemplo
   ```

   En este momento, tendremos creada la base de datos definida en el fichero 'ejemplo.xdb' y estará asociada al usuario 'usuario_ejemplo'. Si el usuario especificado mediante la opción -u, no existiera, sería dado de alta de forma automática; esto quiere decir, que no es necesario dar de alta explícitamente a un usuario, sino que basta con asociar una base de datos a un usuario no existente.

   Las opciones -c, -e, -d y -p implica conectarse con el servidor de base de datos, por ello, si la máquina donde está instalado el servidor de base de datos y donde está 'wag-manager', no son la misma, será obligatorio el uso de la opción –host, para indicar cuál es la dirección del host en donde reside dicho servidor de base de datos. También si dicho servidor de base de datos no se encuentra en número de puerto que éste tiene por defecto, tendremos que especificarlo nosotros, mediante la opción –port.

2. **Cómo cargar la definición estructural de una base de datos en el servidor la base de datos**

   Cargar la definición estructural en el servidor, se refiere, a traducir toda la información contenida en el fichero de definición y almacenarla en el

servidor de base de datos. Para llevar a cabo esta tarea se usa la opción -e o –exec, será necesario que la base de datos haya sido previamente creada y asociada a un usuario. Siguiendo con el ejemplo anterior, supongamos que queremos cargar (o ejecutar) el fichero de definición 'ejemplo.xdb':

```
wag-manager ejemplo.xdb -e
```

Tenemos que tener en cuenta que si la base de datos no ha sido previamente creada y asociada a un usuario, el uso de la opción -e nos dará un error.

No obstante, podemos combinar la tarea de creación de la base de datos y asociación a un usuario, con la ejecución del fichero de definición del siguiente modo:

```
wag-manager ejemplo.xdb -c db_ejemplo -e -
u usuario_ejemplo
```

Llegados a este punto, y si no obtenemos ningún mensaje de error, ya se encuentra la base de datos cargada en el servidor. Esto implica que ya se encuentra disponible la aplicación Web que permitirá realizar el mantenimiento de los datos, por parte del usuario , así como la realización de consultas de dichos datos, por parte de cualquier internauta.

3. **Uso de 'wag-manager' como simple compilador de SQL**

   A parte del uso que hemos visto hasta ahora, la herramienta wag-manager, puede actuar como compilador de SQL. Esto quiere decir que se puede obtener un script SQL a partir de un fichero de especificación de base de datos XML-DB. Esto puede ser útil, si se quieren hacer algunas optimizaciones en la base de datos especificada por algún usuario, o bien como copia de seguridad, etc.

   Por tanto, podemos obtener un script SQL a partir del fichero de definición usando la opción -o.

   ```
   wag-manager ejemplo.xdb -o ejemplo.sql
   ```

   De este modo obtendremos un fichero de salida, denominado 'ejemplo.sql' que contiene la misma información que el fichero de especificación, sólo que en lenguaje SQL. No obstante, debemos saber que el SQL obtenido es especifico para PostgreSQL, de modo que puede no funcionar de forma correcta para cualquier otro sistema, que no guarde compatibilidad con éste.

   Si no indicamos el nombre del fichero de salida, como en este caso, obtendremos un fichero denominado 'out.sql'

   ```
   wag-manager ejemplo.xdb -o
   ```

   Otra opción que podemos utilizar es -a ó –add, la cual nos permite añadir nuevas definiciones estructurales a una base datos. Por eso, usando esta opción junto a la opción -o permite añadir la nueva información al fichero indicado, si es que existía previamente; en caso contrario, este será creado de forma automática.

```
wag-manager ejemplo.xdb -o ejemplo.sql
wag-manager ejemplo2.xdb -o ejemplo.sql -a
```

Se genera como salida un script SQL 'ejemplo.sql', que contiene las definiciones que provienen de ejemplo.xdb, junto con las que provienen de ejemplo2.xdb.

4. **Configuración del directorio de trabajo del WAG para un usuario**

   Se puede establecer cual es el directorio de trabajo para un determinado usuario. El directorio de trabajo será el directorio donde se van a encontrar todos los ficheros pertenecientes a dicho usuario. Entre estos ficheros nos encontraremos con los ficheros de especificación de base de datos, paginas Web y ficheros de configuración. El directorio de trabajo se especifica mediante la opción -p ó –path. Esta opción irá combinada con la opción -u, para indicar a qué usuario se asocia dicho directorio de trabajo.

   ```
   wag-manager -u usuario_ejemplo --
   path=/usr/local/wag/users/d1/
   ```

   Aquí se establece el directorio '/usr/local/wag/users/d1/' como directorio de trabajo para el usuario 'usuario_ejemplo'.

5.2.2 Instrucciones de instalación y mantenimiento

5.2.2.1 Proceso de instalación del WAG

En primer lugar, debemos conseguir el fichero 'wag-1.0.tar.gz' (paquete WAG comprimido en formato GZIP). Seguidamente, este fichero se copiara en un fichero temporal, donde se descomprimirá mediante el siguiente comando:

```
$gunzip wag-1.0.tar.gz; tar -xvf wag-1.0.tar
```

Dentro del directorio donde se descomprima el fichero, se creará un subdirectorio denominado 'wag-1.0'. Dentro de este subdirectorio nos encontraremos los siguientes subdirectorios:

- document/ : Contiene esta documentación.
- bin/ : Contiene los ficheros Perl, que implementan la herramienta 'wag-manager'
- ag/ : Contiene el conjunto de ficheros PHP que implementan Generador de aplicaciones.
- data/ : Contendrá el conjunto de directorios de trabajo de los distintos usuarios.
- config/ : Contiene algunos ficheros que permiten la configuración del servidor de base de datos (en este caso PostgreSQL).

Procedemos, ahora, a la instalación del 'wag-manager', que debe ser llevada a cabo por el usuario 'root':

- Se crea el subdirectorio 'wag/' dentro de '/usr/local/.

