3.- WAINE

3.1.- Descripción

WAINE son las siglas de "*Web Application & INterface Engine*". Se trata de un producto pensado para desarrollar aplicaciones web de una forma rápida y sencilla, permitiendo desarrollar aplicaciones multiusuarios. WAINE tiene como características principales: eficiencia, independencia, simplicidad y reutilización.

La arquitectura de WAINE se basa en un modelo que captura las relaciones entre los principales aspectos de una típica aplicación de gestión (usuarios, menús, formularios…) y el desarrollo de la aplicación se basa en un lenguaje de descripción.

El sistema almacena en una metabase de datos (MDB) toda la información referente a la estructura de la aplicación: usuarios, grupos, formularios, informes, menús para los diferentes usuarios de la aplicación, etc.

El código de la aplicación a realizar se escribe en XML y mediante la ejecución de un script, el sistema pasa todos los datos del código XML a la metabase de datos para posteriormente generar la estructura de la aplicación en concreto.

Un motor (servidor de aplicaciones) se ejecuta en el servidor, y genera los distintos documentos de salida en sus correspondientes formatos (HTML, PHP, PDF…), atacando a las diferentes fuentes de información configuradas (bases de datos, servidores, LDAP, ficheros de textos, directorios, etc.).

El cliente recibe toda esta información debidamente estructurada bien a través de una red local o bien a través de Internet. En la siguiente figura podemos apreciar el funcionamiento.

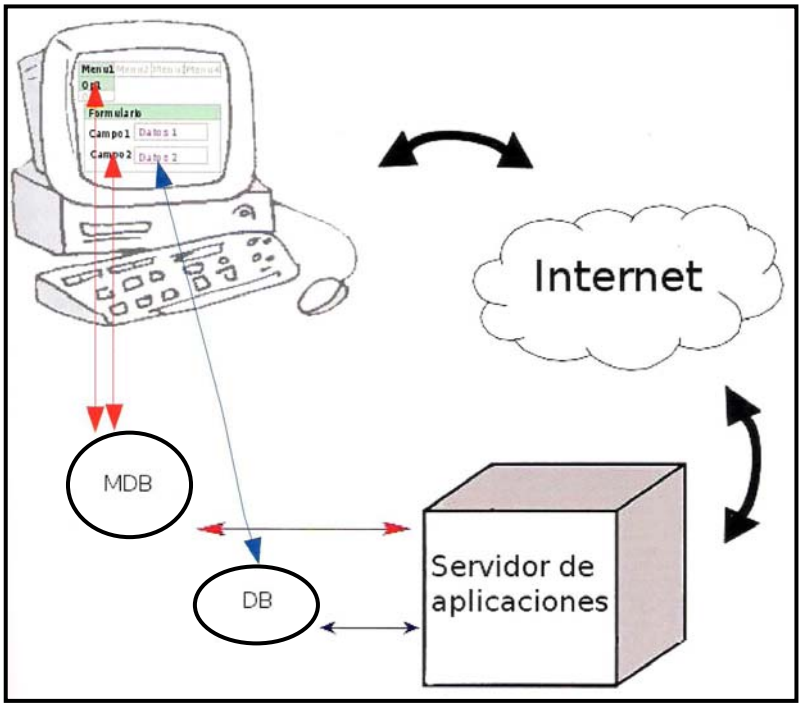

Figura 1.- Funcionamiento de WAINE

3.2.- Arquitectura

La mayoría de las aplicaciones generadas por WAINE suelen seguir una arquitectura de tres niveles: datos, aplicación y presentación, que queda reflejada en la siguiente figura:

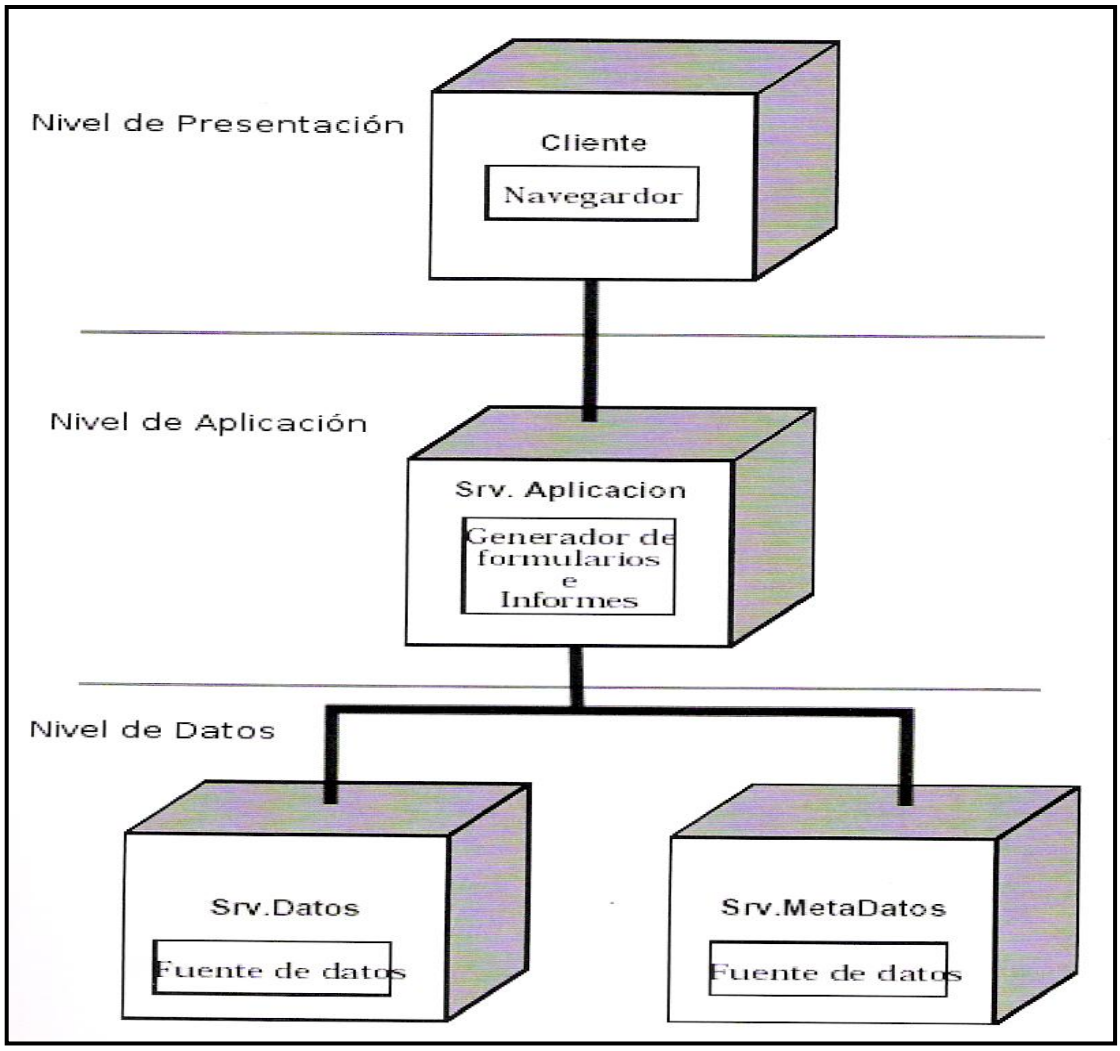

Figura 2.- Arquitectura de 3 niveles

A continuación se detallarán cada uno de los niveles antes mencionados:

- El nivel más bajo es el nivel de datos. En este nivel se tienen todos los datos necesarios para la ejecución de la aplicación. Habrá dos tipos de datos, información relativa al funcionamiento, como serán descripciones de formularios y sus campos y otros aspectos necesarios a partir de los cuales se generará la aplicación, y las fuentes de datos propias de la aplicación, es decir, toda la información que se mostrará o introducirá el usuario y cuyo tratamiento es el objetivo de la aplicación. La información de la aplicación se guardará en las metabase de datos, MDB, sobre la que entraremos en detalle posteriormente.

- El nivel intermedio es el nivel de aplicación. Aquí encuentra el servidor de la aplicación, cuya misión es traducir los datos de la MDB y generar los diversos formularios, menús y otros componentes que formarán la aplicación y que se mostrarán darán al cliente. En este nivel y encargado de las funciones mencionadas se encuentra el motor de WAINE, el cual describiremos posteriormente.

- El tercer y último nivel es el de presentación, encargado de mostrar al usuario cliente de forma adecuada los resultados que le entrega el nivel de aplicación, según cada caso.

La arquitectura de WAINE se basa en cuatro elementos básicos como se ilustra en la figura 2:

- La base de datos de la aplicación (DB). Esta base de datos contiene toda la información que va a manejar la aplicación. El sistema permite la conexión a diferentes fuentes de datos lo que permite independencia de la fuente de datos.

- La Meta-Base de datos (MDB). Después de que los elementos de la aplicación (forms, structs, users, menus, etc…) son creados, estos se representan en una base de datos relacional, la meta-base, cuyo esquema se representa en la figura 4, en el siguiente apartado dedicado a la descripción de dicha base de datos. Para la representación de esta base de datos pueden ser usados diferentes sistemas de implementación los cuales permiten más independencia para el sistema.

- El motor de la aplicación. Dicho motor accede a la meta-base de datos de la aplicación para obtener la información para generar automáticamente la aplicación (menús, formularios, etc…) de forma adecuada en cada momento, y accede a la base de datos de la aplicación para ejecutar las acciones básicas (create, update an delete) y ejecutar las acciones y eventos definidas para los usuarios y seleccionadas por estos.

- El directorio de configuración. El sistema puede ser configurado, modificado para requisitos particulares, y extendido fácilmente mediante la fijación de una serie de parámetros del directorio de configuración.

La endoarquitectura del sistema también es muy simple. Se pueden apreciar dos capas de uso común:

- La capa de acceso de datos: que permite acceder tanto a la MDB como a las diferentes fuentes de datos empleadas por la aplicación

- La capa de render: encargada de generar los distintos documentos de salida.

A ambas capas acceden los distintos módulos del sistema, el de menús, el de informes, el de formularios… Todos ellos se ejecutan sobre el módulo de usuarios y seguridad, que garantizan la validación de la identidad del usuario a lo largo de toda la aplicación, así como el control de acceso a los objetos a los que éste tiene permiso de acceder. En la siguiente figura se muestra un esquema de la endoarquitectura de WAINE.

