Sistemas Informáticos
2004-2005

Gestor Global de Contactos

José Julián Pérez Gutiérrez
Carlos Alberto Romero Díaz
Andrés Sánchez Colmenar

Dirigido por:
Prof. Ismael Rodríguez Laguna
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Índice general

Otros índices

Índice de tablas

Índice de listados

Índice de ilustraciones

CAPÍTULO 1. Resumen

1.1. Resumen en español

El Gestor Global de Contactos ha sido planteado como un sistema electrónico capaz de proporcionar un servicio similar a una agenda de contactos, el cual registrará los datos de los contactos del usuario: datos personales, teléfonos, direcciones, direcciones de correo electrónico, y cualquier otro tipo de dato que se precisase.

Sin embargo, este servicio ofrece una serie de características que se traducen en ventajas para el usuario:

- Es un sistema en el que el que la información de los contactos no es introducida por cada usuario, sino en la que cada usuario introduce sus propios datos que luego pueden ser accedidos por sus contactos.
- Los usuarios del sistema acceden a los datos de sus contactos mediante un sistema de perfiles. Cada usuario puede definir en sus perfiles qué datos del usuario son accesibles a través de dicho perfil, y además se pude asignar una contraseña de forma que un usuario solo pueda obtener el permiso de un perfil si conoce dicha contraseña.
- El sistema se basa en una red de proveedores, intercomunicados por un protocolo creado específicamente para este sistema. Las características de este sistema son:
    - Permitir que cada entidad que desee crear un proveedor pueda usar las tecnologías que considere más adecuadas. Simplemente deberá implementar un módulo capaz de de comunicarse mediante este protocolo para poder unirse a la red global de proveedores.
    - Los proveedores se identifican mediante direcciones de dominios o por direcciones IP. Cada empresa, país, organización, etc. Puede crear su propio proveedor y tener un nombre corporativo.
    - Cada proveedor puede dar un servicio personalizado a sus usuarios, nuevas funcionalidades, locales al proveedor, no contempladas por el protocolo, como por ejemplo, el almacenamiento de archivos, calendario, etc. A su vez cada proveedor puede ofrecer los puntos de acceso que desee a sus usuarios.
    - Al estar formado por múltiples servidores, el sistema es más robusto, pues no tiene ningún núcleo del que dependa el resto, si un proveedor tiene una parada del servicio solo afectará a los usuarios de ese proveedor.
    - Este diseño en red también facilita la expansión del sistema, pudiendo añadirse continuamente proveedores por una red global, por ejemplo, Internet.
- La implementación de diversos puntos de acceso por parte del proveedor permite a sus usuarios acceder fácilmente desde cualquier sitio a sus datos. Posibles puntos de acceso pueden ser: una página web, un teléfono móvil, una agenda electrónica o una aplicación desde terminal, al poder decidir cada proveedor como implementará el servicio, podrá crear los puntos de acceso que considere más útiles para sus usuarios.

Por el momento, la única desventaja relevante de este sistema radica en su propia globalidad, un usuario solo puede acceder a los datos de usuarios de este sistema, por lo que es necesario que toda persona a la que se quiera acceder a sus datos sea usuario del servicio.

1.2. Resumen en inglés

Global Contact Manager has been thought like an electronic system able to provide a service that looks like a contact agenda. All the contact's information (personal information, telephone numbers, addresses, emails, and whatever data type that the users require) will be registered by this system.

But, this service offers some features that might become important advantages to users:

- The user does not have to worry about keeping contacts' information up to date. Each user is meant to keep his own data updated in order to allow access to it to other users (users who, from that point, will have him as a contact).
- System users will be able to access contacts information by a profiles system which let those contacts to define what information will share to others in each profile, and it will let them to keep his information safe by assigning a password to each profile they create. One user will only be able to get access to a profile if he knows its password.
- The system is based on a dealers' network, interconnected by a protocol that has been created specifically for this system. Below there are some of the most important system features:
    - Whatever entity that wants to create a system server, will be able to use the most suitable technologies. The only condition they have implementing their own server is to meet the requirements the protocol impose to every dealer in order to join the global dealers network.
    - The dealers are identified by a domain name or an IP addresses. Each company, country, organization, etc. will be able to create its own dealer with its corporative name as long as they own the domain name.
    - Each dealer can offer a customizable service to its users or add new functionalities only supported by that dealer, like for example, file storing, scheduling, etc. At the same time, each dealer can offer the desired access points.
    - Being a shared network, built of more than one server, makes the system much more reliable, even because the main functionality is provided by each server so they can perfectly stand alone. So, if one dealer has broken down only users of that particular dealer will be affected.
    - This network design makes easier the system's expansion because a new dealer will always be able to connect to the system by a global network like Internet.
- The implementation of several access points by each dealer let its users to get access to their data from nearly every place. There are several chances of access points: web page, cellular phone, electronic agenda, pocket pc or terminal application. Each dealer will have to decide which access points let them offer to their clients a better service and they will only need to implement those, because that is a dealer's choice.

Actually, the only noticeable disadvantage this system has is because of its own global aim: one user will only be able to access to the information of another user who already had a system account so, if they want to get access to any person's data, that person must have an account in any of the GCM dealers.

CAPÍTULO 2. Protocolo de comunicación basado en XML

Desde el principio se plantea la construcción de un protocolo básico de comunicación basado en XML para la comunicación entre proveedores que sirva, de forma eficaz, a la interconexión de muy diversos proveedores constituidos por tecnologías muy diferentes y variadas.

El motivo del diseño de un protocolo unificado viene determinado por la razón del sistema de ser altamente diverso: es necesario el establecimiento de un nexo que pueda unir todos los servidores independientemente de la plataforma en la que hayan sido desarrollados o en la que vayan a ser implementados. Por este motivo, la transmisión de mensajes en texto plano supone el mejor acercamiento y debido a las facilidades que aporta la utilización del XML, se optó por este metalenguaje para la construcción de nuestros mensajes.

Las principales características del protocolo definido son:

- Basado en mensajes (que pueden ser consultas o respuestas)
- Definido mediante una DTD que permite comprobar la validez de cualquiera de los mensajes integrantes del protocolo
- Simplicidad y legibilidad de los mensajes al tratarse de texto XML que puede ser *parseado* por cualquier programa de tratamiento de XML
- UTF-8 como codificación básica para permitir la globalización del sistema

2.1. Modo de conexión inter-servidor

La conexión entre los servidores se realiza mediante *sockets* TCP-IP, todas las comunicaciones se realizarán por el puerto 4444 que debe estar dedicado exclusivamente a su uso para la comunicación con otros proveedores.

El procedimiento de comunicación se basa en conexiones unidireccionales que se cierran en cuanto un mensaje ha sido transmitido, el procedimiento es el siguiente:

1. El servidor que sedea enviar un mensaje a otro proveedor, a partir de ahora denominados servidor A y servidor B respectivamente, crea un socket que abre sobre el puerto 4444 en la dirección IP obtenida mediante el servicio DNS usando el nombre del servidor B.
2. El servidor B ha de recibir la petición de conexión en un socket de servidor que ha de tener permanentemente escuchando sobre el puerto 4444, una vez obtenida la petición ha de abrir un socket para establecer la comunicación TCP-IP.
3. El servidor A envía una cadena en formato UTF8 con el mensaje del protocolo en formato XML, a continuación se cierra el socket.
4. El servidor B ha de esperar a recibir el mensaje del protocolo sobre el socket que ha abierto, una vez leído el mensaje cerrará el socket y se dará como finalizada la conexión.

2.2. Estructura y gramática (DTD) de los mensajes

El protocolo se diseño como protocolo basado en mensajes que pueden ser consultas a un proveedor, o respuestas a una consulta. La gramática que define la estructura de los mensajes es la que aparece en el *Listado 1*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT mensaje (svdL, svdR, idTarea, dts)>
<!ATTLIST mensaje
      vMsg CDATA #REQUIRED
      vSvd CDATA #REQUIRED>
<!ELEMENT svdL (#PCDATA)>
<!ELEMENT svdR (#PCDATA)>
<!ELEMENT idTarea (#PCDATA)>

<!-- CAMPO DATOS -->
<!ELEMENT dts (CDU | RDU | CNC | RNC | CBC | RBC | MERR)>
```

Listado 1. Gramática de un mensaje del protocolo

En la gramática del Listado 1 aparecen los siguientes elementos (véase *Tipos de datos* para una explicación detallada de cada tipo):

- `mensaje`: es el elemento raíz de de los mensajes del protocolo. Tiene dos atributos que son la versión del mensaje (`vMsg`) y la versión del servidor (`vSvd`), estas dos versiones sirven para conocer la versión del protocolo en que está escrito el mensaje y la última versión del protocolo admitida por el servidor.
- `svdL`: se trata del servidor local, de aquel que envía el mensaje. Tipo de dato: *Nombre de servidor*.
- `svdR`: se trata del servidor remoto, de aquel que recibirá el mensaje. Al igual que en el caso anterior, ha de ser del tipo *Nombre de servidor*.
- `idTarea`: se trata del identificador que permite identificar qué respuestas corresponden a cada pregunta puesto que coincidirán cuando sean iguales. Este identificador debe ser global a toda la red de servidores por lo que se construye añadiendo tras el nombre del servidor un identificador único en el dominio del proveedor del que proviene tras el carácter '#'. Ej.: 192.168.1.4#18. Tipo de dato: *Identificador de tarea.*
- `dts`: es el campo que contiene la información propia de la consulta o la respuesta. Sólo puede contener un tipo de dato de mensaje, que identifica de forma inequívoca de qué consulta o de qué respuesta se trata.

El protocolo define los siguientes mensajes:

Consultas (*Tipo*)	**Respuestas (*Tipo*)**
Consulta de datos de usuario (*CDU*)	Respuesta de datos de usuario (*RDU*)
Añadir un nuevo contacto (*CNC*)	Respuesta de añadir un nuevo contacto (*RNC*)
Borrar un contacto (*CBC*)	Respuesta de borrar un contacto (*RBC*)
	Error (*MERR*)

Tabla 1. Tipos de mensaje

2.3. Estructura de los datos (campo dts)

A continuación, y a modo de referencia, se expone la estructura de todos los mensajes así como de la información necesaria para rellenarlo de forma correcta. El esquema que se sigue para cada mensaje es el siguiente:

1. Definición del mensaje: utilidad y modo de empleo
2. Gramática del mensaje
3. Campos que lo componen
4. Tipos de datos de los que hace uso

2.3.1. Consulta de datos de usuario (CDU)

Este mensaje tiene la finalidad de pedir los datos a los que se tiene acceso de un usuario. Es necesario siempre que se desee conocer la información de otro usuario del sistema y que es un contacto nuestro.

```
<!ELEMENT CDU (idR, idD)>
      <!ELEMENT idR (#PCDATA)>
      <!ELEMENT idD (#PCDATA)>
```

Listado 2. Gramática de la consulta de datos de usuario

En esta consulta los campos a rellenar los siguientes:

- `idR`: se trata del usuario que remite la consulta, aquél que quiere conocer los datos de un contacto. Tipo de dato: *Nombre de usuario*.
- `idD`: se trata del usuario del que se desean conocer sus datos. Tipo de dato: *Nombre de usuario*.

2.3.2. Consulta para añadir un nuevo contacto (CNC)

Este mensaje tiene la finalidad de añadir a nuestra lista de contactos uno nuevo. Mediante esta consulta se realiza la petición al servidor propietario de la cuenta del contacto para que compruebe nuestras credenciales de acceso a los datos de un perfil del usuario.

```
<!ELEMENT CNC (idR, idD, perfil, pass)>
      <!ELEMENT idR (#PCDATA)>
      <!ELEMENT idD (#PCDATA)>
      <!ELEMENT perfil (#PCDATA)>
      <!ELEMENT pass (#PCDATA)>
```

Listado 3. Gramática de la consulta para añadir nuevos contactos

En esta consulta los campos a rellenar son los siguientes:

- `idR`: usuario que remite la consulta. Es el usuario que quiere poder acceder a los datos de otro usuario. Tipo de dato: *Nombre de usuario*.
- `idD`: usuario destino de la consulta. Es el contacto, el usuario propietario del perfil al que se quiere poder acceder. Tipo de dato: *Nombre de usuario*.
- `perfil`: perfil al que se quiere tener acceso. Tipo de dato: *Identificador*.

- `pass`: contraseña del perfil al que se quiere tener acceso. Tipo de dato: *Identificador*.

2.3.3. Consulta para borrar un contacto (CBC)

Este mensaje tiene como finalidad comunicar la intención del usuario de no acceder más a los datos de un determinado contacto. Mediante este mensaje se avisa al contacto de las intenciones del usuario, de esta forma, éste sabe que a partir de ese momento no debe permitirle el acceso a sus datos.

```
<!ELEMENT CBC (idR, idD)>
      <!ELEMENT idR (#PCDATA)>
      <!ELEMENT idD (#PCDATA)>
```

Listado 4. Gramática de la consulta para borrar contactos

En esta consulta los campos a rellenar son los siguientes:

- idR: usuario que remite la consulta. Es aquél que quiere dejar de acceder a los datos de otro usuario. Tipo de dato: *Nombre de usuario*.
- idD: destinatario de la consulta. Es el contacto del que se desea dejar de acceder a sus datos. Tipo de dato: *Nombre de usuario*.

2.3.4. Respuesta a la consulta de datos de usuario (RDU)

En la respuesta a la consulta de los datos de usuario, se transmiten los datos a los que el usuario tenía acceso. Este mensaje es el más complejo de todos debido a la cantidad de información que puede transmitir, por ello, y para evitar la sobrecarga en el mensaje, casi todos sus campos son opcionales, es decir, pueden no aparecer cuando no se tiene acceso a dichos datos o el campo no está relleno. En cualquier caso, es importante hacer notar que el orden de los campos está preestablecido por lo que si dos campos aparecen, deben siempre aparecer en el mismo orden. Esto es igual para todos los mensajes, pero es precisamente en este cuando toma especial relevancia.