  ```
  $mkdir /usr/local/wag ; chmod 777 /usr/local/wag.
  ```

- Se copia el subdirectorio 'bin/ dentro del subdirectorio '/usr/local/wag/'.

  ```
  $cp bin/ /usr/local/.
  ```

- Se copia el subdirectorio 'data/' dentro del subdirectorio '/usr/local/wag/'.

  ```
  $cp data/ /usr/local/.
  ```

- Se copia el subdirectorio 'config/' dentro del subdirectorio '/usr/local/wag/'.

  ```
  $cp config/ /usr/local/.
  ```

Procedemos a la instalación de Generador de Aplicaciones (AG):

Suponemos que disponemos de un servidor de Web Apache, preparado para interpretar ficheros PHP. En este caso, la instalación sólo consiste en copiar la carpeta 'ag/' dentro del árbol de documentos del servidor Web. Por ejemplo:

cp ag/ ruta_apache/htdocs/.

Llegados a este punto, el sistema WAG se encuentra totalmente instalado.

5.2.2.2 Inicialización del sistema WAG.

Para iniciazar el sistema WAG, debemos de dar los siguientes pasos. Estos pasos debe de llevarlos a cabo un usuario de PostgreSQL con la capacidad de crear bases de datos y de crear usuarios (por ejemplo, el usuario 'postgres'):

- Configurar el servidor de base de datos Postgres para que permita conexiones TCP/IP.

  ```
  $pg_ctl “-i” parametros-habituales-
  inicializacion
  ```

- Cargar el script 'wag_xml_db_root.sql' en el servidor de base de datos PostgreSQL, accediendo como un usuario con permisos para crear bases de datos y crear usuarios.

  ```
  psql -u -f wag_xml_db_root.sql
  ```

- Cargar el script 'wag_xml_db_user.sql' en el servidor PostgreSQL, accediendo como usuario 'xml_db_user', cuya contraseña es 'casariche2001'.

```
psql wag_xml_db -u -f wag_xml_db_user.sql
Username:wag_xml_user
Password:casariche2001
```

Finalmente el usuario que actúa como Manager del WAG, debe ejecuta el siguiente comando:

```
$/usr/local/wag/bin/initwag
opciones:
--
data=directorio: de datos para los usuario (por de-
fecto sera '/usr/local/wag')
--port:n_puerto : puerto donde se encuen-
tra el servidor de base de datos PostgreSQL
--host:host: host (nombre o IP) donde se encuen-
tra el servidor de base de datos PostgreSQL.
```

En este momento el sistema está listo para funcionar.

Podremos probar el funcionamiento del sistema accediendo con nuestro navegador a la página 'index-db.php', que se encuentra dentro del directorio 'ag/' copiado dentro del árbol de documentos del servidor Web. En esta página nos aparecerá una lista con las base de datos gestionadas por el WAG. En este caso, sólo nos aparecerá la base de datos denominada 'template', que es una base de datos de prueba.

5.2.2.3 Requisitos de WAG

La herramienta WAG requiere la existencia de varios paquetes instalados en el servidor, dónde ésta vaya a ser ejecutada:

- PHP (versión 4.x .Ver documentación de PHP para su instalación).
- Perl (versión 5.005 o superior)
- Se necesita tener instalado un servidor Web Apache, configurado para ejecutar ficheros PHP (versión 4). (Ver documentación de Apache, para su instalación)
- Tener instalado el módulo perl XML::DOM que actúa como interface orientada a objetos, permitiendo la manipulación de ficheros en formato XML. Este módulo perl requiere tener instalados previamente otros dos módulos (deben instalarse en el orden indicado):

  1. Módulo expat, que es un parser de XML de bajo nivel.
  2. El módulo que actúa como parser de XML 'XML-Parser'.

  Los módulos perl los encontrarás en la página de CPAN (www.cpan.org), y pueden ser instalados en el sistema usando los siguientes comandos:

```
$tar -cxvz modulo-perl.tar.gz
$cd modulo-perl
$perl Makefile.PL
$make
$make install (como 'root')
```

Por último, WAG necesita el acceso a un servidor de base de datos Postgres, aunque este no tiene por qué estar instalado en la misma máquina que el WAG.

5.3 Manual del usuario

En esta sección se describirá el modo de proceder de un usuario que quiere hacer pública una base de datos, haciendo uso de la herramienta WAG. También se describirá la función del lenguaje de descripción de base de datos y como debe ser utilizado. Seguidamente, nos centraremos el lenguaje utilizado para la definición de la interfaz (o apariencia) de las páginas Web que presentarán los datos.

Los pasos que debe de dar un usuario para hacer pública una base de datos son los siguiente:

1. En primer lugar, el usuario proporcionar la definición estructural de la base de datos, creando un fichero de definición de base de datos. Este fichero debe de estar escrito en lenguaje XML-DB. Este fichero contendrá la semántica relacionada con la estructura de la base de datos, es decir, cuales son las tablas existentes, que campos contienen cada una de ellas, etc.

2. Este fichero será procesado por el administrador del WAG, haciendo uso de la aplicación 'wag-manager'. Tras haber sido procesado, ya existe una aplicación Web, que permitirá al usuario hacer el mantenimiento de los datos.

3. También es posible, aunque es opcional, la definición del aspecto de las páginas Web que componen las aplicaciones Web. En este caso, el usuario deberá personalizar él mismo las páginas Web, haciendo uso de ficheros escritos en HTML-DB[1]. Estos fichero también serán enviados al servidor, junto con un fichero de configuración.

Como podemos observar, podemos obtener una aplicación Web que nos permita la interacción con nuestra base de datos, únicamente proporcionando la definición estructural de ésta.

[1]Un fichero HTML-DB es un fichero HTML al cual se le añaden una serie de elementos que permiten al usuario indicar donde deben ir los campos dentro de la página web.

5.3.1 Definición estructural de la base de datos

La definición estructural es el primer paso que debemos dar si queremos hacer pública una base de datos. Con la definición estructural lo que se consigue es definir, básicamente, cuales son las tablas que conforman la base de datos y cuales son los campos que pertenecen a cada una de las tablas. La definición estructural de una base de datos se hace mediante un fichero, haciendo uso del lenguaje denominado XML-DB. Este lenguaje es muy sencillo (lo puede utilizar cualquier usuario sin ningún conocimiento en programación) y flexible, ya que se adapta a cualquier tipo de usuario, desde los menos familiarizados con los conceptos asociados a las bases de datos, hasta aquellos más experimentados.

Los datos mínimos que deben aparecer en una definición estructural son:

1. El nombre de la base de datos, el cual permite identificarla de forma unívoca.
2. Las tablas que van a conformar la base de datos.
3. Campos que componen a cada una de las tablas.

Vemos estos conceptos con un ejemplo. Supongamos que queremos publicar los datos referidos a los libros que posee nuestra biblioteca. Estos datos serán: la referencia de cada libro, el título, al autor, la editorial, la fecha de edición y una breve descripción de argumento del libro.

En este caso, podemos dar como descripción estructural de la base de datos los siguientes datos:

1. Nombre de la base de datos: Biblioteca
2. Habrá una única tabla: Libro
3. Los campos que componen la tabla 'Libro' serán: referencia, título, autor, editorial, fecha de edición y descripción.

Como podemos ver, la especificación estructural es muy sencilla y es lo único que necesita WAG para poder generar la aplicación Web que permitirá: por un lado, el mantenimiento de la base de datos al usuario, y por otro, la consulta de cualquier internauta, de los datos que la base de datos contiene.

No obstante, y gracias a la flexibilidad que tiene el lenguaje XML-DB, se puede incluir de forma opcional más información en la definición estructural de la base de datos. Esto permitirá a aquellos usuarios con más experiencia, en el mundo de las bases de datos, obtener mayor provecho de las ventajas propias de las base de datos. Esta información adicional, a la que nos referimos, es el uso de tipos de datos para los campos, clave primarias, el uso de índices, configuración de las páginas Web donde se presentan la información, etc.

5.3.1.1 XML-DB básico

Para aquellos usuarios menos iniciados en el mundo de las bases de datos, vamos resaltar cuales son los elementos indispensables para definir una base de datos.

Lo mínimo que debe incluir una definición estructural de una base de datos es: el nombre de la base de dato (para poderla identificar de forma unívoca), el nombre de las tablas en la que van a almacenar los datos, y los campos que componen dichas tablas. Por tanto, los elementos indispensables son DB, TABLE y FIELD. Además omitiremos todos los atributos opcionales. En este caso, para la base de datos del ejemplo 'biblioteca', el fichero de definición quedará como sigue:

```
<DB name="biblioteca">
   <TABLE name="libro">
      <FIELD name="titulo" />
      <FIELD name="autor" />
      <FIELD name="editorial" />
      <FIELD name="fecha_edición" />
      <FIELD name="breve_descripción" />
   </TABLE>
</DB>
```

Como vemos, la sintaxis de la especificación se ha simplificado mucho, puesto que no hemos limitado a dar la información mínima que necesita WAG. Simplemente con una definición de este tipo, se puede obtener una aplicación Web asociada a la base de datos.

La sintaxis genérica queda como sigue:

```
<DB name="nombre_de_la_base_datos">
  [<TABLE name="nombre_de_la_tabla">
    [<FIELD name="nombre_del_campo" />]+
  </TABLE>]*
</DB>
```

1. El Elemento DB, sólo debe aparecer una sola vez. El atributo 'name' nos permitirá asociar un nombre a la base de datos.

2. Dentro del elemento DB añadiremos tantos elementos TABLE como tablas vaya a tener la base de datos. El atributo 'name' permite asociar un nombre a la tabla

3. Dentro de un elemento TABLE, añadiremos tantos elementos FIELD como campos posea la tabla. También indicaremos su nombre, mediante el atributo 'name'. Debemos añadir al menos un elemento FIELD dentro de la tabla, ya que no tendrá sentido una tabla sin campos.

5.3.1.2 El lenguaje XML-DB

El lenguaje XML-DB es un lenguaje basado en XML[2], que permite realizar la definición estructural de una base de datos. Este lenguaje tiene las características siguientes:

- Es un lenguaje muy sencillo, al estar orientado a personas que carezcan de experiencia en el uso de lenguajes informáticos y de bases de datos.
- Es un lenguaje flexible, ya que permite a usuarios con más experiencia en el diseño de base de datos, hacer uso de claves primarias, ajenas, índices, tipos de datos.
- Es un lenguaje extensible, permitiendo en un futuro agregar nuevos elementos que den más potencia al lenguaje.
- Pose todas las ventajas de los lenguajes basados en XML.

Un fichero XML-DB está formado por un conjunto de elementos, los cuales se encuentran distribuidos en forma de árbol. A su vez un elemento puede contener atributos. Los atributos permiten asociar información a un determinado elemento. Vemos pues, la estructura genérica de un documento XML-DB:

```
<DB name=”nombre”>
  [<TABLE name=”nombre”>
    [<FIELD name=”nombre” [type=”tipo”] [val-
ue=”valor”]
         [key=”yes”] [unique=”yes”] [not-
null=”yes”] />
    </FIELD>]+
    [<PKEY>
      [<PKFIELD name=”nombre” />]+
    </PKEY>]
    [<UNIQUE>
      [<UFIELD name=”nombre” />]+
    </UNIQUE>]
    [<WAG>
       [<SEC-
TION file=”fichero_hdb” [type=”tipo de sec-
cion”]/>]+
     </WAG>]
  </TABLE>]*
  [<INDEX name=”nombre” ta-
ble_name=”nombre_de_la_tabla”
         [unique=”yes”] >
    [<IFIELD  name=”nombre” />]+
  </INDEX>]*
</DB>
```

[2]Lenguaje de marcado extensible, creado por el consorcio W3C (XML: eXtensible Marckup Language)

Esta es la sintaxis del lenguaje XML-DB. Como podemos observar todo fichero XML-DB debe de comenzar por el elemento DB, el cual es obligatorio. Este elemento identifica la base de datos de forma unívoca. El resto del fichero, son las definiciones estructurales asociadas a dicha base de datos. Como podemos ver, las elementos estructurales que nos interesan son las tablas (tables), los campos (fields) y los índices (INDEX). Veamos cuál es la semántica de cada uno de los elementos y sus atributos asociados:

- **DB**. Este elemento es el primero que debemos colocar en un fichero XML-DB. Su uso es obligatorio e indica cual es el nombre de la base de datos que queremos definir. Este nombre la identificará de forma unívoca. El nombre de la base de datos debe ser los más descriptivo posible, y la longitud de dicho nombre no debe sobrepasar los 24 caracteres. Este elemento puede contener elementos TABLE y elementos INDEX.

  ATRIBUTOS

  1. **name="nombre"**. Permite indicar el nombre de la base de datos. Su longitud no debe superar los 32 caracteres[3]. Si los supera, dicho nombre será truncado. No se podrá omitir este atributo.

     ```
     <DB name="biblioteca"> ........ </DB>
     ```

- **TABLE.** Este elemento debe de estar encerrado dentro del elemento DB. Debe de existir un elemento TABLE por cada tabla de datos[4] que queramos añadir a la base de datos. Por tanto, pueden existir tantos elementos TABLE como se deseen. Lo normal es que exista, al menos, un elemento TABLE, aunque puede que no haya ninguno (este caso ya lo veremos más adelante). En su interior puede contener uno o varios elementos FIELD, un elemento UNIQUE (opcional) y/o un elemento PKEY (opcional).

  ATRIBUTOS

  1. **name="nombre"**. Nombre de la tabla. Su longitud no debe superar los 24 caracteres. Si los supera, dicho nombre será truncado. Su uso es obligatorio.

     ```
     <TABLE name="libro"> ........</TABLE>
     ```

- **FIELD.** Los elementos FIELD se colocan dentro del elemento TABLE. Tendremos que colocar un elemento FIELD por cada uno de los campos que formen parte de la tabla de datos, en cuestión. Por esa razón, debe de haber al menos un elemento FIELD dentro de un elemento TABLE (no tiene sentido una tabla sin campos). Podemos colocar tantos campos en una como se deseen, aunque no conviene que las tablas tengan un elevado número de campos.

  ATRIBUTOS

[3]Este restriccíon viene impuesta por el servidor de base de datos, en nuestro caso, PostgreSQL

[4]Tabla de datos: Conjunto de datos organizados por filas (registros) y columnas (campos). Este concepto está asociado a las bases de datos relacionales.

1. **name="nombre"**. Nombre del campo. Su longitud no debe superar los 32 caracteres. Si los supera, dicho nombre será truncado. Su uso es obligatorio.
2. **type="tipo_dato"**. Permite especificar qué tipo de datos va a almacenarse en este campo. Los valores que puede tomar este tipo los veremos más adelante, con más detalle. Es opcional.
3. **value="valor_por_defecto"**. Este valor, será el que tome por defecto este campo, cuando no se le de ningún valor. Es opcional.
4. **key="yes"**. Indica que este campo será clave dentro de la tabla. Es opcional.
5. **unique="yes"**. Indica que no pueden existir más de un registro con el mismo valor en este campo. Es opcional.
6. **notnull="yes"**. Con el uso de este atributo, estamos diciendo que este campo no puede tomar valor nulo[5]. Es opcional.

```
<FIELD name="libro_ref" type=integer key="yes" />
```

Este campo será clave de la tabla libro, una referencia que los identifica de forma unívoca. El tipo será un número entero.

```
<FIELD name="titulo" type=string not-
null="yes" />
```

En este campo, hemos indicado que es de tipo cadena de caracteres con longitud máxima y que no debe de haber ningún registro con este campo con valor nulo.

- **UNIQUE**. Este elemento va colocado dentro del elemento TABLE.Sólo debe de haber un único elemento UNIQUE, siendo su uso opcional. Sirve para indicar qué campos de la tabla no pueden tomar el mismo valor en más de un registro. Sólo podrá albergar en su interior elementos UFIELD, debiendo aparecer , al menos, un elemento UFIELD, pues, no tendrá sentido especificar la propiedad de unicidad sobre un conjunto de campos vacío. Este elemento no tiene atributos.

- **PKEY**. Sólo podemos utilizar un como máximo un elemento PKEY . Su uso es opcional. Permite indicar cuáles son los campos que forman la clave primaria[6]. Dentro de este elemento, únicamente podrán aparecer elementos PKFIELD. Al menos deberá aparecer uno, ya que no tendrá sentido una clave primaria referida que no esté referida a algún campo .

- **INDEX**. Elemento INDEX sólo puede ir dentro de un elemento DB. Se pueden incluir tantos elementos INDEX como sean necesarios. También es opcional su uso. Este elemento permite crear índices sobre uno o más campos de una tabla. No obstante debemos conocer bien cuál es el significado de los índices (lo vamos a ver más adelante), debido a que un uso

[5]Valor nulo, es la ausencia de valor. No hay que confundirlo con la cadena vacía o el valor cero.
[6]Clave primaria: conjunto de campos de una tabla que toman los mismos valores en registros distintos. Además no existe ningún subconjunto que cumpla la propiedad anterior.

indebido de ellos pueden afectar en el buen funcionamiento de la base de datos. Dentro del elemento INDEX sólo podrán aparecer elementos IFIELD. Al menos un elemento IFIELD debe de aparecer, puesto que no tiene sentido crear un índice sin especificar a qué campos se debe aplicar.

ATRIBUTOS

1. **name="nombre_índice"**: Permite indicar el nombre de la base de datos. Su longitud no debe superar los 32 caracteres. Si los supera, dicho nombre será truncado.
2. **tablename="nombre_de_la_tabla".** Este atributo permite especificar a qué tabla se debe asociar el índice. Su uso es obligatorio.
3. **unique="yes"**. Permite hacer que los valores que tomen los campos del índice sean únicos, es decir, no exista más de un registro con el mismo valor en dichos campos. Uso opcional.