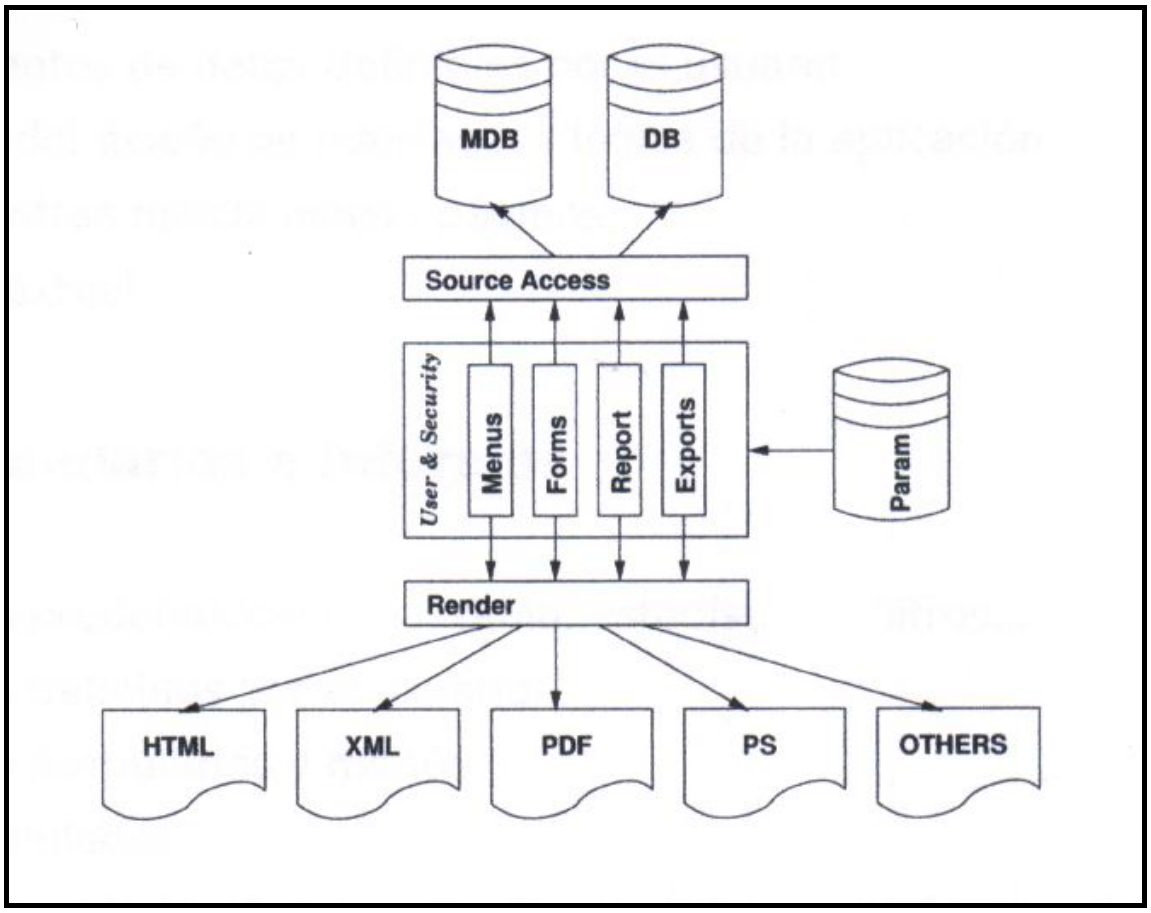

Figura 3.- Endoarquitectura de WAINE

3.3.- MDB

La meta-base de datos es uno de los pilares básicos de toda aplicación WAINE. Aunque estrictamente no se trata nada más que de una base de datos, como puede ser la que contiene los datos propios que la aplicación gestiona, ésta permitirá al motor WAINE generar los diversos elementos que darán lugar a los resultados que ve el cliente. De este modo, en un mismo servidor se podrán tener numerosas aplicaciones de este tipo, dependerá de la potencia del servidor, y cada una tendrá su propia MDB, que el servidor de la aplicación leerá cada vez que haya una petición a la aplicación en cuestión.

La meta-base de datos está estructurada en una serie de tablas, cada una de las cuales hace referencia a una parte bien diferenciada de cada aplicación. A continuación se procede a realizar una descripción más o menos general de las mismas, sus campos

más importantes y las posibles relaciones entre ellas. Antes de pasar a detallar las diferentes tablas y sus campos, en la figura que a continuación se muestra podemos ver el esquema entidad-relación, ERD de la meta-base de datos.

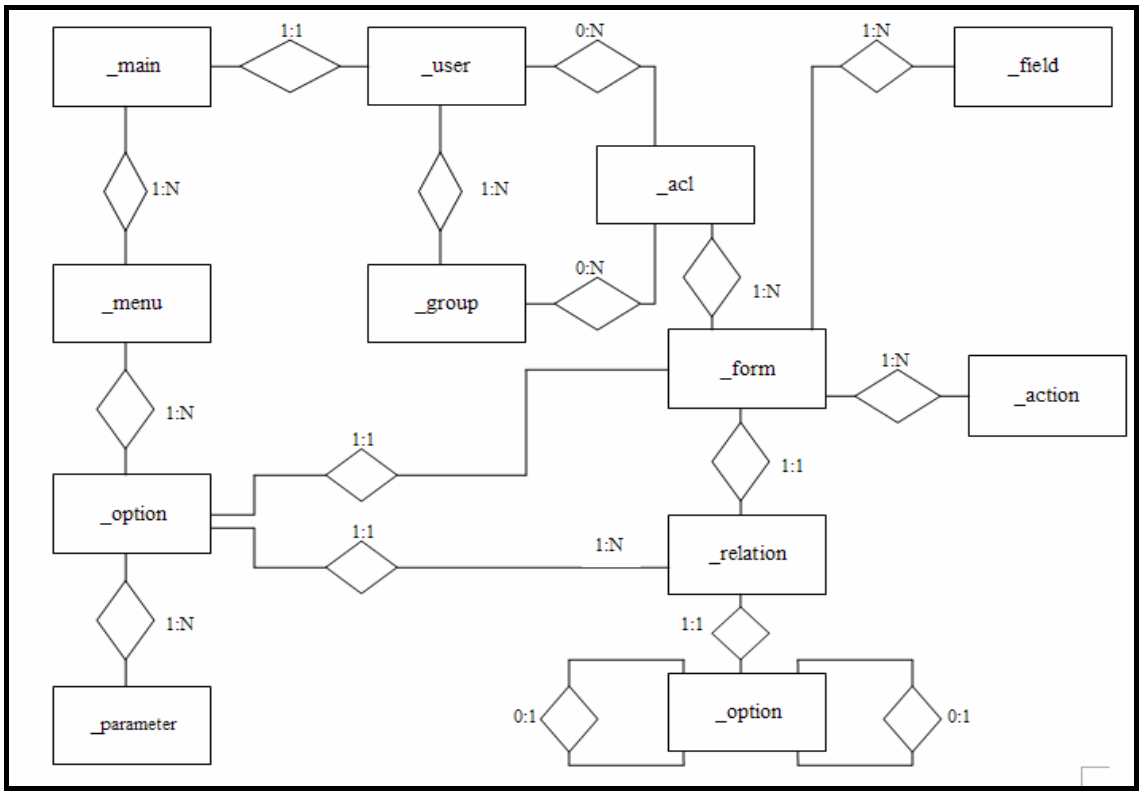

Figura 4.- ERD de la meta-base de datos

- Tablas relativas a la identificación de los usuarios, sus privilegios de acceso, y otros parámetros importantes a tener en cuenta.