```
<!ELEMENT RDU (idD, usr)>
      <!ELEMENT usr (idUsr, dtPerso)>
      <!ELEMENT idUsr (#PCDATA)>
      <!ELEMENT dtPerso (     usrNombre?,
                              usrApellidos?,
                              usrNombreIngles?,
                              usrApodo?,
                              usrLugarNacimiento?,
                              usrFechaNacimiento?,
                              usrNacionalidad?,
                              usrIdiomas?,
                              usrDirs?,
                              usrTfns?
                              usrMails?,
                              usrCmpPerso?)>
            <!ELEMENT usrNombre (#PCDATA)>
            <!ELEMENT usrApellidos (#PCDATA)>
            <!ELEMENT usrNombreIngles (#PCDATA)>
            <!ELEMENT usrApodo (#PCDATA)>
            <!ELEMENT usrLugarNacimiento (#PCDATA)>
            <!ELEMENT usrFechaNacimiento (fd, fm, fa)>
                  <!ELEMENT fd (#PCDATA)>
```

```
                <!ELEMENT fm (#PCDATA)>
                <!ELEMENT fa (#PCDATA)>

          <!ELEMENT usrNacionalidad (#PCDATA)>
          <!ELEMENT usrIdiomas (idioma+)>
                <!ELEMENT idioma (#PCDATA)>

          <!ELEMENT usrDirs (direc+)>
          <!ELEMENT direc ( dirTit,
                            dirTxt,
                            dirCiudad,
                            dirEstado,
                            dirPais,
                            dirCP)>
                <!ELEMENT dirTit (#PCDATA)>
                <!ELEMENT dirTxt (#PCDATA)>
                <!ELEMENT dirCiudad (#PCDATA)>
                <!ELEMENT dirEstado (#PCDATA)>
                <!ELEMENT dirPais (#PCDATA)>
                <!ELEMENT dirCP (#PCDATA)>

          <!ELEMENT usrTfns (tfn+)>
                <!ELEMENT tfn (tTit, tVal)>
                      <!ELEMENT tTit (#PCDATA)>
                      <!ELEMENT tVal (#PCDATA)>

          <!ELEMENT usrMails (mail+)>
                <!ELEMENT mail (tTit, tVal)>

          <!ELEMENT usrCmpPerso (cmp+)>
                <!ELEMENT cmp (cmpN, cmpV)>
                      <!ELEMENT cmpN (#PCDATA)>
                      <!ELEMENT cmpV (#PCDATA)>
```

Listado 5. Gramática de la respuesta de datos de usuario

En esta respuesta, los campos que son obligatorios rellenar son:

- `idD`: es el remitente de la consulta (y destinatario de la respuesta), es decir, es el usuario que solicitó los datos y al que debe serle devuelto el mensaje. Tipo de dato: *Nombre de usuario*.
- `idUsr`: es el identificador del usuario propietario de los datos. Tipo de dato: *Nombre de usuario*.
- `dtPerso`: en este elemento deben aparecer otros con la información del usuario. Puede aparecer vacío cuando no hay ningún dato especificado, o el perfil al que se tiene acceso no permite acceder a ningún campo. Tipo de dato: *Elemento*.

En esta respuesta los posibles campos para rellenar son:

- `usrNombre`: nombre del usuario. Tipo de dato: *Identificador*.
- `usrApellidos`: apellidos del usuario. Tipo de dato: *Identificador*.
- `usrNombreIngles`: nombre en caracteres ingleses del usuario. Tipo de dato: *Identificador*.
- `usrApodo`: apodo (nick) del contacto. Tipo de dato: *Identificador*.

- `usrLugarNacimiento`: lugar de nacimiento del contacto. Tipo de dato: *Identificador*.
- `fd`: día de la fecha. Tipo de dato: *Natural (1:31)*.
- `fm`: mes de la fecha. Tipo de dato: *Natural (1:12)*.
- `fa`: año de la fecha. Tipo de dato: *Natural*.
- `usrNacionalidad`: nacionalidad del usuario. Tipo de dato: *País*.
- `idioma`: idioma que habla el usuario. Tipo de dato: *Idioma*.
- `dirTit`: nombre con el que se conoce a la dirección. Tipo de dato: *Identificador*.
- `dirTxt`: texto de la dirección (valor de la dirección). Tipo de dato: *Identificador*.
- `dirCiudad`: ciudad (población) de la dirección. Tipo de dato: *Identificador*.
- `dirEstado`: estado (provincia) de la dirección. Tipo de dato: *Identificador*.
- `dirPais`: país de la dirección. Tipo de dato: *País*.
- `dirCP`: código postal de la dirección. Tipo de dato: *Identificador*.
- `tTit`: nombre con el que se conoce al teléfono (o al e-mail). Tipo de dato: *Identificador*.
- `tVal`: número de teléfono (o dirección de e-mail). Tipo de dato: *Identificador*.
- `cmpN`: nombre que se le dará al campo personalizable. Tipo de dato: *Identificador*.
- `cmpV`: valor del campo personalizable. Tipo de dato: *Identificador*.

2.3.5. Respuesta a la consulta de añadir un nuevo contacto (RNC)

Esta respuesta transmite si se ha concedido el permiso de acceso, o si no se han proporcionado las credenciales necesarias.

```
<!ELEMENT RNC (idD, res)>
      <!ELEMENT idD (#PCDATA)>
      <!ELEMENT res (#PCDATA)>
```

Listado 6. Gramática de la respuesta a la consulta de añadir contactos

En esta respuesta los campos a rellenar son los siguientes:

- `idD`: remitente de la consulta y destinatario de la respuesta. Tipo de dato: *Nombre de usuario*.
- `res`: resultado de la consulta. El resultado puede ser positivo (se cumplen con las credenciales por lo que se permite el acceso a los datos del perfil) o negativo (no se permitirá el acceso a los datos del perfil). Tipo de dato: *Booleano*.

2.3.6. Respuesta a la consulta de borrar un contacto (RBC)

Esta respuesta comunica al usuario que envió la consulta si se ha conseguido eliminar correctamente el contacto o, por el contrario, si aún lo debe tener en su lista de contactos.

```
<!ELEMENT RBC (idD, res)>
```

```
      <!ELEMENT idD (#PCDATA)>
      <!ELEMENT res (#PCDATA)>
```

Listado 7. Gramática de la respuesta a la consulta de borrar contactos

En esta respuesta los campos a rellenar son los siguientes:

- `idD`: remitente de la consulta y destinatario de la respuesta. Tipo de dato: *Nombre de usuario*.
- `res`: resultado de la consulta. El resultado puede ser positivo (se cumplen con las credenciales por lo que se permite el acceso a los datos del perfil) o negativo (no se permitirá el acceso a los datos del perfil). Tipo de dato: *Booleano*.

2.3.7. Respuesta de error (MERR)

Este mensaje es una respuesta especial. Puede ser la respuesta de cualquiera de las consultas cuando ha ocurrido un error al procesar la consulta. Este mensaje contiene un campo en el que se indica, mediante un código el error que se ha producido de forma que pueda actuarse de la mejor manera posible tras el error.

```
<!ELEMENT MERR (idD, cod)>
      <!ELEMENT idD (#PCDATA)>
      <!ELEMENT cod (#PCDATA)>
```

Listado 8. Gramática de la respuesta de error

En esta respuesta los campos a rellenar son:

- `idD`: remitente de la consulta y destinatario de la respuesta. Tipo de dato: *Nombre de usuario*.
- `cod`: código del error que indica qué error se ha producido. Tipo de dato: *Entero negativo*. Los posibles errores que pueden darse se listan en la *Tabla 2*.

Código	**Error**
-1	Error desconocido.
-50	No ha podido ejecutarse la consulta. Error durante la ejecución.
-51	No se ha podido conectar con la base de datos.
-101	Identificador del usuario erróneo.
-102	El mensaje no tiene como destinatario este proveedor.
-104	El usuario no tiene permiso de acceso a los datos.

Tabla 2. Códigos de error

2.4. Tipos de datos

Los tipos de datos que pueden contener los mensajes del estándar son los siguientes (ver la descripción de los mensajes para conocer qué campo debe tener qué tipos):

- **Identificador**: un identificador es una cadena de caracteres cualesquiera. Debe tenerse en cuenta el peligro de que en un XML se escriba como contenido de este identificador, elementos propios del XML que pudieran hacer variar el significado del mensaje; por ello, los siguientes caracteres deben ser convertidos de la siguiente forma:

<	<	Menor que
>	>	Mayor que
&	&	Ampersand
'	'	Apóstrofo
"	"	Dobles comillas

Tabla 3. Conversión de caracteres en el protocolo XML

De esta forma se consigue que el XML no se corrompa independientemente del contenido del identificador.

- **Nombre de servidor**: los nombres de servidor se corresponden con el dominio o la dirección de IP que los identifica en Internet. Así, si el servidor es accesible a través de la dirección IP 192.168.1.4, el nombre del servidor será *192.168.1.4*.
- **Nombre de usuario**: los nombres de usuario se componen de un identificador de usuario (identificador que no contiene el carácter '#') seguido de una '#' y seguido de la dirección del proveedor (ya sea un nombre de dominio o una dirección IP). Ej.: *carlos#192.168.1.4*.
- **Natural**: debe ser un número entero positivo.
- **Natural (i-j)**: debe ser un número entero positivo comprendido entre i y j. ($n \in \aleph : i \leq n \leq j$).
- **Entero negativo**: debe ser un número entero negativo.
- **Booleano**: su valor puede ser la cadena `true` o la cadena `false`.
- **País**: se trata de un código de tres letras mayúsculas que corresponde al ISO 3166-3. (Véase *Apéndice B. Tabla ISO 3166-3*).
- **Idioma**: se trata de un código de dos letras mayúsculas que se corresponde con el ISO 639-2. (Véase *Apéndice C. Tabla ISO 639-2*).

CAPÍTULO 3. Casos de uso y funcionalidad mínima

Entre los casos de uso que todo proveedor debe implementar pueden distinguirse dos tipos: aquellos que hacen uso de la comunicación inter-proveedor y aquellos internos a un mismo proveedor.

Los primeros son estrictamente obligatorios de implementar y su funcionalidad debe restringirse a aquella que el protocolo XML determine puesto afectan no sólo al proveedor propio, sino a lo ajenos. Los de segundo tipo, ofrecen mucha mayor libertad y la existencia o no de estos, u otros, casos de uso es de elección del equipo de implementación y diseño del proveedor dependiendo de la funcionalidad final que se desee aporte dicho proveedor. Por esta razón, aquí sólo se aportan casos de uso ideales y la funcionalidad que deberían tener para permitir construir un proveedor funcional aunque sin características avanzadas. En capítulos sucesivos se aborda la construcción de un proveedor básico que implemente todos estos casos de uso.

3.1. Casos de uso inter-proveedor

Los casos de uso que vienen predeterminados por el protocolo son los de consultar los datos de un usuario, añadir un nuevo contacto (o conseguir el permiso de acceso a los datos que un perfil permita) y borrar un contacto (o informar del deseo de dejar de tener el permiso de acceso a los datos de un usuario).

Todos estos casos de uso siguen una secuencia más o menos determinada de mensajes que requiere siempre de una consulta y una respuesta.

3.1.1. Datos de usuario

Finalidad: Acceder a la información de otro usuario del sistema.

Requisitos: El usuario tiene que ser actualmente un contacto nuestro (debemos haber realizado con anterioridad, y al menos una vez, el caso de uso Añadir contacto con resultado positivo).

Consulta que inicia el caso de uso: Consulta de datos de usuario (CDU).

Posibles respuestas:

- Respuesta de datos de usuario (RDU): La consulta se ha llevado a cabo con éxito y el proveedor destino ha respondido con la información a la que se tiene acceso.
- Respuesta de error (MERR): Ha ocurrido un error. El más común puede ser el de no tener permiso de acceso que ocurre cuando no lo hemos añadido correctamente como contacto.

3.1.2. Añadir contacto

Finalidad: Aportar las credenciales necesarias para acceder a los datos que un perfil determine de otro usuario del sistema.

Requisitos: Ninguno.

Consulta que inicia el caso de uso: Consulta de añadir contacto (CNC).

Posibles respuestas:

- Respuesta a la consulta de añadir contacto (RNC): Esta respuesta indica si se ha permitido el acceso o no.

- Respuesta de error (MERR): Ha ocurrido un error.

3.1.3. Borrar contacto

Finalidad: Indicar a un determinado usuario que no se desea seguir accediendo a sus datos.

Requisitos: El usuario tiene que ser actualmente un contacto nuestro (debemos haber realizado con anterioridad, y al menos una vez, el caso de uso Añadir contacto con resultado positivo).

Consulta que inicia el caso de uso: Consulta de borrar contacto (CBC).

Posibles respuestas:

- Respuesta a la consulta de borrar contacto (RBC): Esta respuesta indica si se ha podido borrar correctamente el contacto o no.

3.2. Casos de uso internos

Estos casos de uso se aportan con fines indicativos, puesto que a pesar de no ser necesarios para poder implementar el protocolo si lo son para que el servidor tenga la funcionalidad mínima necesaria para su funcionamiento.

3.2.1. Modificar datos

Finalidad: Permitir la modificación de los datos personales del usuario. Este caso de uso debe permitir que se añadan nuevos campos personales, así como direcciones, teléfonos e e-mails. También debe permitir borrarlos o modificarlos.

3.2.2. Listar contactos

Finalidad: Mostrar la lista de contactos de un determinado usuario para facilitar la selección del contacto del que se deseen mostrar los datos.

3.2.3. Crear perfil

Finalidad: crear un perfil de usuario para especificar los datos que se quieren compartir de modo que otros usuarios puedan acceder a ellos.

3.2.4. Modificar perfil

Finalidad: permitir la modificación de un perfil ya creado de forma que pueda actualizarse para quitar o añadir campos.

3.2.5. Borrar perfil

Finalidad: Permitir borrar un perfil que no fuera ya necesario. En este caso debe tenerse en cuenta lo que hacer con aquellos usuarios que estaban accediendo al perfil, por el que puede optarse por avisar o no hacer nada (por cuestiones prácticas, es necesario que un proveedor sea capaz de

3.2.6. Listar usuarios con acceso a un perfil

Finalidad: Mostrar la lista de usuarios del sistema que tienen derechos para acceder a un determinado perfil propio, de forma que se pueda controlar en todo momento quiénes son los que acceden a cada perfil.

3.2.7. Quitar permiso de acceso a perfil

Finalidad: Como complemento del anterior, este caso de uso permite hacer que un usuario deje de tener permiso para acceder a nuestros datos personales. Por supuesto, este caso de uso requiere que pueda cambiarse la contraseña del perfil puesto que es lo que vale de validación del usuario para obtener el derecho a acceder a los datos del perfil.

CAPÍTULO 4. Arquitectura del proveedor

La capa de negocio se ha implementado mediante una aplicación Java que actúa como un proveedor sin ninguna interfaz gráfica relevante. Toda interacción del administrador con el proveedor se hace a través de un icono del *System Tray* de Windows® de forma que pueden realizarse las tareas necesarias para su funcionamiento.

La arquitectura de este proveedor se define como un conjunto de módulos. Cada módulo es independiente de los demás y de forma que pueden trabajar concurrentemente, de forma que el proveedor puede estar llevando a cabo varios trabajos de forma simultánea.

Los módulos se comunican entre sí mediante una serie de mensajes que describen la tarea que debe ser realizada, o una respuesta con datos a una solicitud.

4.1. Configuración del proveedor

El proveedor dispone de un archivo de configuración que es leído cuando se inicia la aplicación, este archivo denominado 'configuracion.xml' contiene la información necesaria para configurar la comunicación con el resto de elementos del sistema además de controlar la carga.

Los parámetros que constan en este archivo son:

- `nombre`: Es el nombre de dominio o IP de la instancia de servidor GCM para el que el proveedor está trabajando.
- Parámetros de la base de datos:
    - 'bd_name': Nombre de la base de datos.
    - 'bd_user': usuario de la base de datos con los permisos necesarios para realizar todas las operaciones que necesite el proveedor.
    - 'bd_ password': contraseña del usuario de la base de datos.
    - 'bd_server': dirección IP en la que se encuentra la base de datos.
    - 'bd_port': Puerto del servicio de la base de datos.
- Puertos:
    - 'puerto_proveedor': es el puerto definido en el protocolo, por este puerto llegarán todas las solicitudes del protocolo, a su vez cualquier mensaje que sea enviado a otro servidor ha de ser a este puerto.
    - 'puerto_pa_web': Puerto por el que han de llegar las peticiones del punto de acceso web.
    - 'puerto_pa_aplic_java': puerto por el que han de llegar las solicitudes del punto de acceso java.
- Rendimiento:
    - 'max_lm': Número máximo de hilos lectores de mensajes del protocolo.
    - 'max_em': Número máximo de hilos generadores de mensajes del protocolo.

Toda esta información es extraída del fichero al cargarse el proveedor y es almacenada en la clase 'ConfiguracionProveedor' del paquete 'estandar', desde esta clase podrán acceder el resto de módulos del proveedor para consultar los datos que necesiten.

Un ejemplo del archivo de configuración es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuracion>
      <nombre>gcm-tipo.com</nombre>
      <bd>
            <bd_name>sistemas_informaticos</bd_name>
            <bd_user>proyecto</bd_user>
            <bd_password>XXXXXX</bd_password>
            <bd_server>172.16.0.7</bd_server>
            <bd_port>3306</bd_port>
      </bd>
      <puertos>
            <puerto_proveedor>4444</puerto_proveedor>
            <puerto_pa_web>5555</puerto_pa_web>
            <puerto_pa_aplic_java>7766</puerto_pa_aplic_java>
      </puertos>
      <rendimiento>
            <max_lm>5</max_lm>
            <max_em>5</max_em>
      </rendimiento>
</configuracion>
```