```
<INDEX name="libro_índice" ta-
ble_name="libro" unique="yes">
     <IFIELD name="titulo" />
      <IFIELD name="editorial" />
</INDEX>
```

Crea un índice para la tabla 'libro', sobre los campos 'titulo' y 'editorial', y además obligando a que tomen valores únicos.

5.3.1.2.1 Tipo de datos En este subapartado vamos a ver cuales son los tipos de datos que podemos utilizar y cuáles son las características asociados a cada uno de ellos.

- ***string y string(n)***: Este tipo se utiliza cuando vamos a almacenar cadenas de caracteres. Si utilizamos simplemente el tipo string, entonces estamos indicando que el tamaño de la cadena va a ser variable, y no sabemos con certeza cuanto puede ser la longitud de esta. Si le asociamos un número n entre paréntesis, le indicaremos cual será la longitud máxima de dicha cadena de caracteres.
- **bool**: En este caso lo que se van a almacenar son los valores lógicos 't' (cierto) o 'f' (falso).
- **integer**: Este tipo nos va a permitir almacenar datos numéricos de tipo entero. El rango permitido para este tipo será desde el -2147483648 al +2147483647 .
- **dinteger:** También permite la entrada de datos numéricos de tipo entero. El rango permitido será desde el -9.2233720369e+18 al +9.2233720369e+18.
- **sinteger:** Entrada de datos de tipo entero, cuyo rango permitido va desde -32768 a +32767
- **float**: Entrada de datos numérico de tipo real . Permite hasta un máximo de 6 decimales.

- **lfloat**: Entrada de datos numéricos de tipo real, permitiendo un máximo de hasta 15 decimales. Ejemplo: 34,23493094.
- **time:** Permite la entrada de una hora en el formato hh:mm:ss o hh:mm. Ejemplo: 23:34:42.
- **date:** Permite la entrada de una fecha con los siguientes formatos: dd/mm/aaaa, dd/mm/aa o dd/mes/aaaa. En este último caso se puede utilizar el mes en letra (en inglés: january, february, march, april, may, june, july, etc.). También se puede utilizar la abreviatura de cada mes (en inglés: jan,feb,mar,apr,may,jun, etc.) Ejemplo: 12/10/99 , 12/10/1999 , 12/october/1999 o 12/oct/99.
- **datetime:** Permita la entrada a la vez de una fecha y una hora. Ejemplo: 12/10/1999 14:35:23.

5.3.1.2.2 Los índices El uso de índices sobre un campo de una tabla permite que la búsqueda de registros dentro de ésta se hagan de forma más eficiente, cuando se impongan condiciones sobre dicho campo. Un índice para una tabla, de forma similar a lo que ocurre con el índice de un libro, permite que el sistema encuentre un determinado registro, sin necesidad de comprobar todos y cada uno de los existentes en la tabla. Siguiendo con el símil del índice en un libro, dicho índice permite al lector buscar una página determinada sin necesidad de tener que buscar en cada una de las páginas del libro.

En nuestro ejemplo de la base de datos 'biblioteca', si se realizan muchas búsquedas de libros, por ejemplo, a través del campo 'Autor', sería aconsejable asignar un índice a la tabla 'libro' sobre el campo 'autor'.