  1. **_group:** contiene la información relativa a los distintos grupos de usuarios. Cada grupo de usuarios tendrá una serie de privilegios o características que determinarán la visión que los mismos tendrán de la aplicación. Se distinguen los siguientes campos principales:

     - gid: identificador único del grupo.
     - name: nombre del grupo.
     - descr: breve descripción del grupo.

  2. **_user:** almacena la información específica de cada uno de los usuarios. A destacar:

     - uid: identificador de usuario. Único para cada uno.
     - gid: identificador del grupo al que pertenece el usuario.
     - name: nombre del usuario.

- passwd: contraseña asociada al usuario, y que le permitirá el acceso a la aplicación.
- mainid: identificador del main de usuario. El main indicará que menús, opciones, etc, de la aplicación serán accesibles por el usuario, es decir, se le mostrarán a este cuando entre en la aplicación.
- descr: breve descripción del usuario

3. **_main:** como se ha mencionado, cada grupo de usuarios tendrá una visión de la aplicación, es decir, tendrá acceso a una serie de datos, verá un determinado menú con unas determinadas opciones, una serie de formularios, etc. Para conseguir esto por cada vista distinta que se quiera tener deberemos contemplar un main distinto, podría decirse que una aplicación distinta, que determinará las distintas opciones que se le mostrarán a los usuarios de cada grupo. Cada uno de estos main tendrá una entrada propia en esta tabla.

   - mainid: identificador del main.
   - caption: nombre asignado al main.

- Tablas relativas a la creación de los menús de usuario.

1. **_menu:** contiene la información relativa a cada menu. Un main estará compuesto por uno o varios elementos menu.

   - menuid: identificador único del menu.
   - mainid: identificador del main al que está asociado el menu. Un main estará formado por todos los elementos menu que compartan este identificador. Un mismo menu puede aparecer en varios main, pero será necesario definirla de nuevo para cada caso.
   - ord: indica el orden del menú entre todos los demás elementos menú del main asociado. Este parámetro es el que nos permite establecer un orden entre los distintos elementos menu, de forma que podemos indicar como en que posición queremos que se muestre cada uno de ellos.
   - caption: texto asociado al menu y que aparecerá cuando se cargue el mismo.
   - img: permite asociar una imagen al menu.

2. **_option:** guarda la información relativa a cada una de las opciones de cada menu.

   - optionid: identificador único del elemento option.

- menuid: identificador del menu al que está asociado el option en cuestión. Igual que ocurría entre un menu y su main, un mismo option puede aparecer en varios menus, pero será necesario definirlo de nuevo para cada caso.
- ord: indica el orden del option entre todas los demás option del menu asociado.
- caption: texto asociado al option y que aparecerá cuando se cargue el contenido del menú asociado.
- structid: identificador de la estructura asociada con este option. Las estructuras podrán asociarse a más de un menu, de forma que se podrán tener menus con opciones idénticas para usuarios muy diferentes sin tener que repetir el código en exceso.
- url: url del fichero a ejecutar cuando se selecciona el option.
- img: permite asociar una imagen a la opción.

3. **_parameter:** almacena parámetros, elementos param, importantes relativos a cada elemento option de un menu, como serán el formulario al que ataca el option en cuestión o una indicación de la presencia o ausencia de un navegador junto al formulario, por ejemplo.

   - parameterid: identificador único de parámetro.
   - structid: identificador de la estructura a la que está asociado el parámetro en cuestión.
   - name: nombre asignado al parámetro.
   - value: valor asignado al parámetro. A partir de este campo el servidor de aplicaciones de WAINE podrá saber que aplicar cuando reciba una petición para ejecutar una opción de un menú.

4. **_struct:** almacena parámetros importantes relativos a cada estructura. Definirá lo que se muestra al seleccionar la opción asociada en un formulario simple o una estructura más compleja como puede ser un formulario con varios niveles de filtrado.

   - structid: identificador único de estructura.
   - type: identificador del tipo de estructura.

5. **_workflow:** almacena parámetros importantes relativos a cada workflow, siendo un workflow un tipo de estructura que permite la gestión de datos de manera secuencial, literalmente se trata de un “flujo de trabajo”.

- workflowid: identificador único de workflow.
- structid: identificador de la estructura que lleva asociado cada paso del workflow.
- msg: mensaje a mostrar en el paso correspondiente del proceso.
- previd: identificador del paso anterior. Se corresponde con el workflowid del paso previo.
- prevmsg: mensaje que se muestra como enlace al paso previo.
- nextid: identificador del paso siguiente dentro del proceso secuencial.
- nextmsg: mensaje que se muestra como enlace al paso siguiente.
- altid: identificador del paso actual.
- altmsg: mensaje que se muestra como enlace al paso actual.

- Tablas relativas a los formularios de consulta

1. **_form:** contiene los datos relativos a los formularios, como por ejemplo de donde tomar los datos y por qué campo ordenarlos.

   - formid: identificador único de formulario.
   - source: fuente de datos del formulario. Por lo general se corresponderá con el nombre de una tabla de la base de datos. aunque puede hacer referencia a otras posibilidades, como una tabla de la MDB, por ejemplo para la gestión de usuarios se suelen hacer referencias a las tablas _user y _group.
   - caption: mensaje mostrado cuando se llama a este formulario. Por defecto, aparecerá como título en la tabla que contenga al mismo.
   - filter: indica la regla de filtrado que debe aplicarse al realizar consultas con este formulario.
   - orderby: campo del formulario por el que se ordenaran los resultados obtenidos en las consultas.
   - theme: nombre del fichero donde está escrito el tema propio de este formulario. No es obligatorio, ya que existe un tema por defecto.

2. **_field:** guarda los parámetros de los distintos campos de un formulario.

   - fieldid: identificador único de campo.
   - formid: identificador del formulario al que está asociado el campo.