Tabla 4. Archivo de configuración del proveedor

4.2. Base tecnológica del proveedor

Para obtener un diseño robusto y eficiente, a la vez que fácil de mantener o ampliar, se ha diseñado una arquitectura tecnológica sobre la que se implementan todos los módulos internos del proveedor.

Todas estas clases que implementan la base tecnológica se encuentran en el paquete 'estandar' del proyecto, a excepción de la clase mensaje, usada como interfaz básico de todas las solicitudes internas del proveedor y respuestas a estas, y que ha sido definido en su propio paquete junto a todas las clases que definen.

Además, el proveedor es configurable a través del archivo 'configuracion.xml', en este se define el nombre del proveedor (dominio o IP que es usado en el nombre de sus usuarios), la configuración de la base de datos utilizada, los puertos del protocolo o de los puntos de acceso y varios parámetros de control de carga de trabajo.

4.2.1. Hilos de ejecución

Cada modulo del proveedor esta formado por una o más unidades de ejecución, cada unidad de ejecución implementa la clase 'Hilo' que otorga a la unidad de ejecución de las siguientes características:

- Un thread dentro del proceso, por lo que puede ejecutar su código concurrentemente con el resto de hilos.
- Un buzón, que se define como una cola de mensajes, y adonde otros hilos podrán enviar mensajes.
- Una referencia a la clase 'GestorHilos' descrita más adelante.
- Un identificador de hilo único para todo el proveedor.

Los hilos han sido diseñados para seguir un proceso de ejecución típico: Una vez inicializado, el hilo entra en un bucle, en el que mientras no se le indique la orden de finalizar hará lo siguiente.

Primero leerá un mensaje de su buzón, una vez obtenido el mensaje determinará que tarea debe realizar y la ejecutará, una vez terminada, si hay se ha de enviar una respuesta a otro hilo, generará la respuesta y se la enviará, entonces termina el ciclo y se comienza de nuevo leyendo otro mensaje.

Cada clase que implemente la interfaz hilo usará una constante de tipo que habrá tenido que ser definida en la interfaz 'CteGlobal', donde se almacenan los posibles tipos de hilos y los posibles estados.

4.2.2. Buzones

Los buzones son clases diseñadas para recibir y almacenar mensajes hasta que la clase propietaria del buzón vaya haciendo uso de ellos.

Los buzones están implementados como un monitor java, estos monitores tienen la cualidad de ejecutar un control de exclusión mutua sobre aquellos métodos a los que se etiquete como "synchronized" dentro de la clase monitor. Esto permite que al ser usada por varios hilos, no pueda haber dos hilos procesando un método simultáneamente, lo cual podría corromper la integridad de los datos del buzón.

El buzón implementa internamente una cola de objetos que implementan la interfaz 'Mensaje', es decir, los mensajes se van acumulando, y cada vez que el hilo propietario solicita un mensaje a su buzón, este le devuelve el mensaje que lleva más tiempo dentro de la cola, en el caso de que el buzón no contenga mensajes, el hilo se quedará 'dormido' en el buzón a la espera de que llegue algún mensaje que procesar.

4.2.3. Gestor de hilos

El gestor de hilos (clase 'GestorHilos') es una clase 'singleton' (solo existe una dentro del proveedor) que se implementa mediante un hilo independiente de los demás cuyo trabajo es el de centralizar el control de todos los hilos del proveedor, este objeto almacena las referencias al resto de hilos y asigna el identificador a cada hilo cuando se crea, además, todo hilo puede acceder al gestor de hilos para mostrar cualquier mensaje que deba ser conocido por el administrador o de informar sobre un error.

4.2.4. Mensajes

De forma que cualquier tipo de mensaje pueda ser almacenado en un buzón y además se puedan determinar tanto el tipo de mensaje, como el remitente del mensaje, se ha definido un interfaz que obliga a que todos los mensajes cumplan estos requisitos.

Además a cada mensaje se le da un identificador mediante un registro implementado en forma de *singleton*, que asegura que el identificador sea único en todo el proveedor.

Todos estos mensajes, sean del tipo que sean disponen de una serie de parámetros en los cuales se definen los módulos internos del proveedor a los que se dirigen, o el módulo remitente, de forma que cualquier módulo pueda saber a donde enviar otro mensaje en caso de generar una respuesta.

4.2.5. Monitores de carga

Varios módulos del proveedor hacen uso de un control de carga, el esquema es el siguiente. Existe un punto de entrada unificado a estos módulo llamados monitores, cada monitor es una clase 'singleton' (solo puede haber una instancia de la clase en el proveedor) y sincronizada de forma que no puede haber dos métodos ejecutándose a la vez. Estos monitores poseen una lista de hilos de la misma clase, todos estos hilos realizan el mismo tipo de tarea en paralelo, de

forma que se pueden ejecutar múltiples trabajos al mismo tiempo, cada hilo puede tener varios trabajos en espera, por lo que el monitor se encarga de determinar la carga de trabajo de cada hilo y cuando llega un nueva petición al monitor, este se lo entrega al hilo menos cargado, también es posible que el monitor decida crear un nuevo hilo si el resto están cargados, aunque existe un límite de hilos que puede ser determinado en la configuración del proveedor.

4.3. Comunicación dentro del proveedor

El proveedor está diseñado de forma modular, para comunicar a los módulos existe una serie de mensajes implementados a partir de clases java que implementan la interfaz mensaje descrita en la sección tecnológica del servidor. Estos mensajes se dividen principalmente en dos ramas, mensajes con funcionalidades auxiliares dentro del proveedor, y mensajes representantes de mensajes del protocolo GCM.

Mientras que los mensajes auxiliares se utilizan para tareas tales como pasar *sockets* entre clases receptoras y los hilos encargados de procesar las comunicaciones; o de pasar mensajes de error al gestor de hilos. Los mensajes representantes del protocolo contienen toda la información correspondiente a un mensaje del protocolo en forma de estructura de datos java, estas estructuras de datos se implementan en el paquete 'proveedor_datos'.

Los mensajes que representan al protocolo se separan a su vez en dos tipos diferenciados, consultas y respuestas, cada una de estas ramas dispone de una clase abstracta que a su vez implementa la interfaz mensaje. Estas clases abstractas definen los una serie de constantes de tipos de consultas o repuesta que han de utilizar las clases que las extienden para poder identificar a que tipo de mensaje del protocolo corresponden.

Las clases de datos están implementadas siguiendo un diseño de factoría, esto quiere decir que los mensajes realmente usan las clases de datos como interfaces que genera una clase factoría, esta clase factoría devuelve realmente clases que implementan las interfaces externas de datos. Este método permite una mejor un mejor mantenimiento de las clases de datos, ya que se puede cambiar la estructura de las clases sin necesidad de modificar todos los mensajes que hacen uso de ellas, ya que los mensajes acceden a los datos como si fueran interfaces.

4.4. Módulos del proveedor

Internamente el proveedor consta de tres módulos diferenciados por su funcionalidad, cada módulo. Cada uno de estos módulos se encarga de la comunicación con las otras capas del servidor y de comunicar las solicitudes que lleguen a través de él, al módulo que haya de encargarse de procesar la solicitud.

4.4.1. Módulo de comunicaciones

Es el encargado de las comunicaciones con el resto de proveedores, tiene la función de transformar los mensajes del protocolo que llegan al proveedor en mensajes propios del proveedor que implementen la clase 'Mensaje'; así como de transformar cualquier mensaje interno que se quiera entregar a otro proveedor a un mensaje del protocolo.

Por tanto el módulo de comunicaciones se subdivide en dos, clases de lectura para recibir mensajes, y clases de escritura para enviarlos.

Como punto de entrada para los mensajes del protocolo existe un hilo denominado 'ReceptorConexiones' que mantiene continuamente abierto un *socket* de servidor (ServerSocket) sobre el puerto TCP/IP definido en el protocolo. Los *sockets* de servidor se caracterizan por escuchar en un puerto hasta que un *socket* normal se conecta al puerto sobre el que el está escuchando, momento en el que abre un nuevo *socket* para proporcionar una vía de comunicación con el *socket* remoto. Una vez abierto el nuevo *socket*, el receptor crea un

mensaje en el que se referencia al nuevo *socket* y se lo envía al monitor de carga de los lectores de mensajes.

El monitor de lectores 'MonitorLM', es un *singleton* a la vez que un monitor reconcurrencia, que se encarga de gestionar la carga en la lectura de mensajes, este monitor tiene una lista con cada hilo de lectura y análisis de mensajes del protocolo, cuando llega un nuevo mensaje de nueva conexión con un *socket*, el monitor decide mediante el tamaño de las colas de los buzones de cada hilo lector, cual de ellos está menos cargado de trabajo, también puede tomar la decisión de crear un nuevo hilo siempre y cuando no se haya llegado ya a un límite de hilos definido en la configuración del proveedor.

Una vez el monitor de lectores de mensajes ha entregado el mensaje de nueva conexión a un lector, este se encargará de leer la cadena UTF8 del *socket*, una vez leída el lector usa la clase 'AnalizadorEstandarXML' que transforma la cadena de texto leída del *socket* a un mensaje java de la clase que corresponda según el tipo de mensaje leído. Además el analizador devuelve la información sobre el servidor remitente del mensaje, con su nombre podemos comparar a través del servicio DNS que la dirección IP asociada al dominio coincide con la dirección IP del *socket* remoto, lo cual permite autentificar al servidor remoto.

La clase 'AnalizadorEstandarXML' usa la tecnología DOM para procesar los mensajes XML que llegan al lector, DOM es capaz de construir una estructura con forma de árbol de nodos en el que cada cono representa un elemento del mensaje XML. A partir de este árbol el analizador determina la información de versión del protocolo que se está utilizando, nombre del servidor remitente y destinatario, y tipo de mensaje.

Una vez determinado el tipo de mensaje, el analizador pasa la parte del mensaje con el contenido a una de las clases java que implementan la interfaz 'Mensaje' y que será la encargada de analizar el resto del mensaje XML y almacenar la información de este mediante las clases de datos del proveedor.

Una vez transformado el mensaje del protocolo a un mensaje Java, el lector determina según el tipo de mensaje y el resto de información que lleve asociado a que módulo ha de enviarse.

Para mandar mensajes a otros servidores GCM, los módulos del proveedor envían los mensajes java a la sección de escritura del módulo de comunicaciones, de forma similar al 'MonitorLM', la sección de escritura dispone de otro monitor llamado 'MonitorEM' que se encarga de controlar la carga de los escritores y crear nuevos cuando sean necesarios.

El lector de mensajes transmitirá los mensajes que llegan de los otros módulos a un hilo de clase 'EscritorMensajes' el cual se encarga de transformar los mensajes que lee de su buzón en mensajes XML del protocolo, y posteriormente enviarlos al servidor destinatario.

Los mensajes XML son creados manualmente como cadenas de texto a partir de la información del mensaje java, el cual tiene la responsabilidad de ser capaz de devolver una cadena en el formato XML del protocolo con los contenidos del mensaje, a esta cadena, el escritor de mensajes le añadirá una cabecera con los datos de versión del protocolo remitente y destinatario.

Una vez obtenido el mensaje XML el escritor de mensajes abrirá un *socket* sobre la IP que obtendrá mediante el servicio DNS usando el nombre del servidor destinatario, además utilizará el puerto descrito en el protocolo. Cuando el servidor destinatario abra a su vez un puerto para establecer comunicación, el escritor de mensajes transmitirá la cadena XML y a continuación cerrará el puerto. Entonces se considerará trasmitido el mensaje y el escritor volverá a consultar su buzón en busca de un nuevo mensaje que transmitir.