```
<INDEX name="libro_autor_índice" ta-
ble_name="libro" unique="yes">
  <IFIELD name="autor" />
</INDEX>
```

Como podemos ver, el atributo <<unique="yes" >>, fuerza a la unicidad del valor de una columna o de la combinación de varias columnas.

5.3.2 Definición de una interface propia

Como ya hemos visto antes, sólo con la definición estructural de la base de datos, se va a construir una aplicación Web, asociada a dicha base de datos. De hecho, vamos a poder acceder de forma independiente a cada una de las tablas que conforman la base de datos. Puesto que las páginas Web, que constituyen la aplicación, se generan de forma automática, sin la intervención del usuario, es posible que se produzcan algunos desajustes en la distribución de los campos en las páginas, o que simplemente, el usuario desee personalizar algunos elementos de las páginas Web.

Por todo lo expuesto anteriormente y para dar mayor flexibilidad al usuario, la herramienta wag ofrece la posibilidad de que sea el usuario mismo quien

diseñe las páginas Web que constituyen la aplicación Web, y que mediante una serie de elementos especiales pueda especificar qué campos de la base de datos desea que aparezcan y dónde desea ubicarlos.

En primer lugar, vamos a ver cuál es la estructura de páginas de una aplicación Web asociada a una base de datos determinada. Dado que cada una de las tablas de una base de datos debe de poder ser consultada y mantenida, es necesario que para cada tabla existan las siguientes posibilidades:

1. **Inserción de datos (INS)**: Esta sección será de acceso restringido, sólo para el propietario de la base de datos a la cual pertenece la tabla. Esta sección está compuesta por una única página Web. Esta página Web está dividida en dos marcos. Uno de ellos es configurable por el usuario, y permite la entrada de datos a la tabla de forma manual. En el otro marco, lo que nos encontramos será la posibilidad de cargar los datos en la tabla desde un fichero de texto, el cual se enviará al servidor. Este marco no es configurable por el usuario.

2. **Consulta de datos (CONS)**: Esta sección será a la que podrán acceder cualquier usuario de Internet. Esta sección está a su vez formada por tres páginas Web, que son configurables por el usuario. La primera es la página de consulta de datos (SCONS1); la segunda, es la página que muestra el listado de registros que podemos consultar (SCONS2); y por último, la página que muestra el registro consultado (SCONS3).

3. **Modificación de datos (UPD)**: Esta sección será de acceso restringido, sólo para el propietario de la base de datos a la cual pertenece la tabla. También está constituida por tres páginas configurables: la página de consulta de datos (SUPD1), la página que muestra el listado de registros solicitado (SUPD2) y la página que permite modificar el registro indicado (SUPD3).

4. **Eliminación de datos**: Esta sección será de acceso restringido, sólo para el propietario de la base de datos a la cual pertenece la tabla. La sección de eliminación de datos, posee también tres páginas: la página de consulta (SDEL1), el listado de registros resultantes (SDEL2) y la página que muestra el registro solicitado y que podemos eliminar.

5.3.2.1 Configuración de las páginas

Para realizar la configuración de las páginas se utilizará el elemento WAG asociado a una tabla de datos. Este elemento nos permiten asociar los ficheros HTML-DB a las distintas secciones y páginas de una de una tabla.

La sintaxis de este elemento, dentro de un fichero XML-DB es la siguiente:

```
...
<TABLE ...>
```

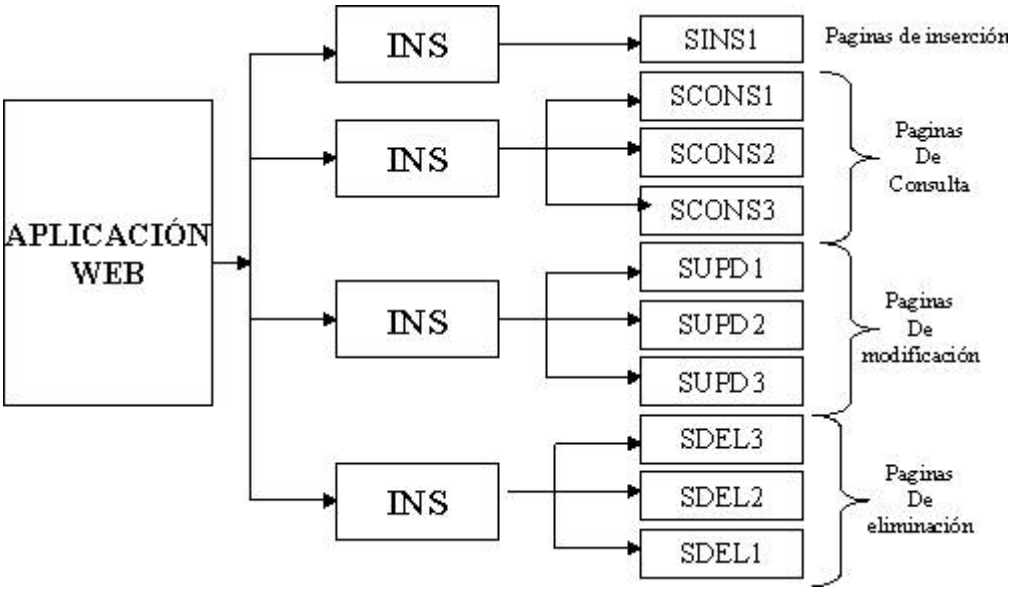

Figura 5.3: Estructura de una aplicación Web

```
    [<WAG>
     [<SECTION file="fichero_HTML-
DB" type="tipo_seccion"/>]+
    </WAG>]
</TABLE>
...
```

- El elemento para la configuración de una aplicación Web es WAG. Dentro de este elemento sólo podemos colocar elementos SECTION.
- El elemento SECTION permite configurar una sección o página para una tabla. Dentro de este elemento no podrán alojarse ningún nuevo elemento.Podemos utilizar tantos elementos SECTION como subsecciones de una aplicación queramos configurar (un máximo de 10 subsecciones, por tanto de 10 elementos)

    1. El atributo 'file', indica el nombre del fichero de configuración de la sección (formato HTML-DB). Este fichero será buscado tomando como base el directorio de trabajo del usuario propietario de la base de datos, a la cual pertenece la tabla que estamos configurando. El uso de este atributo es obligatorio.
    2. El atributo 'type', que es opcional, indica a qué sección (o página) de la estructura de una aplicación Web, queremos configurar. Si no especifica ninguna, se utilizará sobre todas las páginas que componen la aplicación Web

Veamos los siguientes ejemplos:

Configuración de la sección CONS de la aplicación asociada a la tabla “libro”

```
...
<TABLE name=”libro”>
...
  <WAG>
    <SEC-
TION file=”biblioteca_cons.hdb” type=”CONS” />
  </WAG>

...
</TABLE>
```

Configuración de todas las secciones de la aplicación asociada a la tabla “libro” .

```
...
<TABLE name=”libro”>
...
  <WAG>
    <SECTION file=”biblioteca_cons.hdb”/
  </WAG>

...
</TABLE>
```

Configuración de las páginas SUPD1 y SUPD3 de la tabla “libro” de la base de datos “biblioteca”.