- ord: indica el orden de aparición del campo de entre todos los campos del formulario.
- caption: nombre a mostrar en la vista del formulario asociado al campo.
- type: indica el tipo de campo del que se trata, pudiendo tomar los valores string, int, text, etc.
- source: identifica el nombre del campo de la tabla al que se asocia el campo en cuestión del formulario.
- len: longitud del cuadro que se abre al mostrar el formulario para poder modificar los datos.
- maxlen: longitud máxima que se permite al campo en cuestión.
- attr: atributo del campo en cuestión. Podrá indicar si el campo es oculto (H), de sólo lectura (R), etc.
- search: junto al siguiente campo permiten asignar valores de un campo de otra tabla a este campo. Search guarda el nombre de la tabla a consultar.
- searchfld: indica que campo de la tabla anterior es el que debe consultarse. Para mostrarlos todos los posibles valores que puede tomar se utiliza un combo.
- canbenull: indica si el campo puede tomar el valor NULL.
- picture: cadena que marca como debe ser el formato de los datos introducidos de acuerdo a una serie de normas.
- defvalue: valor por defecto para el caso en el que no se rellene el campo.
- msg: mensaje de error mostrado en caso de no haber rellenado un campo que no pueda ser nulo o que no se cumpla el formato enriquecido.

3. **_action:** almacena valores y parámetros necesarios para posibles acciones, entendiéndose éstas como acciones asociadas a botones.

   - actionid: identificador único de acción.
   - formid: identificador del formulario al que está asociada la acción.
   - ord: indica el orden de aparición de la acción de entre todas las acciones del formulario.
   - type: indica el tipo de acción de la que se trata, a saber, web, abre una ventana nueva, system, ejecuta un comando del sistema, o execute, ejecuta una sentencia sql.

- caption: texto que se mostrará en el botón asociado a la acción.
- msg: mensaje de error mostrado en caso de no haber rellenado un campo que no pueda ser nulo o que no cumpla el formato enriquecido.

4. **_acl:** guarda parámetros relativos a la ejecución de los formularios, de forma que un mismo formulario sea visto de manera diferente por distintos usuarios.

   - uid: identificador del usuario al que se aplicarán las reglas descritas en este apartado del código.
   - gid: identificador del grupo al que se aplicarán las reglas descritas en este apartado del código.
   - formid: identificador del formulario al que va asociado esta regla.
   - read: indica si el usuario tiene permiso de lectura.
   - ins: indica si el usuario tiene permiso de inserción de registros nuevos en la tabla asociada al formulario.
   - upd: indica si el usuario tiene permiso para modificar registros.
   - del: indica si el usuario tiene permiso para borrar registros.
   - action: indica si el usuario tiene permiso para ejecutar acciones.

3.4.- Características y funcionalidades

En este apartado se comentarán las características y funcionalidades más importantes de WAINE:

3.4.1.- Generales

- Aplicaciones 3 capas
- Independencia de la plataforma servidor
- Independencia del navegador
- Independencia del origen de datos
- Independencia de la meta-base de datos usada
- Independencia del diseño de interfaces / lógica de aplicación
- Desarrollo rápido de las aplicaciones
- Fuerte parametrización
- Independencia de la plataforma cliente

- Acceso a múltiples fuentes e datos: bases de datos, ficheros,…
- Capa de fuentes de datos definibles por el usuario
- Separación del diseño de interfaces / lógica de la aplicación
- Enlace con otras aplicaciones / páginas
- Ayuda contextual

3.4.2.- Formularios e informes

- Formularios predefinidos: ficha, tabla, estadísticas, filtros…
- Formularios definidos por el usuario
- Temas para formularios y menús
- Campos calculados
- Cálculos acumulados: totales, medias, máximos, mínimos y definidos por el usuario
- Widgets definidos por el usuario
- Acciones definidas por el usuario
- Impresiones y exportación en formatos PS, PDF, XML, XLS…
- Soporte para workflows / wizards

3.4.3.- Seguridad

- Autenticación de usuarios a través de base de datos, LDAP, PAM…
- Restricción de acceso por dirección IP
- Menús personalizados para usuarios / grupos
- ACL’s sobre formularios y reports
- Auditorías sobre usuarios: accesos, acciones,…

3.5.- Desarrollo en WAINE

Previamente a la descripción del archivo XML que contendrá el código necesario para la creación de nuestra aplicación llevaremos a cabo un análisis sobre como trabajar con WAINE.

El primer paso es decidir donde guardar los datos relativos a la aplicación en si, es decir, hemos de crear la base de datos. Para ello realizaremos un diagrama entidad-relación, ERD, en el que se mostrará la estructura de la base de datos que creemos, tablas y relaciones entre las mismas.

Una vez solucionado el problema del almacenamiento de la información hemos de analizar que tipos de usuarios queremos que accedan a nuestra aplicación, que información se mostrara a cada grupo de usuarios, es decir, definición de los distintos menús, los campos que contendrán los distintos formularios, que campos se verán y cuales estarán ocultos…

Una vez analizado a fondo el problema y diseñada como será la aplicación se comenzará a escribir el archivo XML a partir del cual se generará la dicha aplicación. Dicho archivo tendrá extensión .asl, siendo ASL un lenguaje XML definido mediante su DTD.

Tras escribir el archivo XML que contendrá la aplicación se aplicará sobre este un script, asl2mdb, que generará la MDB de nuestra aplicación.

Finalmente WAINE nos permite cambiar el aspecto de la aplicación a nuestro gusto. Para ello existe una serie de archivos de extensión .cfg que contienen una serie de variables que definirán el diseño gráfico de la aplicación que creemos: colores, tipo de letra, alineación, aspecto del menú,…

Con el fin de que el lector pueda entender el código de la aplicación que se muestra en posteriores capítulos, a continuación se comentan las instrucciones más importantes a la hora de desarrollar aplicaciones con WAINE.

3.5.1.- Instrucciones ASL

3.5.1.1.- Grupos y usuarios

Para la creación de los diferentes grupos de usuarios y usuarios de los que constará la aplicación se usan las siguientes instrucciones:

```
<group gid="0" name="demo">
      <user  uid="0" name="demo"  passwd="demo" mainid="main1"
         descr="Single user"/>
</group>
```

Con el código mostrado en el cuadro superior se crea un grupo de usuarios cuyo nombre es demo y cuyo identificador es 0. Entre las etiquetas group se definirán todos

los usuarios que inicialmente pertenecerán a nuestra aplicación y formarán parte del grupo indicado, en nuestro caso pertenecerán al grupo demo.