4.4.2. Módulo de base de datos

El módulo de base de datos se encarga de gestionar las conexiones con la base de datos y de ejecutar las consultas que reciba.

Por motivos de seguridad, este módulo delega toda la responsabilidad de la integridad de los datos a la capa de datos y al gestor de bases de datos, con lo que evita tener que comprobar si los datos introducidos son válidos, o no, al saber de antemano que la capa de datos no almacenará, en ningún caso, datos que sean incorrectos o que incumplan alguna de las restricciones de integridad impuestas a los mismo. De esta forma, se simplifica el módulo y se limita a realizar tres funciones principales:

- Gestionar las conexiones con la base de datos.
- Gestionar las sesiones de los usuarios del proveedor.
- Responder a las consultas dirigidas al proveedor por los usuarios del proveedor o por los usuarios externos al mismo.

4.4.2.1. Gestión de conexiones con la base de datos

Este módulo tiene la capacidad de gestionar más de una conexión con la base de datos, con lo que aprovecha una de las ventajas que aportan los gestores de bases de datos actuales, y *MySQL* en particular que será el gestor empleado en la capa de datos.

El módulo gestiona 10 conexiones con la base de datos, aparte de una conexión especial que emplea para realizar pequeñas *queries* colaterales a las consultas. Cada una de ellas en un hilo independiente lo que las hace autónomas e independientes unas de otras. Los problemas de concurrencia, se dejan para ser resueltos por el gestor de bases de datos, simplificando así la gestión de los hilos de conexión. Este número de conexiones permite ejecutar de forma simultánea 10 consultas a la base de datos.

Al considerar que todas las consultas a responder han de ejecutar un número similar de *queries*, el módulo emplea *round-robin* como modo de distribución de las consultas. Así asigna cada consulta a la siguiente conexión supuestas estas en un vector circular.

A nivel de implementación, la clase 'MonitorBD' es la encargada de comunicarse con el resto de los módulos del proveedor así como de recibir las consultas que habrá de ejecutar en la base de datos.

La clase 'BaseDatos' implementa la conexión básica con la base de datos que es empleada por 'ConexionBD' para realizar la conexión de forma independiente. Así, cada objeto de 'ConexionBD' es un hilo de conexión con la base de datos independiente de los demás.

'ConexiónBDMonitor' es la conexión especial del monitor para realizar aquellas *queries* que no requieren de gran procesamiento de información y que son siempre colaterales a alguna otra consulta.

4.4.2.2. Gestión de sesiones

La gestión de sesiones se lleva de forma que pueda conocerse en cada momento qué usuarios han iniciado sesión (y están actualmente conectados) y cuáles no. Así como de saber a través de qué puntos de acceso han establecido la conexión.

Esta gestión de sesiones permite establecer controles más estrictos de seguridad como no responder a consultas que provengan de otros remitentes diferentes de aquellos que han iniciado sesión en el proveedor. Por supuesto una excepción a esto son aquellas consultas que provienen del módulo de comunicaciones, y por ende de usuarios de otros proveedores, y las consultas de inicio de sesión y de creación de cuentas de usuario que han de ser siempre remitidas por usuarios que aún no han iniciado sesión o que no poseen aún una cuenta de usuario.

Las clases 'Login' y 'LoginUser' permiten llevar este control de sesiones de forma que el monitor puede comprobar en 'Login' si un determinado usuario ha iniciado o no sesión.

4.4.2.3. Ejecución de las consultas

El sistema de ejecución se basa en la creación de un ejecutor determinado para cada tipo de consulta a ejecutar, de forma que cuando se recibe una consulta, se comprueba su tipo y se le envía al ejecutor apropiado.

Todas las conexiones de la base de datos pueden ejecutar todas las consultas, y como han de ser independientes, deben crear una instancia de cada tipo de ejecutor y así poder ejecutarlas.

Cada conexión con la base de datos almacena en una tabla *hash* una instancia de cada tipo de ejecutor (cuyo código *hash* coincide con el de la consulta que ejecuta) de forma que el proceso para la ejecución de una consulta es:

1. Comprobar el tipo de la consulta a ejecutar
2. Obtener el ejecutor apropiado de la tabla hash
3. Ejecutar la consulta en dicho ejecutor

A partir de este momento es el propio ejecutor el responsable de redirigir la respuesta apropiada al origen de la consulta.

4.4.3. Módulo de punto de acceso

El módulo de puntos de acceso es el encargado de recibir las solicitudes de los puntos de acceso, realizar las transacciones que sean necesarias y enviar de vuelta las respuestas al punto de acceso solicitante.

Como cada punto de acceso puede estar implementado de forma muy distinta y usando tecnologías diferentes el módulo de puntos de acceso esta diseñado de forma que existan varios conectores, cada conector ha de encargarse de obtener las solicitudes de su punto de acceso sea en el formato que sea, mientras que un punto de acceso puede enviar las solicitudes en forma de cadenas de texto que siguieran un determinado formato, otro punto de acceso podría enviarlas en forma de clases java.

Así pues estos conectores han de transformar las solicitudes a los tipos de mensajes de consultas y respuestas que usa el proveedor, además deben ser capaces de almacenar los datos necesarios para poder transmitir de vuelta una respuesta a la solicitud del usuario.

Una vez el conector ha transformado la solicitud en un objeto de la clase ‘Consulta’, este objeto es enviado a un hilo llamado ‘GestorPA’ el cual se encarga de crear una tarea para gestionar cada solicitud.

Cada tarea creada por el gestor llevará asignado un identificador de tarea único en todo el sistema GCM de forma que aunque se deban realizar transmisiones a otros servidores GCM la tarea sea inconfundible. Esto ha de ser así porque cada tarea representa una instancia de un caso de uso del protocolo, y lo más normal es que simultáneamente existan muchas instancias del mismo caso de uso ejecutándose a la vez a lo largo de todo el servicio GCM, por lo que ha de determinarse unívocamente a que servidor corresponde la tarea y dentro de este a que solicitud en concreto de que usuario.

Las tareas tienen un esquema de funcionamiento bastante sencillo, toda tarea es generada por la consulta proveniente de la solicitud del usuario, a partir de esta consulta, la tarea crea otra serie de consultas que serán enviadas a los otros módulos del proveedor para recopilar la información necesaria para la respuesta, ya sea al módulo de la base de datos en caso de que los datos necesarios sean locales al servidor; o al módulo de comunicaciones si los datos necesarios han de provenir de otro servidor del servicio GCM.

Una vez enviadas las consultas, la tarea entra en un bucle en la que irá recibiendo y procesando las respuestas según lleguen, cuando todas las respuestas a las solicitudes generadas por la tarea hayan llegado, la tarea creará una respuesta única resultante de unificar todas las que llegaron,

respuesta que enviará de vuelta al conector adecuado para que este lo transforme en un tipo de mensaje que entienda su punto de acceso.

Dentro del proveedor tipo solo ha sido necesario implementar uno de estos conectores, ya que todos los puntos de acceso usan tecnologías basadas en Java, por lo que los puntos de acceso usan el mismo conjunto de mensajes que el proveedor, por lo que no es necesaria la traducción de mensajes.

Este conector hace uso de dos hilos, por una parte, un hilo llamado 'ReceptorWeb' que escucha por un *socket* servidor conectado al puerto del conector web de forma similar al receptor de conexiones del protocolo. Y por otra parte un hilo llamado 'ConectorWeb' que se encarga de procesar cada conexión.

El conector web lee el mensaje java del *socket* que le pasa el receptor *web*, y crea un identificador de tarea para dicha comunicación, el cual asigna al mensaje leído y lo envía al gestor del punto de acceso mediante un mensaje auxiliar que además de contener el mensaje leído del *socket*, indica el conector al que se ha de enviar la respuesta. Una vez enviado el mensaje, el conector guarda un registro en una tabla con el identificador de tarea como índice y el *socket* abierto con el punto de acceso que envió el receptor.

Cuando se envía de vuelta una respuesta al conector *web*, este lee el identificador de tarea de la respuesta, a través de la cual es capaz de indexar en la tabla y obtener el *socket* adecuado del punto de acceso, entonces el conector escribe en el *socket* la respuesta y lo cierra, borrando de la tabla el registro de la solicitud ya concluida.

CAPÍTULO 5. Arquitectura de la base de datos

La base de datos es encargada de los datos del servidor. Sus funciones principales son:

- Almacenar y conservar los datos necesarios de los usuarios y del proveedor.
- Gestionar y conservar las restricciones de integridad de los datos para evitar errores en los datos almacenados.

Para conseguir realizar correctamente las funciones hace uso de las características del gestor de base de datos MySQL (características que son comunes con casi todos lo gestores actuales) como son:

- **Control de las restricciones de integridad**: muchos datos deben cumplir con ciertas restricciones como que una dirección no puede existir si no está asociada a un usuario existente en el servidor.
- **Gestión de transacciones**: determinadas consultas del proveedor requieres más de una modificación de la base de datos que deben ejecutarse todas o ninguna porque si se quedase en medio por cualquier motivo, podría quedar en estado indeterminado y no ser válidos los datos almacenados en la base de datos, por ello, todas aquellas consultas que requiriesen más de una modificación habrán de ser ejecutadas dentro de una transacción.

5.1. Estructura en tablas de la base de datos

La base de datos se estructura en las siguientes tablas:

- usuario

```
CREATE TABLE usuarios (
      id                VARCHAR(20) NOT NULL,
      password          VARCHAR(20) NOT NULL,
      PRIMARY KEY (id)
);
```

- usuario_info_personal

```
CREATE TABLE usuarios_info_personal (
      usuario           VARCHAR(20)       NOT NULL,
      nombre            VARCHAR(50)       NOT NULL,
      apellidos         VARCHAR(50)       NOT NULL,
      nombre_ingles     VARCHAR(100)      NOT NULL,
      apodo             VARCHAR(50)       NOT NULL,
      lugar_nacimiento  VARCHAR(50)       NOT NULL,
      fecha_nacimiento  DATE,
      nacionalidad      VARCHAR(3)        NOT NULL    DEFAULT 'XXX',
      FOREIGN KEY (usuario)
            REFERENCES usuarios(id)
            ON UPDATE CASCADE ON DELETE CASCADE,
      FOREIGN KEY (nacionalidad) REFERENCES codigos_pais(cod),
      PRIMARY KEY (usuario)
);
```

- usuario_direcciones

```
CREATE TABLE usuarios_direcciones (
      usuario           VARCHAR(20)       NOT NULL,
      id                VARCHAR(50)       NOT NULL,
      texto             VARCHAR(255)      NOT NULL,
      codigo_postal     VARCHAR(20)       NOT NULL,
      poblacion         VARCHAR(50)       NOT NULL,
      provincia         VARCHAR(50)       NOT NULL,
      pais              VARCHAR(3)        NOT NULL    DEFAULT 'XXX',
      FOREIGN KEY (usuario)
            REFERENCES usuarios(id)
            ON UPDATE CASCADE ON DELETE CASCADE,
      FOREIGN KEY (pais) REFERENCES codigos_pais(cod),
      PRIMARY KEY (usuario, id)
);
```

- usuario_campos

```
CREATE TABLE usuarios_campos (
      usuario           VARCHAR(20)       NOT NULL,
      tipo              VARCHAR(5)        NOT NULL,
      id                VARCHAR(50)       NOT NULL,
      texto             VARCHAR(50)       NOT NULL,
      FOREIGN KEY (usuario)
            REFERENCES usuarios(id)
            ON UPDATE CASCADE ON DELETE CASCADE,
      FOREIGN KEY (tipo) REFERENCES codigos_campo(campo),
      PRIMARY KEY (usuario, id, tipo)
);
```

- usuarios_idiomas

```
CREATE TABLE usuarios_idiomas (
      usuario           VARCHAR(20)       NOT NULL,
      idioma            VARCHAR(2)        NOT NULL,
      FOREIGN KEY (usuario)
            REFERENCES usuarios(id)
            ON UPDATE CASCADE ON DELETE CASCADE,
      FOREIGN KEY (idioma) REFERENCES codigos_idioma(cod),
      PRIMARY KEY (usuario, idioma)
);
```

- grupos

```
CREATE TABLE grupos (
      usuario     VARCHAR(20)       NOT NULL,
      grupo       VARCHAR(50)       NOT NULL    DEFAULT '_undef_',
      FOREIGN KEY (usuario)
            REFERENCES usuarios(id)
            ON UPDATE CASCADE ON DELETE CASCADE,
      PRIMARY KEY (usuario, grupo)
);
```

- grupos_contactos

```
CREATE TABLE grupos_contactos (
```

```
      usuario     VARCHAR(20)       NOT NULL,
      grupo       VARCHAR(50)       NOT NULL    DEFAULT '_undef_',
      contacto    VARCHAR(100)      NOT NULL,
      nombre      VARCHAR(50)       NOT NULL,
      apellidos   VARCHAR(50)       NOT NULL,
      FOREIGN KEY (usuario, grupo)
            REFERENCES grupos(usuario, grupo)
            ON UPDATE CASCADE ON DELETE CASCADE,
      PRIMARY KEY (usuario, contacto)
);
```

- perfiles

```
CREATE TABLE perfiles (
      usuario                 VARCHAR(20) NOT NULL,
      perfil                  VARCHAR(50) NOT NULL,
      password          VARCHAR(20) NOT NULL,
      FOREIGN KEY (usuario)
            REFERENCES usuarios(id)
            ON UPDATE CASCADE ON DELETE CASCADE,
      PRIMARY KEY (usuario, perfil)
);
```

- perfiles_info_contactos

```
CREATE TABLE perfiles_info_personal (
      usuario           VARCHAR(20) NOT NULL,
      perfil            VARCHAR(50) NOT NULL,
      nombre            VARCHAR(1)  NOT NULL    DEFAULT 'N',
      apellidos         VARCHAR(1)  NOT NULL    DEFAULT 'N',
      nombre_ingles     VARCHAR(1)  NOT NULL    DEFAULT 'N',
      lugar_nacimiento  VARCHAR(1)  NOT NULL    DEFAULT 'N',
      fecha_nacimiento  VARCHAR(1)  NOT NULL    DEFAULT 'N',
      nacionalidad      VARCHAR(1)  NOT NULL    DEFAULT 'N',
      idiomas           VARCHAR(1)  NOT NULL    DEFAULT 'N',
      apodo             VARCHAR(1)  NOT NULL    DEFAULT 'N',
      FOREIGN KEY (usuario, perfil)
            REFERENCES perfiles(usuario, perfil)
            ON UPDATE CASCADE ON DELETE CASCADE,
      PRIMARY KEY (usuario, perfil)
);
```