```
...
<TABLE name=”libro”>
...
  <WAG>
    <SECTION file=”pagina1.hdb” type=”SUPD1” />
    <SECTION file=”pagina3.hdb” type=”SUPD3” />
  </WAG
...
</TABLE>
```

Obsérvese, como cuando un elemento no contiene elementos dentro de su interior, se puede utilizar el formato <SECTION .... />, en vez de <SECTION ... >....</SECTION>. Esta notación es siempre válida para lenguajes basados en XML, como es este caso.

5.3.2.2 El lenguaje HTML-DB

El lenguaje HTML-DB no es más que un enriquecimiento del lenguaje HTML con una serie de elementos. De este modo, los usuarios podrán, por un lado,

construir ellos mismo las páginas Web que componen la aplicaciones (uso de los elementos propios de HTML). Por otro lado, pueden ubicar y dar formato a los campos que se desean presentar, dentro de dichas páginas (uso de los elementos nuevos introducidos por WAG).

No obstante, nosotros no vamos a centrar en ver con detalles cuáles son los elementos añadidos por HTML-DB a HTML. Para obtener información sobre HTML , ver "Capítulo 2 , Apartado 2.3 Tecnologías utilizadas" de este documento.

1. **Elemento WAG-INPUT**

   WAG-INPUT especifica varios tipos de controles asociados a un campo, para la entrada de datos. El tipo de control se especifica con el atributo TYPE, que puede tomar los siguientes valores:

   (a) **HIDDEN**: el elemento de entrada no se muestra al usuario, aunque su contenido se envía al servidor.

   (b) **RESET**: botón de reset, que borra todos los datos introducidos o y se restablecen los valores por defecto.

   (c) **SUBMIT**: botón de envío. Al ser pulsado, se envían los datos al servidor de base de datos.

   (d) **TEXT**: línea de texto. El atributo SIZE, que es opcional , permite indicar su tamaño.

   Otros atributos del elemento WAG-INPUT son los siguientes:

   - **NAME**: nombre del campo al que queremos asociar el elemento de entrada/salida
   - **MAXLENGTH**: sólo válido para TYPE="TEXT" y TYPE="PASSWORD", indica el número máximo de caracteres de la caja de texto. Puede ser mayor que el tamaño de la caja, en cuyo caso el texto se desplazará dentro de ella. Por defecto, este valor es ilimitado
   - **SIZE**: sólo válido para TYPE="TEXT" y TYPE="PASSWORD", indica el tamaño de la caja que contiene al texto.
   - **VALUE**: valor inicial del elemento de entrada.
   - **TABINDEX**: orden que ocupa el campo dentro de la página.

2. **Elemento WAG-TEXTAREA**

   Este elemento permite ligar un elemento TEXTAREA de HTML a un campo de la tabla . Mediante el uso de este elemento, podemos introducir un bloque de texto a través de una ventana de texto con barras de desplazamiento. El tamaño de la ventana se especifica con los atributos ROWS y COLS. El texto puede tener un tamaño ilimitado y no pasa a la siguiente línea al llegar al final de la ventana, siendo necesario para ello introducir un retorno de carro manual.

   Los atributos de este elemento son:

   - **NAME**: nombre del campo de la base de datos que queremos asociar al elemento de entrada/salida.

- **COLS**: número de columnas de texto de la ventana.
- **ROWS**: número de líneas de texto de la ventana.
- **TABINDEX**: orden que ocupa el campo dentro de la página.