La definición de los usuarios se llevará a cabo mediante la etiqueta user. El parámetro uid de user se usará para indicar el identificador del usuario correspondiente, que como antes dijimos deberá se único. A continuación se indicarán el nombre y el password del usuario mediante los parámetros name y passwd, y el menú de usuario que se le asociará mediante el parámetro mainid. Finalmente se puede añadir una pequeña descripción mediante el parámetro descr.

3.5.1.2.- Menús

En este apartado veremos como crear los diferentes menús y submenús para los distintos usuarios del sistema.

```
<main id="main1">

       <menu caption="Proyectos" img="usr.jpg">
           <option caption="Project" call="st_project"/>
           <option caption="Project Config" call="st_project"/>
       </menu>

      <menu caption="Proceso">
           <option caption="Process" type="form">
               <param name="formid" value="frm_process"/>
               <param name="navigator_position" value="W"/>
               <param name="navigator_fields" value="1"/>
               <param name="navigator_width" value="3"/>
               <param name="button_data" value="1"/>
           </option>
           <option caption="Activity" type="relation">
               <param name="form_split" value="rows=100,*"/>

               <param name="formid" value="frm_process"/>
                  ...
            </option>
            ...
      </menu>
      ...
</main>
```

Entre las etiquetas main se coloca el menú correspondiente al grupo de usuarios que pertenezcan a ese main. Cada grupo de usuarios tendrá su correspondiente main con sus elementos correspondientes.

Una vez definido el main, se indica entre las etiquetas menu los submenús principales y a su vez dentro de estos, las etiquetas definen el contenido de estos submenús.

En la etiqueta main se indican el identificador del menú y el nombre del mismo mediante los parámetros id y caption.

En la etiqueta menu se indican el nombre del submenú que se mostrará en el menú y que agrupará los elementos que se indiquen dentro de esta etiqueta mediante los elementos option.

Los elementos option presentan dos formatos posibles. En ambos casos con la etiqueta caption se indica el nombre que se mostrará en el submenú y a través del cual podremos acceder a la dicha opción. Si el elemento option presenta el parámetro call, entonces dicho parámetro contendrá el valor del struct que indicará el formulario a mostrar y el elemento option será un elemento vacío. En caso contrario el campo type indicara el tipo de formulario que se presenta en esta opción, y el elemento option contendrá elementos param que determinarán las características del formulario. Algunos de los elementos param más importantes son:

- **formid:** indica el nombre del formulario que se mostrará.
- **form_type:** indica el tipo de formulario que se mostrará, sus valores pueden ser form, table, list…
- **button_xxxx:** indica la presencia o no de un determinado botón o conjunto de botones.
- **Navigator_position:** indica la presencia del navegador y la zona de la pantalla en la cual se mostrará el mismo.
- **Navigator_field:** indica el campo que se mostrara en el navegador, y a través del cual accederemos a los datos del formulario.
- **source_filter_field:** indica el campo por el cual se filtra el formulario.
- **source_filter_where:** se usa para filtrar el formulario mediante instrucciones SQL.

En la siguiente figura se muestra un ejemplo de un menú creado mediante WAINE.

Figura 5.- Menú de ejemplo

3.5.1.3.- Formularios

Mediante la etiqueta form podemos crear los diversos formularios que formarán parte de nuestra aplicación. A continuación, se muestra un ejemplo de cómo desarrollar un formulario que nos permitirá explicar sus diversos campos.

```
<form id="frm_process" source="Process" caption="Process">
        <orderby>pk</orderby>
        <fields>
           <key source="pk"/>
          <string source="name" caption="Name" len="22"
                  maxlen="255"/>
           <text source="descr" caption="Description">
              <width>22</width>
              <height>2</height>
           </text>
        </fields>
    </form>
```

En la etiqueta form se indican el identificador del formulario, a través del cual accederemos a dicho formulario, mediante la etiqueta id, la fuente de datos, o tabla de la base de datos de la cual se coge la información para este formulario, mediante la etiqueta source, y el nombre del formulario, que se indicará mediante el campo caption y que será el texto que se mostrará al sacar por pantalla el formulario.

En el interior del elemento form se recogerán otra serie de elementos que determinaran algunos aspectos de la representación y el tipo de los datos.

Mediante el elemento orderby se indica a partir de que campo se quiere que se ordenen los datos de la tabla para mostrarlos en el formulario. Entre los elementos fields se indican los datos que formarán parte del formulario, indicando para cada uno de estos de que tipo es, cual es su fuente, es decir, de que campo de la tabla toman su valor y con que nombre se mostrarán en el formulario, además de otros datos que serán propios de cada tipo de dato. El elemento key indica cual es la clave primaria de la tabla a la que accede el formulario y es obligatorio.

Algunos de los parámetros más importantes a indicar en los elementos del formulario son:

- source: indica el nombre del atributo de la tabla al que hace referencia ese campo.
- caption: sirve para indicar el nombre que se mostrara para identificar al campo.
- type: se usa para definir el tipo del campo: int, string, text… en caso de que definamos el elemento como field, siendo posible definir el tipo del campo directamente mediante elementos de un tipo (string, text,… vease el código anterior).

- len: campo que controla la anchura del campo del formulario.
- maxlen: indica el número máximo de caracteres que se pueden introducir en ese campo.
- canbenull: este parámetro determina si el campo al que se refiere puede tomar o no un valor nulo, y su valor puede ser Y o N.
- msg: se usa para indicar el mensaje que se mostrará en caso de error al introducir el valor del campo.

A continuación se puede observar un formulario creado mediante el código anterior.