- perfiles_direcciones

```
CREATE TABLE perfiles_direcciones (
      usuario           VARCHAR(20) NOT NULL,
      perfil            VARCHAR(50) NOT NULL,
      direccion         VARCHAR(50) NOT NULL,
      FOREIGN KEY (usuario, perfil)
            REFERENCES perfiles(usuario, perfil)
            ON UPDATE CASCADE ON DELETE CASCADE,
      FOREIGN KEY (usuario, direccion)
            REFERENCES usuarios_direcciones(usuario, id)
            ON UPDATE CASCADE ON DELETE CASCADE,
      PRIMARY KEY (usuario, perfil, direccion)
);
```

- perfiles_campos

```
CREATE TABLE perfiles_campos (
      usuario           VARCHAR(20) NOT NULL,
      perfil            VARCHAR(50) NOT NULL,
      tipo              VARCHAR(5)  NOT NULL,
      campo             VARCHAR(50) NOT NULL,
      FOREIGN KEY (usuario, perfil)
            REFERENCES perfiles(usuario, perfil)
            ON UPDATE CASCADE ON DELETE CASCADE,
      FOREIGN KEY (usuario, campo, tipo)
            REFERENCES usuarios_campos(usuario, id, tipo)
            ON UPDATE CASCADE ON DELETE CASCADE,
      PRIMARY KEY (usuario, perfil, tipo, campo)
);
```

- perfiles_contactos

```
CREATE TABLE perfiles_contactos (
      usuario           VARCHAR(20) NOT NULL,
      perfil            VARCHAR(20) NOT NULL,
      contacto          VARCHAR(100)      NOT NULL,
      FOREIGN KEY (usuario, perfil)
            REFERENCES perfiles(usuario, perfil)
            ON UPDATE CASCADE ON DELETE CASCADE,
      PRIMARY KEY (usuario, perfil, contacto)
);
```

También hace uso de tres tablas donde se almacenan códigos necesarios:

- codigos_pais

```
CREATE TABLE codigos_pais (
      pais              VARCHAR(50)       NOT NULL,
      cod               VARCHAR(3)        NOT NULL,
      PRIMARY KEY (cod)
);
```

- codigos_idioma

```
CREATE TABLE codigos_idioma (
      idioma            VARCHAR(50)       NOT NULL,
      cod               VARCHAR(2)        NOT NULL,
      PRIMARY KEY (cod)
);
```

- codigos_campo

```
CREATE TABLE codigos_campo (
      campo             VARCHAR(5)        NOT NULL,
      PRIMARY KEY (campo)
);
```

CAPÍTULO 6. Arquitectura de los puntos de acceso

A pesar de que como se ha expuesto los puntos de acceso pueden ser de muy variada naturaleza (todo depende de lo que el gestor del servidor considere necesario ofrecer a sus clientes), para el diseño de un servidor funcional se optó por la creación de tres puntos de acceso que pudieran servir de referencia:

1. Un punto de acceso basado en JSP.
2. Otro, basado en tecnología WAP.
3. Y por último, uno a través de una aplicación de Java construida con la ya clásica API de Java.

6.1. Punto de acceso basado en JSP (WebAdv)

El punto de acceso WebAdv es el punto de acceso más completo en el servidor tipo ya que es capaz de llevar a cabo todas las funcionalidades contempladas en el protocolo.

6.1.1. Tecnología JSP

Este punto de acceso ha sido diseñado como un módulo cargado en un servidor de aplicaciones web. El servidor elegido ha sido Tomcat debido a su compatibilidad con las arquitecturas de servlets y JSP basadas en java.

Los Servlets son la respuesta de la tecnología Java a la programación CGI. Son programas que se ejecutan en un servidor Web y construyen páginas Web. Construir páginas Web al vuelo es útil (y comúnmente usado) por un número de razones:

- La página Web está basada en datos enviados por el usuario. Por ejemplo, las páginas de resultados de los motores de búsqueda se generan de esta forma, y los programas que procesan pedidos desde *sites* de comercio electrónico también.
- Los datos cambian frecuentemente. Por ejemplo, un informe sobre el tiempo o páginas de cabeceras de noticias podrían construir la página dinámicamente, quizás devolviendo una página previamente construida y luego actualizándola.
- Las páginas Web que usan información desde bases de datos corporativas u otras fuentes. Por ejemplo, usaríamos esto para hacer una página Web en una tienda on-line que liste los precios actuales y el número de artículos en stock.

Java Server Pages (JSP) es una tecnología que nos permite mezclar HTML estático con HTML generado dinámicamente. Muchas páginas Web que están construidas con programas CGI son casi estáticas, con la parte dinámica limitada a muy pocas localizaciones. Pero muchas variaciones CGI, incluyendo los servlets, hacen que generemos la página completa mediante nuestro programa, incluso aunque la mayoría de ella sea siempre lo mismo. JSP nos permite crear dos partes de forma separada. Aquí tenemos un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Welcome to Our Store</TITLE></HEAD>
<BODY>
      <H1>Welcome to Our Store</H1>
      <SMALL>Welcome,
      <!-- User name is "New User" for first-time visitors -->
      <% out.println(Utils.getUserNameFromCookie(request)); %>
```

```
      To access your account settings, click
      <A HREF="Account-Settings.html">here.</A></SMALL>
      <P>Regular HTML for all the rest of the on-line store's Web
      page.</p>
</BODY>
</HTML>
```

Listado 9. Ejemplo de código JSP

Además, otra ventaja de usar JSP radica en que mientras cada archivo JSP corresponde al equivalente de una página HTML, pero generada de forma dinámica. Todos los JSP de un módulo pueden hacer de recursos comunes a la aplicación, esto recursos son los *bean.* Los bean son clases normales y corrientes de java que se pueden usar dentro del servidor web con distintos alcances, para usar un bean basta con incluir en el JSP una declaración con la siguiente sintaxis:

<jsp:useBean id="beanName" scope="session" class="someClass"/>

Así pues para cada *bean* usado en un JSP se ha de definir el nombre del *bean*, el alcance que puede ser de página, de sesión o de aplicación, hay que tener en cuenta que es fundamental que el nombre de un *bean* de sesión o de aplicación sea el mismo en todos los JSP que lo usen ya que si no se crearían tantos *bean* como nombres distintos se usaran.

Aparte de los objetos bean, las páginas JSP también pueden hacer uso de clases java estándar, en la que los objetos declarados serán locales a código de la página JSP.

Todo esto permite hacer que las páginas JSP sean capaces de generar una interfaz gráfica idéntica a la que se puede generar con una página HTML, pero con toda la potencia de java para controlar la información que se ha de mostrar en la página, o procesar los datos provenientes de los formularios.

6.1.2. Configuración del proveedor

El punto de acceso WebAdv contiene una clase llamada 'Configuration', la cual contiene los datos necesarios para ponerse en contacto con el proveedor tipo y con la base de datos local del punto de acceso, los parámetros definidos en esta clase en forma de constantes son:

- DEALER_NAME: Nombre del dominio del servidor GCM al que representa el punto de acceso.
- DEALER_PORT: Puerto en el que escucha el conector del proveedor para las solicitudes del punto de acceso WebAdv.
- DEALER_PA_PASSWORD: en caso de necesidad de mayor seguridad en la conexión entre proveedor y punto de acceso, se puede usar esta constante como contraseñar para identificar al punto de acceso en las comunicaciones con el proveedor.
- PA_DATABASE_ADDRESS: dirección IP en la que se puede encontrar la base de datos del punto de acceso.
- PA_DATABASE_PORT: Puerto en el que la base de datos del punto de acceso está dando servicio.
- PA_DATABASE_USER: Nombre de un usuario de la base de datos con los permisos necesarios.
- PA_DATABASE_PASSWORD: contraseña del usuario de la base de datos.
- PA_DATABASE_NAME: Nombre de la base de datos del punto de acceso.

Los valores de esta clase han de ser definidos antes de compilar el módulo, de esta manera se mejora la seguridad del sistema al ser imposible obtener esta información una vez instalado el módulo al estar fusionadas dentro del código java que se desplegará en el servidor *Tomcat*.

En caso de querer instalar este punto de acceso cambiando algún parámetro, tales como la localización del proveedor, o el acceso a la base de datos, bastará con modificar las constantes de la clase y volver a compilar el módulo.

A continuación se muestra un ejemplo de la configuración:

```
public interface Configuration {

  public final static String DEALER_NAME = "172.16.0.7";
  public final static int DEALER_PORT = 5555;
  public final static String DEALER_PA_PASSWORD = "jh8a6hgb";