```
<WAG-
TEXTAREA COLS=50 ROWS=5 NAME="nombre_del_campo">
  valor por defecto
</WAG-TEXTAREA>
```

3. **Elemento WAG-NEXT**

   Con este elemento le estamos indicando a WAG donde debe de colocar un enlace para ir a la página siguiente.

   ```
   <WAG-NEXT>Siguiente</WAG-NEXT>
   ```

4. **Elemento WAG-BACK**

   Con este elemento indicaremos al WAG dónde debe de colocar el enlace para volver a la página anterior.

   ```
   <WAG-BACK>Anterior</WAG-BACK>
   ```

5. **Elemento WAG-FORM**

   Este es un elemento que permite insertar de forma automática un formulario dentro de la página Web, que contiene todos los campos asociados a una tabla. Todos los campos serán presentados mediante elementos INPUT, de tipo TEXT. Este Elemento no puede usarse junto con un elemento WAG-INPUT, WAG-SELECT o WAG-TEXTAREA. Este elemento es idóneo cuando queremos construir una página Web que esté asociada a más de una tabla de datos, ya que es un elemento genérico, es decir, no especifica qué campos son los que deben de aparecer o no.

   Los atributos de este elemento son:

   - **MAXLENGTH**: indica el número máximo de caracteres de la caja de texto. Cada caja de texto tiene asociado una longitud máxima por defecto, dependiendo del tipo de dato que se almacene en el campo asociado a dicha caja de texto. Por ello, el valor de este atributo sólo se tendrá en cuenta cuando dicho valor sea menor que el que tiene asociado la caja de texto por defecto. Si por el contrario, la longitud especificada en el atributo es mayor que el valor de longitud por defecto, se tomará éste último.
   - **SIZE**: indica el tamaño máximo que debe tener una caja que contiene texto. De forma análoga al atributo MAXLENGTH, cada caja de texto tiene asociado un tamaño máximo por defecto, en función del tipo de datos del campo asociado a la caja. Si el tamaño especificado aquí es menor que el tamaño por defecto, se tomará como valor de longitud la especificada con el atributo SIZE. Si por el contrario, la longitud por defecto es menor a la especificada por el atributo, se tomará como válido el valor por defecto.

     ```
     <WAG-FORM maxlength=100 size=60>
     ```

6. **Elemento WAG-TITLE**

   Este elemento permite al usuario colocar un título a la página Web. Este título aparecerá con un formato por defecto y permitirá indicar la sección, dentro de la aplicación Web, en la que nos encontramos. Podemos personalizar el color de fondo del título y el color del texto mediante los siguientes atributos:

   - **BGCOLOR:** indica el color de fondo con que aparecerá en la página.
   - **TEXTCOLOR**: Indica el color del texto.

7. **Elemento WAG-LIST**

   Este elemento va a permitir indicar al WAG, dónde debe aparecer dentro de una página, la lista de enlaces a los registros resultantes de una consulta. Este elemento sólo tiene sentido colocarlo, por tanto, en aquellas páginas posteriores a una página de consulta (explícita o implícita), es decir, sólo en las páginas pertenecientes a las subsecciones SCONS2, SUPD2 y SDEL2. El usuario puede personalizar en este caso el color de fondo de la lista y el color del texto que aparece dentro de ella. Los atributos son:

   - **BGCOLOR**: Permite establecer el color de fondo de la lista.
   - **TEXTCOLOR:** Asigna un color de al texto que aparece dentro de la lista de enlaces.

Capítulo 6

Conclusiones

Como ya se ha explicado con más detalle en capítulos anteriores, la herramienta WAG tiene dos objetivos principales: hacer más sencillo a los usuarios el proceso de publicación de bases de datos en Internet, a través de la Web; y por otro lado, facilitar la tarea de administración de las bases de datos de las aplicaciones Web al Webmaster.

Son muy escasas las herramientas [1] existentes, que van dirigidas hacia un objetivo igual o similar que WAG.

La herramienta WAG ofrece importantes ventajas, entre las que podemos destacar:

- La herramienta WAG ha sido realizado como parte de un proyecto fin de carrera, como software de libre distribución . El código de la aplicación, por tanto, podrá utilizarlo, modificado y distribuido de forma libre. Pero además, toda el software sobre el cual se apoya WAG, así como las herramientas empleadas para su elaboración, son de libre distribución.
- Es multiplataforma. La herramienta WAG pude correr sobre plataformas tales como Linux, Unix, Windows. Todo el software sobre el que se apoya WAG es multiplataforma, por lo que propiedad es heredada por la herramienta WAG.
- No sólo es una herramienta para crear aplicaiones Web con acceso a bases de datos , sino que proporciona un servicio de mantenimiento de los datos a los usuarios y de administración al Webmaster.
- Los usuarios pueden optimizar la apariencia de las páginas donde aparecerán los datos, de forma muy flexible, usando páginas Web estándares con HTML, a las cuales se le añaden una serie de marchas especiales para indicar la distribución de la información dentro de la página Web (HTML-DB)
- Está orientada a aquellos usuarios menos familiarizados con los entresijos de la programación en Internet, aunque también puede ser utilizadas

[1] Son herramientas comerciales

por personas más experimentadas. No se necesita programación, por parte de los usuarios, a diferencia de las mayorías de las herramientas existentes.

- Es una herramienta abierta posibles extensiones, que pueden aumentar sus prestaciones.

Existen algunas desventajas de la herramienta WAG, con respecto a otras alternativas existentes:

- WAG suele proporcionar aplicaciones con menos funcionalidad, como puede ser la generación de consultas complejas (impliquen varias tablas), lo cual si suele ser soportado por algunas herramientas comerciales existentes. Este es el precio a pagar por la reducción de complejidad para el usuario, a la hora de definir la base de datos.

-Ausencia de una interface gráfica que haga más intuitiva, si cabe, la especificación de los datos que se publicarán en la aplicación Web.

No obstante, WAG puede ser fácilmente extendido gracias a que es un software de libre distribución. Por un lado, se puede habilitar una herramienta gráfica que permita generar al usuario de forma automática ficheros de especificación de base de datos (XML-DB) y por otra parte, se puede habilitar mecanismos para permitir la creación de consultas que implique más de una tabla de datos pertenecientes a una misma base de datos. También es posible complementar WAG con más módulos para soportar otros tipos de servidores (MySQL, Oracle,etc), a parte del actualmente soportado (PostgreSQL).

Apéndice A

Código del 'Gestor del WAG'

A.1 wag-manager.pl

A.2 COMPILER.pl

A.3 ELEMENT.pl

A.4 METADATA.pl

A.5 USERDATAMANAGER.pl

A.6 DB.pl

Apéndice B

Código del Generador de Aplicaciones (GA)

B.1 ins.php

B.2 ins_upload.php

B.3 upload.php

B.4 cons.php

B.5 upd.php

B.6 del.php

B.7 messager.php

B.8 menu.php

B.9 index-db.php

B.10 index-table.php

Bibliografía

[1] XML Bible. Elliotte Rusty Harold. Ed. IDG Books Worldwide 1999

[2] Programming Perl. Larry Wall, Tom Christiansen y Randal L. Shwartz. O'Really & Associates, Inc. 1996

[3] Creación de aplicaciones web a través de ejemplos.Jesús Bobadilla Sancho y otros. Ed. RA-MA 2000

[4] Creación de BD en Internet. Joseph Sinclair y Carol McCullough .Ed. Anaya Multimedia 1997

[5] Conceptos de bases de datos. Conceptos fundamentales. Ramez Elmasri y Shamkant B. Navathe. Ed. Addison-Wesley 1997

[6] Introducción a los sistemas de base de datos.C.J. Date. Ed. Prentice Hall 2001

[7] PHP4. Pedro Fábrega. Ed.: Prentice Hall 2000

[8] Creación de aplicaciones Web con PHP4. Tobias Ratschiller y Till Gerken. Ed. Prentice Hall 2000

[9] Documentation for PostgreSQL version 7.1. http://www.postgresql.org