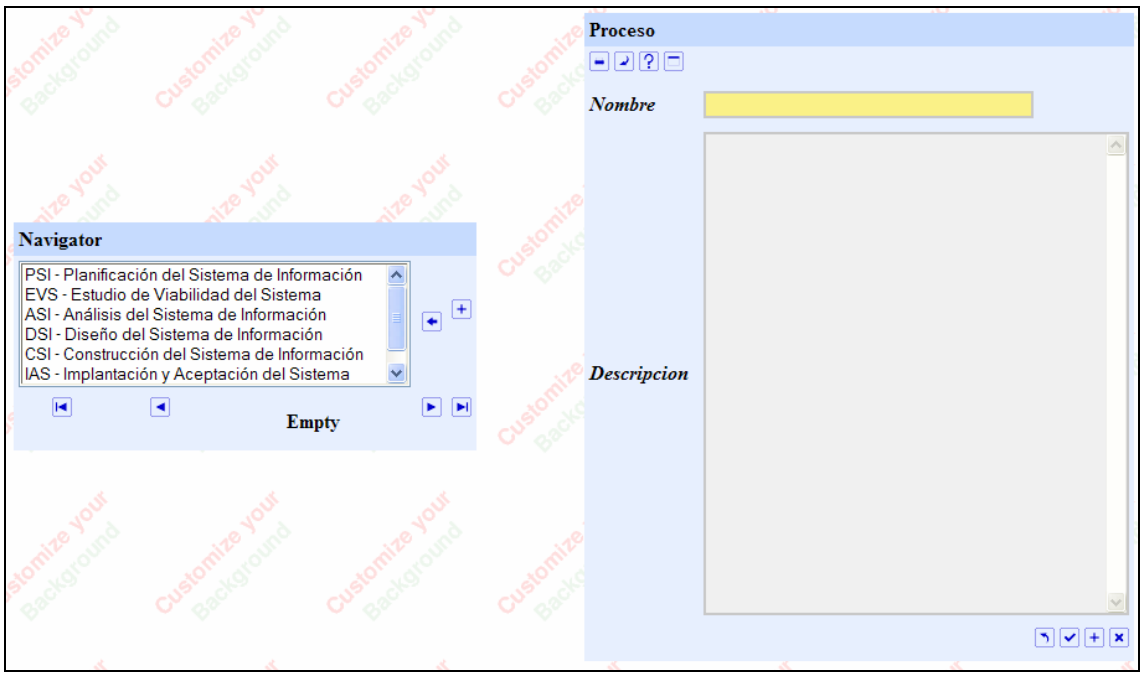

Figura 6.- Formulario de ejemplo

3.5.1.4.- Struct

Mediante el elemento struct se pueden crear estructuras más complejas, permitiendo dividir la pantalla en varias partes. Para ello, el struct contiene en su interior una serie de elementos param, los cuales definen las características del la estructura que formaremos. Un struct puede contener a otro struct pudiendo crear así estructuras más complejas aun.

El elemento struct contiene la etiqueta id en la que se identificador del struct y la etiqueta type en la que se indica el tipo del struct. Los tres tipos de struct más útiles a la hora de realizar una aplicación son:

- Relation: este es el más usado. Se usa para establecer una relación entre los distintos formularios o estructuras que componen el struct.

- Tabs: este tipo de struct permite tener varios formularios en hojas distintas de forma que se nos muestran varias pestañas con los nombres de cada hoja. Según la hoja que queramos ver, el formulario que queramos ver, pincharemos sobre la pestaña con el nombre correspondiente, y se nos mostrará el formulario adecuado.

- Split: este tipo de struct divide la pantalla en tantas partes como queramos, de forma que en cada parte pondremos un formulario u otro struct. En este caso los diversos formularios no están relacionados

En la siguiente figura podemos observar un ejemplo de cada uno de los tipos indicados.

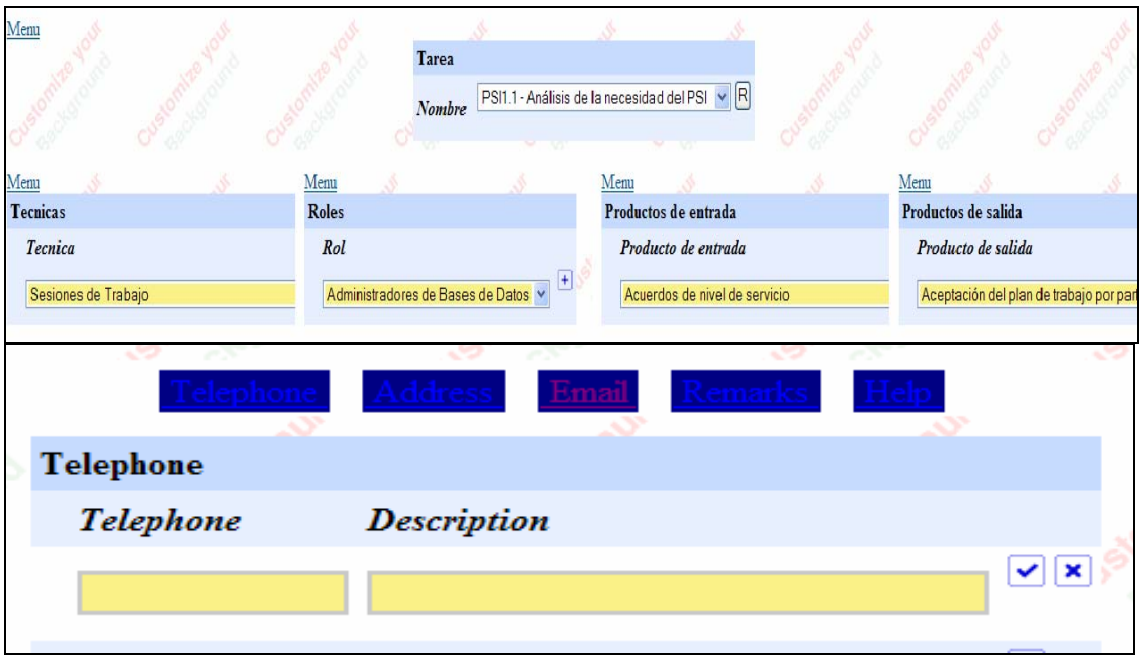

Figura 7.- Ejemplos de struct. En la figura superior un relation con un formulario y un struct tipo split. En la figura inferior un tipo tab

Los elementos param de los struct son similares a los de los option. A continuación se muestrán los valores más usados en los struct, sin incluir los ya mencionados en el elemento option.