  public final static String PA_DATABASE_ADDRESS = "localhost";
  public final static int PA_DATABASE_PORT = 3306;
  public final static String PA_DATABASE_USER = "proyecto";
  public final static String PA_DATABASE_PASSWORD = "proyecto";
  public final static String PA_DATABASE_NAME = "pa_data";
}
```

Listado 10. Ejemplo de la configuración de WebAdv

6.1.3. Soporte de múltiples idiomas

Siguiendo el concepto de globalidad del sistema CGM, el punto de acceso ha sido dotado con una base tecnológica que permite la visualización de la página en cualquier idioma.

Esta tecnología se basa en el uso por parte del código JSP de una serie de referencias en vez de escribir el texto de la página directamente en el código HTML, cada referencia tiene asociadas una serie de traducciones según los idiomas que hayan sido definidos.

Para implementar este sistema el punto de acceso dispone de una pequeña base de datos en la que mediante tres tablas se definen los idiomas, las referencias y las traducciones para cada referencia en los distintos idiomas definidos.

Para mejorar la eficiencia de la traducción que ha de efectuar el servidor *web* cada vez que se genera una página, los datos de traducciones se cargan en una estructura la primera vez que se pide una traducción, esta estructura se almacena en un bean de aplicación, por lo que todos los JSP usan la misma instancia de la estructura.

Esta estructura consiste en una tabla *hash* que indexa mediante el identificador asociado a cada idioma otra tabla *hash* por cada idioma definido, las tablas de idioma están a su vez indexadas por la referencia que corresponde a cada traducción del idioma de la tabla. Así pues, simplemente indexando primero con el código del idioma que usa el usuario y luego con la referencia para obtener la traducción del término deseado.

Esta estructura esta rodeada por la clase *Languages* que ofrece además de la función que obtiene las traducciones otra para reiniciar la estructura de los idiomas, ambas funciones deben acceder a través de un control de concurrencia tipo lectores – escritores que permite la simultaneidad de las lecturas a la vez que controla que no pueda haber ninguna lectura mientras que se esté efectuando alguna escritura.

También aporta funciones que permiten trabajar con la base de datos idiomática, que junto una serie de páginas JSP, permite la inserción de idiomas, referencias y términos, además de su actualización, de forma cómoda para el administrador del punto de acceso.

6.1.4. Control de sesiones

Para controlar que solo los usuarios del sistema pueden acceder a la aplicación web y almacenar los datos del usuario una vez se haya registrado, de forma que estén disponibles para cualquier página JSP que los necesite, se usa un *bean* con alcance de sesión de la clase 'Session', definida en el paquete 'standard' del punto de acceso.

El objeto de sesión hace uso de una tabla en la base de datos del punto de acceso denominada *user*, en esta tabla se almacena el nombre del usuario y el idioma que este ha elegido para ver la aplicación.

Para controlar la sesión, cada página JSP llama a un método de este *bean* al principio de la página, este método determina si el usuario ya se ha autentificado o si en caso de algún problema interno la aplicación está fuera de servicio, en caso de que ocurra alguna de estas dos situaciones, la función ejecuta una redirección sobre la página JSP que está realizando la llamada o bien a la página 'login' (autentificación), o bien a la página 'outOfService' (fuera de servicio).

Para autentificarse el usuario ha de introducir en la página 'login' su nombre de usuario del servicio (sin necesidad de incluir la parte del nombre correspondiente al dominio del servidor GCM) y la contraseña que tenga asignada en el servicio, una vez introducidos el bean de sesión se encarga de comunicarse con el proveedor el cual indicará si los datos son correctos o no, en caso de que lo sean, el objeto de sesión inicializa los datos del usuario con la información de la base de datos del punto de acceso y da como iniciada la sesión, la cual permanecerá abierta hasta que el usuario realice la operación de logout, con su consecuente comunicación al proveedor.

6.1.5. Gestión de formularios

Existe un paquete dentro del punto de acceso WebAdv denominado 'forms' y cuya misión es la de gestionar el uso de los formularios. La clase abstracta 'FormManager' del paquete contiene las bases de todos los formularios, los cuales están compuestos por una serie de campos que han de ser implementados mediante la clase 'FormField'.

Cada campo creado como un objeto de esta clase puede definir usando una constante de la clase FormField, que indica el tipo de dato se debería escribir en el campo, además se puede definir si es obligatorio rellenar el campo y el nombre del campo dentro del formulario. Entre los tipos de datos que pueden representar los campos están: direcciones de dominio, nombres de cuenta de usuario, direcciones de correo electrónico, valores naturales, etc.

Cuando se procesa el formulario el FormManager asigna a cada campo el valor que se obtiene del objeto *request* de la petición HTML, entonces el campo comprueba que el dato que obtiene es del tipo adecuado llamando a una de las funciones de comprobación de datos descritas más adelante. Si alguno de los campos del formulario se considera incorrecto, el formulario entero se considerará como incorrecto y será necesario volver a mostrar el formulario.

Cada formulario que extienda de la clase form manager simplemente ha de rellenar en su constructor que campos definen el formulario, e implementar un método getHTML que devuelva el código HTML con el formulario, para facilitar a generar este código se usarán las funciones de visualización de DisplayFunctions descritas más adelante.

6.1.6. Funciones de visualización

Existe una serie de funciones de visualización repartidas entre las clases 'DisplayFunctions' y 'UserDisplayFunctions'. Cada una de estas funciones devuelve un código HTML que puede ser insertado directamente en las páginas JSP que hagan uso de ellas.

Las clases de 'UserDisplayFunctions' son las encargadas de generar el código de las cabeceras, menús y pies de página de todas las páginas JSP de la aplicación. Estas funciones son dinámicas en el sentido de que las opciones que muestran son diferentes para cada página, según los parámetros que esta le pase a cada función en su llamada.

Además el código generado, como el resto de la página se muestra siempre en el idioma seleccionado por el usuario.

Una de las características más importantes de estas funciones, en concreto de la cabecera y pie de página, es mostrar el formulario de selección de idioma, ya que este permite al usuario cambiar el idioma en el que ve la página en cualquier momento.

Aparte están las funciones de 'DisplayFunctions', las cuales se encargan de generar el código HTML para distintas partes de los formularios, existiendo funciones para generar campos normales de texto, áreas de texto, campos de contraseña, campos de opciones, y algunos más específicos, tales como campos de selección de idiomas, de países o campos de fechas.

Además, algunos de estas funciones están asociadas a las funciones de gestión de formularios, por lo que están preparadas para mostrar errores en el caso de que sean detectados por los componentes de los formularios a los que representan.

6.1.7. Conector con el proveedor

La clase 'Connetor' definida en el paquete 'communications' permite la comunicación del punto de acceso WebAdv con el proveedor. Esta clase provee al resto de páginas JSP o al *bean* de sesión de funciones por las cuales el punto de acceso puede enviar consultas definidas en el paquete 'proveedor_mensajes.consultas' generado por el propio proveedor tipo. Una vez el conector envía una consulta se queda a la espera de una respuesta del proveedor en forma de un mensaje de los definidos en el paquete 'proveedor_mensajes.respuestas'. Mediante este simple procedimiento el punto de acceso puede hacer un uso completo de todas las funcionalidades que el proveedor facilita a sus usuarios.

La conexión que la clase 'Connector' abre con el proveedor es a través de un socket abierto contra la IP del proveedor y el puerto que estén definidos en la clase Configuration antes definida.

6.1.8. Comprobación de tipos y aplicación de formato a datos

La clase 'InputCheckFunctions' del paquete 'standard' contiene una serie de funciones que pueden dividirse en dos tipo de funcionalidades.

Primero está una serie de funciones que permiten comprobar si los datos que pasan a estas funciones tienen un formato acorde al tipo de dato que representa cada una de estas funciones, estas son las funciones usadas lo los campos de los formularios definidos mediante el gestor de formularios.

Aparte se encuentra una serie de funciones que permiten dar formato a los datos que hayan de ser mostrados en código HTML, estas funciones se aseguran de que la aparición de ciertas cadenas de caracteres no puedan corromper la estructura de ningúin código HTML, esto se realiza sustituyendo toda aparición de ciertos carateres potencialmente peligrosos para la estructura HTML en otros que no ocasionen problemas, los dos más destacados son los caracteres '"' y los '<' que son capaces de romper la estructura de un atributo o de una etiqueta respectivamente, de forma que el usuario podría llegar a introducir su propio código HTML en la web si no fuera controlado.

6.1.9. Flujo de trabajo de la aplicación WebAdv

La aplicación WebAdv tiene capacidad para ejecutar todas las funcionalidades ofrecidas por el proveedor tipo. Estas funcionalidades se dividen en tres grupos:

6.1.9.1. Contactos del usuario

El usuario dispone de una lista de los contactos a los que puede consultar datos, el usuario puede crear grupos para organizar a sus contactos y mover a los contactos entre ellos.

Pulsando sobre un contacto de la lista, el usuario pasará a ver los datos que los perfiles a los que tenga acceso permitan.

Para añadir nuevos contactos, el usuario simplemente tendrá que pulsar sobre el enlace de 'Nuevo Contacto' y rellenar los datos del nombre del nuevo contacto, el servidor al que pertenece, el perfil al que se que se quiere tener acceso y la contraseña de dicho perfil.

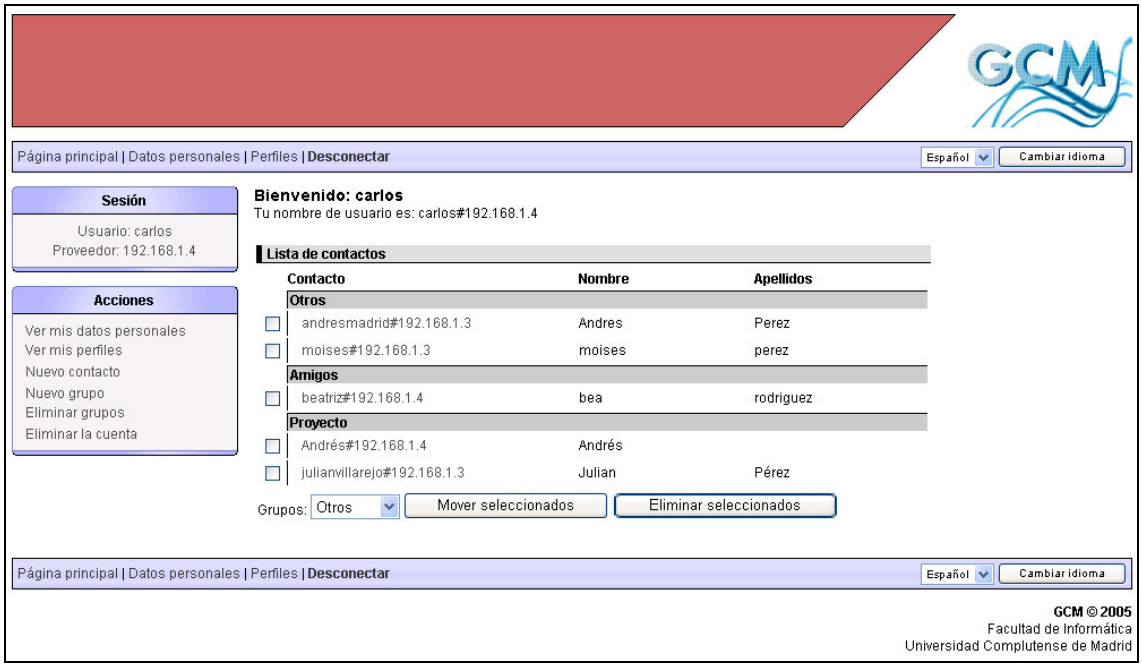


Ilustración 1. Pantalla principal de WebAdv

6.1.9.2. Datos del usuario

El usuario puede ver y modificar sus datos en la zona de datos personales, además puede añadir y eliminar todos los capos de teléfonos, *emails*, direcciones o idiomas del usuario que desee. También puede añadir campos personalizados con la información que el usuario pueda querer compartir con sus contactos.

Todas estas opciones permiten que cada usuario cree y personalice sus datos personales todo lo que necesite para que el servicio se adapte a sus necesidades de la mejor forma posible.

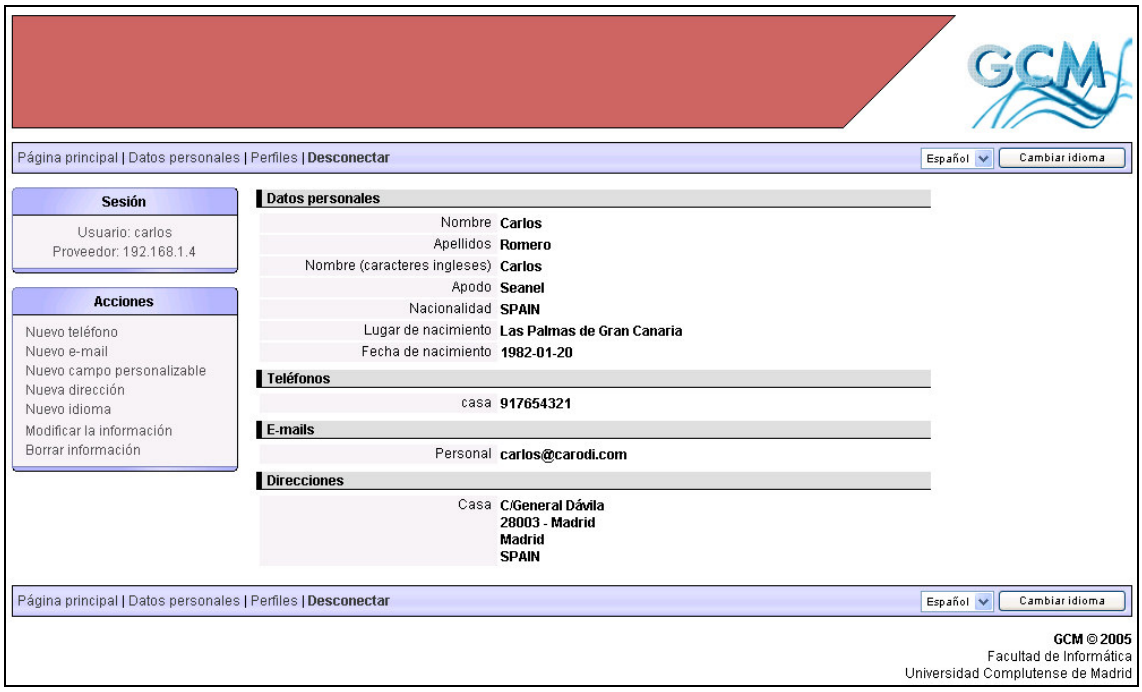


Ilustración 2. Pantalla de datos personales de WebAdv

6.1.9.3. Perfiles del contacto

El usuario podrá crear o eliminar los perfiles que desee, así como actualizarlos cambiando los datos a los que de acceso el perfil, o cambiando la contraseña necesaria para obtener el permiso del perfil.

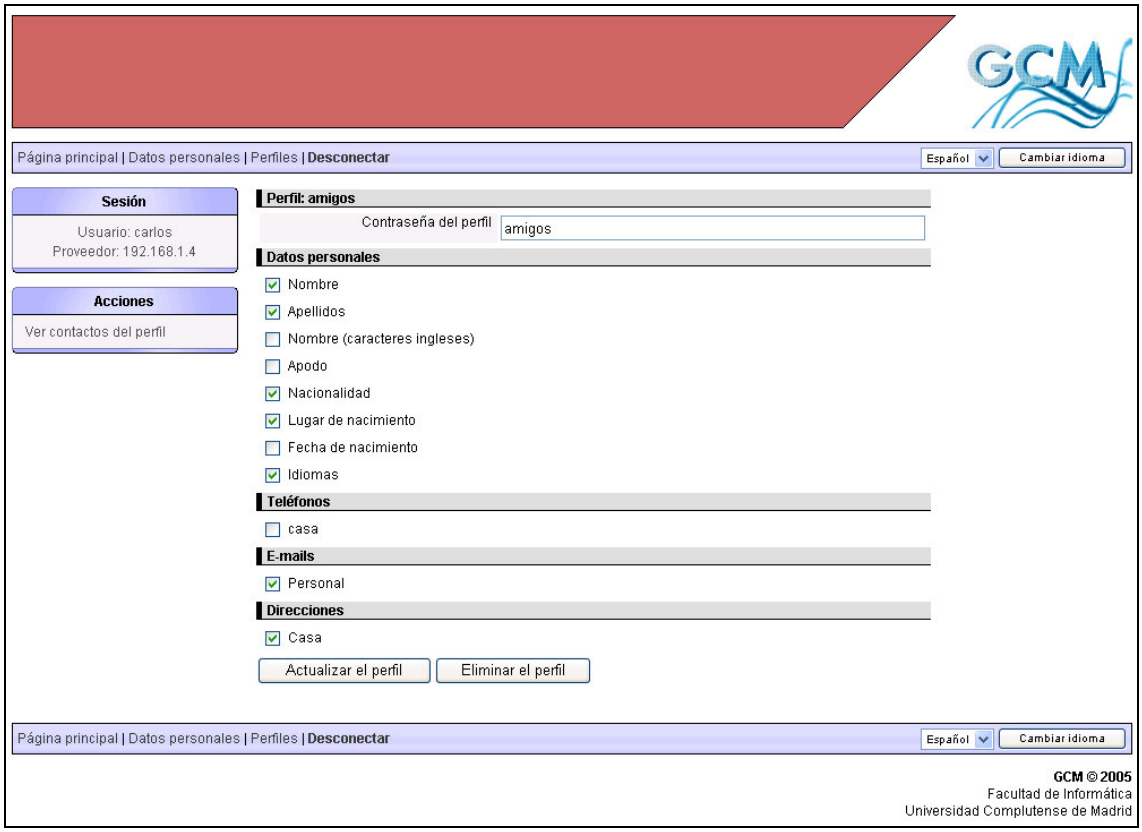


Ilustración 3. Pantalla de perfil de WebAdv

La zona de perfiles también permite ver la lista de usuarios con permiso de acceso al perfil, pudiendo eliminar cualquier de estos permisos.

6.2. Punto de acceso WAP

6.2.1. Resumen

Para el desarrollo del punto de acceso WAP, se ha usado la tecnología Java y el servidor Tomcat. Se ha utilizado la tecnología de páginas activas JSP con retorno de código WML 1.0, en un primer desarrollo, protocolo compatible con la práctica totalidad de los teléfonos móviles. Posteriormente se han intentado otros desarrollos en WML 2.0 (avanzado) para dispositivos más avanzados como móviles más actuales y Pocket PC.

El desarrollo del punto de acceso, finalmente se baso en el estándar WML 1.0. La decisión de desarrollar el más antiguo radica en su gran diferencia con todos los elementos de diseño Web que se usan actualmente. WML 1.2 y 2.0 se trata de híbridos, o incluso, en el caso del último de código XHTML puro con alguna restricción, por lo que consideramos que, no nos aportaba gran cosa su implementación.

6.2.2. Investigación de tecnologías

La investigación sobre la tecnología WAP comenzó prácticamente a la par con el comienzo del proyecto. La búsqueda se ha centrado sobre todo en Internet, a través de páginas de información, tutoriales y foros de tecnología WML: www.wmlclub.com, www.solocursos.net, www.programacion.net, etc., etc. La información más relevante, la hemos incluido, a modo de resumen, en esta memoria.

Las dificultades en el estudio de WAP se han centrado, sobre todo, en el aprendizaje de un nuevo lenguaje de programación, similar al XHTML, pero diferente en la estructura y en el planteamiento, con nuevos conceptos y etiquetas que nada tienen que ver con lo conocido para el lenguaje HTML. La segunda dificultad surgió en la búsqueda de simuladores potentes y fiables. El lenguaje WML 1.0, que ha sido el usado para el desarrollo de este punto de acceso, es "obsoleto", comparado con los nuevos desarrollos para móviles que se basan en el estándar XHTML+MP, y que permiten su visualización en navegadores web estándar.

Todas las especificaciones de estos estándares, se pueden en www.wapforum.org. El detalle de la información técnica que nos ofrece esta página supera con creces las necesidades del modulo que más tarde realizaríamos.

6.2.3. Tecnología WAP

El protocolo WAP es una especificación estándar para aplicaciones que operan sobre redes inalámbricas, con un énfasis particular cuando hablamos de dispositivos móviles, especialmente teléfonos. Los estándares fueron publicados por el WAP Forum, un cuerpo formado en Junio de 1997 por Ericsson, Nokia, Motorola y Unwired Planet. Posteriormente se han añadido, al menos un centenar de empresas, incluyendo IBM, Hewlett-Packard, VISA y Microsoft.

Muchas clases de dispositivos soportan el estándar WAP, desde simples dispositivos celulares a los más nuevos móviles multimedia y PDAs. Todos soportan algún modo de tecnología WAP, a pesar de diferir en la manera de presentación, en las pantallas y los teclados o dispositivos de entrada de datos. La principal misión de la especificación WAP es mantener un orden y proporcionar un entorno de desarrollo común para habilitar que las aplicaciones se ejecuten sobre diferentes plataformas.

El medio inalámbrico tiene sus particularidades y problemas, conocidos por todos los usuarios de teléfonos móviles. Precisamente para contrarrestar y convivir con este tipo de eventualidades

surge también el estándar WAP que, guardando el modelo de Internet, optimiza cada componente para un funcionamiento en entorno inalámbrico. Como ejemplo, siempre se guarda el estado de una sesión en caso de que la conexión se pierda, proveyendo formatos comprimidos para la transferencia de datos y soportando las mismas aplicaciones independientemente del dispositivo de entrada y salida.

6.2.4. La pila de protocolos WAP

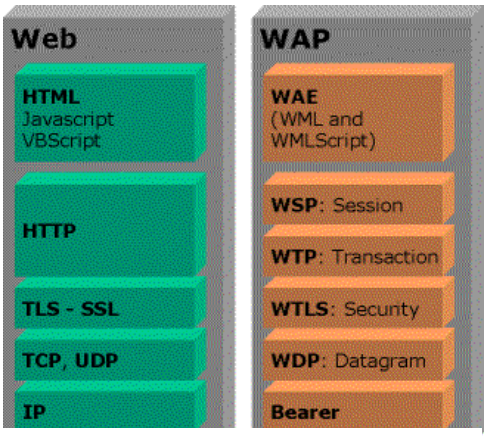


Gráfico 1. Pila de protocolos WAP

Con este título nos referimos a las diferentes partes que integran la especificación WAP. Al igual que el protocolo web tradicional (HTTP), los distintos componentes puede entendidos respectivamente como las distintas capas de una pila. La interacción del usuario se limita a las capas superiores y el hardware de comunicación se apoya en las inferiores.

Si nos fijamos en el conjunto de capas inferiores, éstas contienen varios protocolos (también llamados bearer protocols), que son, actualmente, parte de la especificación WAP, pero que, a la vez, proporcionan el enlace entre WAP y el hardware actual de comunicaciones:

- IP (Internet Protocol) y PPP (Point to Point Protocol) son los dos protocolos de más bajo nivel cuando se efectúa un acceso normal telefónico a Internet (Muchos móviles de primera generación funcionan con una llamada telefónica normal y enviando los datos WAP a través de un enlace de módem tradicional. IP y PPP se usan en este caso).
- SMS (Short Message Service) es una funcionalidad que proporcionan la práctica totalidad de los dispositivos móviles y que consiste en el envío y recepción de mensajes cortos de texto). Por esta vía se pueden transmitir datos binarios y también usarse con aplicaciones WAP.
- GPRS (General Packet Radio System) es un sistema de telefónia móvil más moderno que proporciona conexiones con más ancho de banda, y redes permanentes entre dispositivos celulares y otros dispositivos inalámbricos. En lugar de llamar por teléfono y conectarse directamente al servidor, un teléfono celular mantendrá una conexión permanente a Internet. GPRS está basado en IP.

La siguiente capa incluye varios protocolos de comunicación WAP de bajo nivel, entre los que se encuentran WTP (Wíreless Transaction Protocol), WTLS (Wíreless Transaction Layer Security) y WDP (Wíreless Datagram Protocol). WTP y WDP proporcionan el enlace real entre la capas superiores y las inferiores. Con WTLS tenemos servicios más seguros (encriptación y autenticación). Estos protocolos no son significativos para el programador habitual, a no ser que se esté diseñando algún tipo de navegador.

El segundo nivel más alto de la pila es el protocolo de comunicaciones de alto nivel, también llamado WSP (Wireless Session Protocol). Proporciona una "simulación" de http, protocolo usado entre servidores y navegadotes web.

La última capa es la WAE (Wireless Application Environment), y es parte de lo que el usuario final podrá ver cuando interactúe con el sistema. WAE proporciona un modelo similar a WWW para escribir aplicaciónes, habilitando a los programadores para que la transición de entornos sea sencilla.

La capa WAE incorpora varias características importantes, incluyendo urls como www.wap.net, o los MIME Content types. HTML se sustituye por WML (Wireless Markup Language),

JavaScript se sustituye por WMLScript. El formato natural de las imágenes será ahora el WBMP (Wireless Bitmap).

6.2.5. Del servidor al cliente

La utilización de un teléfono WAP es igual a la de un navegador Web: El usuario teclea para solicitar una URL. Pero, al contrario que los navegadores estándar que usan HTML para visualizar la información en la pantalla del ordenador, los teléfonos WAP utilizan WML, un lenguaje abierto desarrollado por el WAP Forum, que permite adaptarse a pequeños dispositivos de mano. Al igual que el HTML, WML se construye por medio de "tags" y permite la presentación de texto e imágenes, entrada de información y formularios.

El teléfono WAP utiliza las capacidades de información de conexiones inalámbricas convencionales para que el usuario realice peticiones al gateway WAP. EL gateway WAP convierte éstas en peticiones HTTP y las envía a través de Internet. Cuando el servicio requerido responde, el gateway WAP vuelve a enviar la información al teléfono WAP.

El gateway WAP es el núcleo de la plataforma WAP. Su capacidad para actuar en esta clase de teléfonos como un proxy HTTP, permite a los suscriptores acceder a cualquier site WWW. Algunos proveedores de información ofrecen igualmente servicios WML que usan WML para aprovechar la interfaz del teléfono WAP. Estos servicios pueden además iniciar la comunicación "impulsando" la información al gateway WAP, que como respuesta, transmite la misma a un teléfono WAP. Este proceso se denomina notificación.

Además de la translación HTML, la oferta de servicios del gateway varía. Estos pueden ser un servicio de protección de información por medio del mantenimiento de una base de datos de teléfonos WAP y sus privilegios de acceso, un servicio de fax que permitiese a los usuarios de teléfonos WAP mandar por fax contenido de un site Web a una máquina de fax local, o servicios de correo, organizadores o directorios. Todos ellos dependen de la suite de servicios que ofrezca cada gateway.

Ver el esquema de la Ilustración 4.

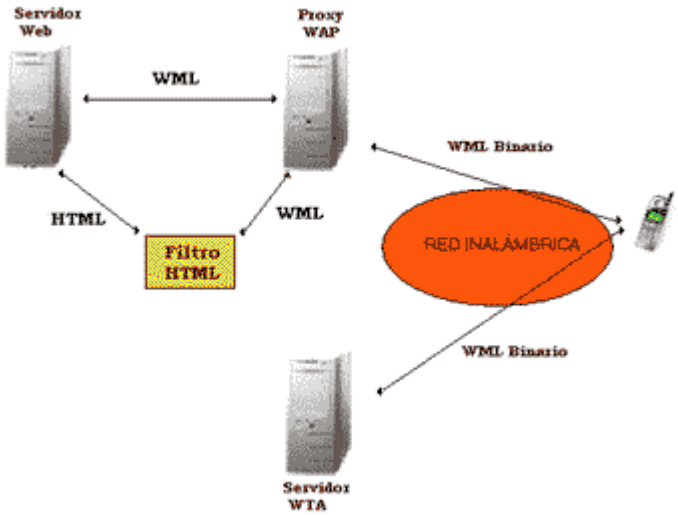


Ilustración 4. Esquema WAP

6.2.6. El gateway WAP

Básicamente podemos hablar dos funciones:

1. Traducir la petición WAP, escrita en WML, a una petición WWW, permitiendo así que el cliente WAP pueda realizar peticiones al servidor Web.

2. Codificar las respuestas del servidor a un formato binario de modo que sea entendible por el cliente WAP.

El Servidor WTA (Wireless Telephony Application) puede responder las peticiones WAP del cliente directamente; de este modo, permite ofrecer acceso WAP a determinadas características de la infraestructura de comunicaciones del operador de red.

Por otro lado tenemos el Servidor Web, que se comunica con el Proxy WAP de dos posibles modos:

1. Si el servidor Web proporciona un contenido WAP, como por ejemplo WML o WMLS, entonces no se necesita ningún filtro HTML.
2. Si el servidor Web proporciona un servicio WWW, como HTML, entonces se usa un filtro HTML para traducir el contenido WWW en uno WAP. Como se ve en el ejemplo, el filtro HTML, puede encargarse de traducir una respuesta HTML en una WML, y devolvérsela al Proxy WAP.

Ver la Ilustración 5.

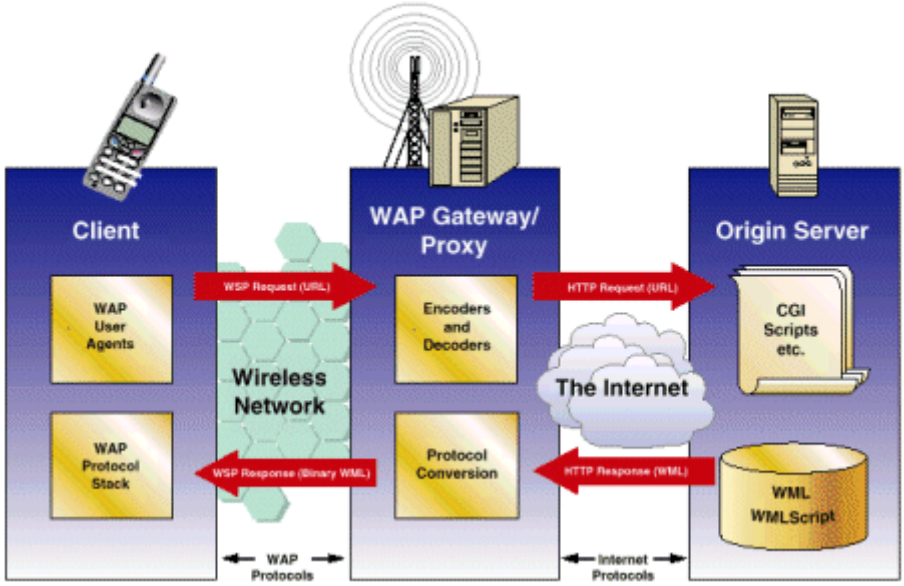


Ilustración 5. Gateway WAP

6.2.7. Comparativa operación WEB, operación WAP

A continuación, se muestra un comparativa del funcionamiento de un operación WEB e a la WAP, dado que la arquitectura de la plataforma WAP, está influida por la infraestructura y diseño de la WWW.

6.2.7.1. Operación Web

1. El usuario abre el navegador y especifica la URL
2. El navegador analiza la URL y envía una petición HTTP o HTTP segura (HTPPS) al servidor web.
3. El servidor Web analiza la petición y determina que recuperar. Si la URL especifica un archivo estático como en este ejemplo), el servidor Web lo recupera. Si la URL especifica un programa CGI, el servidor Web inicia el programa.
4. El servidor Web coloca un encabezado HTTP o HTTPS en el archivo estático o programa CGI y lo manda de vuelta al navegador.
5. El navegador interpreta la respuesta y despliega el contenido al usuario.

Ver la Ilustración 4.

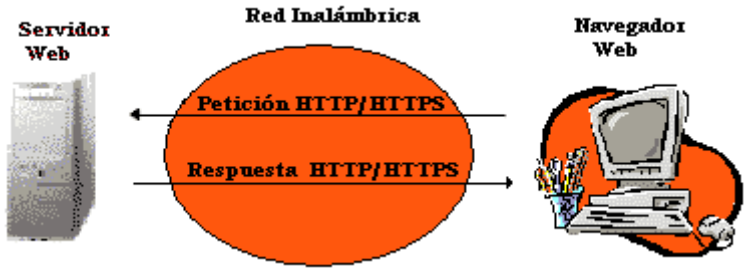


Ilustración 6. Operacion WEB

6.2.7.2. Operación WAP

Las transacciones WAP utilizan el mismo modelo básico, siendo la principal diferencia que el teléfono y el gateway WAP sustituyen en conjunto al navegador Web.

1. El usuario utiliza un teléfono WAP para solicitar una URL
2. El navegador WAP crea una petición que contiene la URL e información que identifique al suscriptor y las envía al gateway WAP.
3. El gateway WAP interpreta la petición, genera una petición convencional HTTP o HTTP Secure (HTTPS) y la envía al servidor Web.
4. El servidor Web interpreta a su vez la petición y determina que recuperar. Si la URL especifica un archivo estático, el servidor Web lo recupera. Si la URL especifica un programa CGI, el servidor Web inicia el programa.
5. El servidor Web coloca un encabezado HTTP o HTTPS en el archivo estático o programa CGI y lo manda de nuevo al gateway WAP.
6. El gateway WAP interpreta la respuesta, valida el WML, genera una respuesta (quitando el encabezado HTTP o HTTPS) y lo envía al teléfono WAP.
7. El navegador WAP interpreta la respuesta y despliega el contenido al usuario.

6.2.8. El lenguaje WML

WML es un lenguaje de marcas comprendido dentro del estándar XML 1.0, esto conlleva que WML debe cumplir con la sintaxis de XML 1.0. Vamos a describir brevemente los rasgos más importantes de esta sintaxis.

6.2.8.1. Sensible a mayúsculas/minúsculas

Todos los elementos de WML son sensibles a mayúsculas/minúsculas, esto incluye las etiquetas, los atributos, los identificadores, las variables...

6.2.8.2. El conjunto de carácteres

El conjunto de carácteres definido por defecto es el ISO/IEC-10646 que es el mismo que el Unicode 2.0 WAP soporta los siguientes subconjuntos de Unicode:

- UTF-8
- ISO-8859-1 o ISO Latin-1
- UCS-2

Se definen en la etiqueta

<?xml version="1.0" encoding="UTF-8"?>

6.2.8.3. Etiquetas

Todas las etiquetas en WML se escriben en minúsculas. Hay dos tipos de etiquetas, las contienen elementos, para lo cual hay una etiqueta de inicio y otra de fin. Los atributos de las etiquetas han de ir siempre en la etiqueta de incio.

<etiqueta> Inicio

</etiqueta> Fin

Y las etiquetas que no contienen elementos que tienen el siguiente formato:

<etiqueta/>

Ejemplo sencillo:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="t1" title="Tarjeta 1">
     - <p>Hola mundo !</p>
</card>
</wml>
```