- **form_split:** mediante este param se indica la subdivisión que se hará de la pantalla. Esta subdivisión se pude indicar en filas o en columnas, y en número de una de estas o en tanto por ciento (véase el código de ejemplo siguiente).

- **tab_name:** en el caso de un struct tipo tab cada una de las partes del struct tiene que llevar un param de este tipo en cuyo valor se indica el

nombre que queremos que aparezca en las pestañas mediante las cuales seleccionaremos que parte del struct queremos ver.

- **structid:** se usa para indicar el identificador de un struct que forma parte de otro struct.

```
<struct id="st_listatecnicasdos" type="relation">
            <param name="form_split" value="rows=40%,*"/>
            <param name="formid" value="frm_task"/>
            <param name="form_type" value="combo"/>
            <param name="fields_hidden" value="2:3"/>
            <param name="source_filter_field" value="fkact"/>

            <param ord="2" name="structid"
                              value="st_listatecnicastres"/>
</struct>

<struct id="st_listatecnicastres" type="split">
            <param name="form_split" value="cols=25%,25%,25%,*"/>

            <param ord="1" name="formid" value="form_tastec"/>
            <param ord="1" name="form_type" value="list"/>
            <param ord="1" name="fields_hidden" value="1:2:3:4"/>
            <param ord="1" name="source_filter_field" value="pk"/>
            <param ord="1" name="nofill" value="1"/>

            <param ord="2" name="formid" value="form_tasrol"/>
            <param ord="2" name="form_type" value="list"/>
            <param ord="2" name="fields_hidden" value="1:2:3:4"/>
            <param ord="2" name="source_filter_field" value="pk"/>
            <param ord="2" name="nofill" value="1"/>

            <param ord="3" name="formid" value="form_tasip"/>
            <param ord="3" name="form_type" value="list"/>
            <param ord="3" name="fields_hidden" value="1:2:3:4"/>
            <param ord="3" name="source_filter_field" value="pk"/>
            <param ord="3" name="nofill" value="1"/>

            <param ord="4" name="formid" value="form_tasop"/>
            <param ord="4" name="form_type" value="list"/>
            <param ord="4" name="fields_hidden" value="1:2:3:4"/>
            <param ord="4" name="source_filter_field" value="pk"/>
            <param ord="4" name="nofill" value="1"/>
</struct>
```

3.5.1.5.- Workflow

Un workflow es una sucesión de pantallas enlazadas cada una de ellas con la pantalla que le precede y con la posterior, pudiendo enlazar cada pantalla con un formulario, una estructura, una página web…

La llamada al workflow se hace mediante un conjunto de instrucciones como el que se muestra a continuación, añadiendo dicho conjunto de instrucciones en el submenú que queramos enlazar con el workflow.

```
<option caption="WorkFlow" type="workflow">

      <param name="workflowid" value="register-user-1"/>

</option>
```

Como vemos se indicará en el type que se trata de un workflow, variante que no vimos antes, y posteriormente se indicará de que workflow se trata mediante el param workflowid.

A continuación se definirán las distintas pantallas que componen el workflow mediante un conjunto de instrucciones similar al que se muestra a continuación.

```
<workflow id="register-user-1" url="pagina1.html">
      <msg>Paso 1</msg>

      <next id="register-user-2" msg="Paso 2"/>

</workflow>
```

Para la segunda pantalla además de la etiqueta next se usará la etiqueta prev para que enlace con la pantalla anterior y alt para que haga una llamada a la pantalla 2 (aunque podría hacer referencia a cualquier otra pantalla del workflow).

```
<workflow id="register-user-2" strucid="st_strucid">
      <msg>Paso 2</msg>

      <prev id="register-user-1" msg="Paso 1"/>
      <alt id="register-user-2" msg="Again"/>
      <next id="register-user-3" msg="Paso 3"/>

</workflow>
```

Podríamos tener de este modo muchas más pantallas. Finalmente, la última pantalla del workflow se define como las anteriores pero sin especificar la pantalla que le sigue, puesto que es la última.

```
<workflow id="register-user-3" strucid="end.html">
      <msg>Paso 3</msg>

      <prev id="register-user-2" msg="Paso 2"/>

</workflow>
```

3.5.1.6.- Actions

Entre las etiquetas buttons se colocan los distintos botones que se vayan a definir. A su vez dentro de estas, mediante etiquetas action, se especificarán las distintas acciones.

Existen distintos tipos de acciones:

- System: ejecuta un comando del sistema
- Web: ejecuta una llamada a una pagina web
- Execute: ejecuta un comando de la base de datos

El siguiente fragmento de código muestra como insertar un action para ejecutar un comando del sistema. Es entre las etiquetas code donde se colocan los comandos a ejecutar o la pagina web que se mostrará.

```
<buttons>

      <action type="system" caption="System command">
            <tooltip>Ejecutar comando del sistema</tooltip>
            <code>echo Action exec %username: %date %hour %hostip >>
                  ./login.log</code>
            <msg>¿Está usted seguro?</msg>
      </action>

</buttons>
```

3.5.1.7.- Image-file

Se trata de un campo de un formulario mediante el cual se muestra un archivo de imagen. A continuación se muestra un ejemplo del código.

```
<image-file source="photo" caption="image">
      <width>100</width>
</image-file>
```

Mediante la etiqueta width se indica la anchura del recuadro que contendrá la imagen.

Figura 8.- Ejemplo de image-file

3.5.1.7.- Blob-file

Se trata de un campo de un formulario mediante el cual se almacenará un fichero. Esto se consigue mediante la siguiente instrucción.

```
<blob-file source="ejemplo" caption="Ejemplo">
```

En la siguiente figura podemos observar un ejemplo de este tipo de datos.

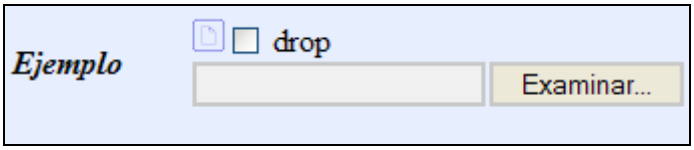

Figura 9.- Ejemplo de blob-file