Listado 11. Ejemplo de código WML

Vamos a explicar detenidamente el ejemplo:

`<?xml version="1.0"?>`

Indica que es un documento XML de versión 1.0 por lo tanto cumple todas las restricciones y reglas de los documentos XML.

`<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"`

`"http://www.wapforum.org/DTD/wml_1.1.xml">`

Indica el tipo de documento XML y donde localizar la especificación del tipo de documento.

`<wml>`

Indica que comienza un página WML.

`<card id="t1" title="Tarjeta 1">`

Indica que comienza una tarjeta que tiene como identificador "t1" y cuyo título es "Tarjeta 1"

`<p>`

Indica que comienza un párrafo de texto. A diferencia de HTML esta etiqueta es obligatoria si queremos escribir texto, además debe cerrarse con la correspondiente </p>

`Hola Mundo !`

Este es el texto que aparecerá en nuestro navegador.

`</p>`

Como señalamos anteriormente, con esta etiqueta indicamos que el párrafo ha terminado y no vamos a escribir más texto dentro de este párrafo.

`</card>`

Fin de la tarjeta.

</wml>

Fin de la página WML

6.2.8.4. Barajas y cartas

A las páginas WML se les suele llamar barajas porque están compuestas por cartas. Una carta es la unidad de información que un navegador WAP puede mostrar. El navegador nos permite pasar de una carta a otra dentro de la baraja para así poder acceder a todas las cartas.

Baraja:

Una baraja de cartas se marca con las etiquetas `<wml> ...</wml>` dentro de estas marcas irán todas las cartas de la baraja. Puede contener las etiquetas head, template y es obligatorio que al menos tenga una etiqueta card.

Carta:

Una carta es la unidad de información que se muestra en un navegador WAP, una carta puede contener texto, campos de datos, enlaces...

La etiqueta es `<card> ... </card>` y algunos de los atributos son title que nos permite indicar el título de la carta, id que nos proporciona una manera de identificar la carta. El atributo id es común para todos las etiquetas WML y nos permite identificar un elemento dentro de un documento WML.

Ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="t1" title="Tarjeta 1">
     - <p>Hola mundo !</p>
</card>
<card id="t2" title="Otra tarjeta">
     - <p>Bienvenido</p>
</card>
</wml>
```

Listado 12. Programa WML

6.2.8.5. Texto

El texto debe ir entre las etiquetas `<p>... </p>` ya que así está definido en el DTD de WML. Podemos incluir saltos de línea con la etiqueta `<br/>`.

La etiqueta `<p>` puede llevar los siguentes atributos:

`align=""` Puede contener los valores `left, right y center`. Indica la alineación del texto. Por defecto es `left`

`mode=""` Puede ser `warp o nowrap`. warp significa que el texto puede ir en varias líneas y nowrap quiere decir que el texto no puede ser roto en varias líneas. Por defecto es warp

Aunque los navegadores WAP estén muy limitados en lo referente al apartado visual podemos hacer algunos efectos en el texto. Estas son las etiquetas para dar formato al texto: `<em>, <strong>, <i>, <b>, <u>, <big>, <small>.`

6.2.8.6. Eventos y tareas

Los eventos y las tareas nos proporcionan un mecanismo para realizar acciones sobre las tarjetas, permitiéndonos navegar entre tarjetas y construir pequeños interfaces para dar más funcionalidad a las páginas.

El más importante de todos es quizás en elemento <do>, ya que nos permite asignar una tarea sobre una acción. La etiqueta <do> puede contener uno de las siguientes tareas, <go>, <prev>, <noop>, <refresh> que indican la acción se realiza sobre la etiqueta.
Los atributos más importantes que contiene la etiqueta <do> son: type="", label="" y name="".

type="". indica sobre qué botón del navegador se aplica la acción, las más comunes son "accept", "prev" y "help".

label="". Texto que aparece asociado a la acción.

name="". Nombre de la acción, es imprescindible si vamos a asignar más de una acción a un mismo tipo.

El contenido de la etiqueta <do> es la tarea que se realizará al seleccionar esa acción, y estas tareas pueden ser:

<go href=""/> Permite ir a la dirección indicada en el atributo href.

<prev/> Permite ir a la tarjeta anterior, en la historia del navegador.

<noop/> Es una acción que no realiza nada.

<refresh>...</refresh> Refresca el contenido de la tarjeta actual, volviéndola a pedir al servidor.

6.2.8.7. Imágenes en WML

El uso de imágenes en las navegadores WAP esta limitado, pero aun así es posible poner imágenes en dichas páginas.

La etiqueta para poner imágenes en las páginas WAP es <img/>, con ella podemos poner una imagen e indicar un texto alternativo por si nuestro navegador WAP no es capaz de representar dicha imagen.

La etiqueta <img/> tiene los siguientes atributos:

alt="". Texto alternativo que se visualiza si nuestro navegador no es capaz de visualizar la imagen.

src="". URL de la imagen. Normalmente esta ha de estar en formato .wbmp

vspace="numero". Espacio vertical en blanco entre la imagen y el resto de la página.

hspace="numero". Espacio horizontal en blanco entre la imagen y el resto de la página.

Align="". Puede ser top, middle o bottom indica la alineación de la imagens con respecto al texto.

top. Alineado a la parte superior.

middle. Alineado al centro.

bottom. Alineado con la parte inferior del texto.

height="numero". Altura de la imagen.

weight="numero". Anchura de la imagen.

Las imágenes deben ser de formato WBMP, un formato especialmente desarrollado para el entorno inalámbrico que consiste en imágenes sencillas en blanco y negro. Si queremos transformar alguna imagen existente lo tendremos que hacer con alguna aplicación que nos transforme a formato bmp, p. Ej.: WBMPCreator.

A grandes rasgos estas serían las características del lenguaje WML que más hemos usado para la programación del punto de acceso WAP de consulta.

En cualquier caso, y hasta ahora, sólo hemos hablado de páginas estáticas. Para la interacción que requiere la aplicación GCM necesitamos código dinámico.

6.2.9. Integración con el sistema GCM

Usando un conector distinto, por consiguiente un puerto TCP distinto, el punto de acceso WAP accederá al módulo Proveedor valiéndose del soporte del Tomcat. El servidor Tomcat, en este caso, deverá devolver código WML cuando se le soliciten las páginas. Para ello tendremos que modificar, dentro del archivo confs.http, los *mime type* admitidos e indicarle en cada archivo jsp que desarrollemos el código que tendrá que devolver. En el ejemplo adjunto podemos ver la cabecera que hay que hay que añadir para conseguir código WML.

Ejemplo:

```
<?xml version="1.0"?>
<%@ page contentType="text/vnd.wap.wml"%>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
     "http://www.wapforum.org/DTD/wml13.dtd">

<wml>

<%@ page import="proveedor_mensajes.*,proveedor_datos.*,pa.beans.*" %>
<%@ page import="proveedor_mensajes.respuestas.*" %>
<%@ page import="java.io.*" %>

<jsp:useBean               id="datosUsuario"               scope="page"
class="pa.beans.BeanDatosUsuario"/>


  <template>
    <!-- Template implementation here. -->
    <do type="prev"><prev/></do>
  </template>
```

Listado 13. Ejemplo de integración con GCM

Tenemos por tanto, archivos jsp, virtualmente servlets como motor, y que nos devuelven código WML que es interpretado por los navegadores WAP, es decir, los incluidos con los móviles, PDAs, etc.

6.2.10. Emuladores y herramientas de desarrollo utilizadas

En las primeras pruebas del lenguaje se emplearon kits de desarrollo de Nokia, los cuales permitían las pruebas en SDK´s que imitaban móviles de esa marca y que para los primeros pasos con la tecnología sin pasar a páginas dinámicas, fueron más que suficientes. Los SDK´s de Nokia son programas que muestran en pantalla un móvil a tamaño natura, su pantalla, su teclado y que interactúa con el usuario como si de un teléfono real se tratara. El aspecto, de estos emuladotes de Nokia, como se puede ver en la imagen adjunta (uno de los emuladotes que utilizamos, serie 3300), es espectacular y muy cómodo para hacer las pruebas, pero el kit de desarrollo que nos ofrecía era demasiado parco y con muy pocas herramientas de depuración y desarrollo, además de poco o nulo soporte al código dinámico.

6.2.11. Desarrollo final

Ilustración 7. Openwave

Optamos finalmente por desarrollar el punto de acceso usando JBuilder y creando páginas dinámicas donde las necesitáramos y código WML en las estáticas que no requiriesen de actividad. Las pruebas de programación las hemos hecho con emuladotes genéricos y muy extendidos en el mundo WAP, como son los de Openwave. Véase la imagen adjunta:

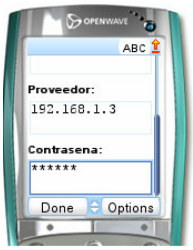
Para las pruebas con dispositivos reales usamos, primeramente un prototipo de Proveedor instalado en un servidor Web abierto en Internet, servidor Tomcat instalado en el equipo y punto de acceso WAP cargado. Desde un teléfono móvil muy antiguo Siemens ME45, pero con soporte WAP y GPRS, es decir, conectándonos a Internet, usando un Gateway público (Movistar), accedíamos a la dirección IP de ese servidor. A continuación esta misma prueba se realizó con una Pocket PC TSM500, con soporte telefónico y con conexión también GPRS. Todas estas primeras pruebas, aunque muy limitadas en el número de casos de uso fueron exitosas. Finalmente, por comodidad, velocidad y por la facilidad posterior de pasar "al mundo real", para el desarrollo de la aplicación de consulta completa optamos por el uso único y exclusivo del emulador WAP genérico Openwave V7 (www.openwave.com).

Hablando, de nuevo, del caso real, para un proveedor completo y sin necesidad de usar Gateways particulares (Movistar), necesitaríamos, tal y como hemos explicado en el resumen de la tecnología WAP, un servidor adicional conectado a línea telefónica, con las funciones de Gateway WAP. Los usuarios que quisieran conectarse al servicio tendrían que conectarse con sus dispositivos móviles al teléfono de ese servidor, para posteriormente ser comunicados con el resto de servidores o proveedores a los que hicieran consultas.

6.2.11.1. Funcionamiento del punto de acceso WAP

Adjuntamos a modo de ejemplo algunas capturas de pantalla para mostrar el funcionamiento del punto de acceso WAP.

Saludo del sistema, para pasar posteriormente a la Identificación de usuario:



El dispositivo móvil nos solicitará el nombre, el proveedor y nuestra contraseña para ese proveedor.

Tras la identificación, pasaremos a un modo de consulta que nos muestra en primer lugar nuestros datos personales.

Junto a los datos personales, el usuario podrá consultar los contactos a los que tiene acceso y los datos a los que el perfil correspondiente le permita acceder

6.2.12. Futuro

La evolución de la tecnología WAP es imparable y lo desarrollado para este proyecto no deja de ser ligeramente “obsoleto”, ya que los nuevos estándares WAP 1.2 y XHTML+MP (WAP 2.0), y sobre todo este último, rompen con las limitaciones que hemos puesto de manifiesto en la explicación de la tecnología.

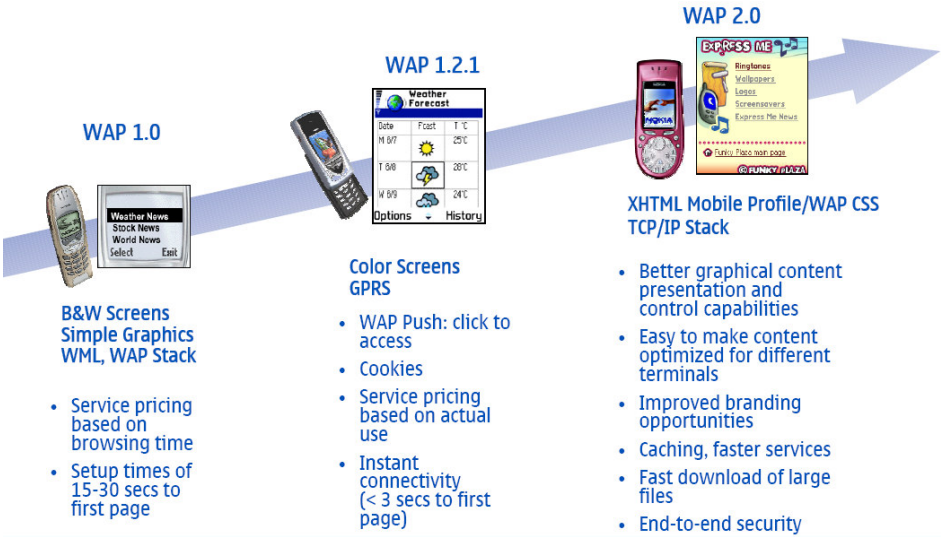


Ilustración 8. Evolución WAP

En un sistema hipotético futuro también se podrían implementar otras funcionalidades, como posibilidad de consultar al sistema enviando SMS, posibilidad de modificación de datos, crear cuentas desde dispositivos móviles, enviar contactos a otros usuarios usando SMS, etc., etc.

Los últimos desarrollos nos hablan de un lenguaje similar al HTML estándar, aunque mucho más limitado y *parametrizado*, con menos etiquetas, pero con una funcionalidad muy alta. Paralelamente a estos avances, han sido también la aparición de dispositivos móviles cada vez más potentes culminando en los últimos tiempos con la tecnología UMTS que permite la interacción multimedia de video y audio entre dispositivos móviles.

6.3. Punto de acceso Java

6.3.1. Resumen

Para el desarrollo del punto de acceso de Java se ha usado el mismo entorno de desarrollo que para el proveedor, el Jbuilder. Con la librería javax.swing y java.awt, herramientas gratuitas que nos proporciona el lenguaje Java se han creado todas las ventanas y entornos necesarias para construir el módulo de consulta al sistema GCM desde una aplicación Java.

A modo de ejemplo sólo se han implementado algunos casos de uso, todos de consulta. La ampliación al resto es inmediata, además de la ampliación a cualquier funcionalidad que se requiriese, una vez obtenidos los datos de cualquier contacto.

6.3.2. Investigación de tecnologías

La investigación acerca del entorno awt y swing de java se ha centrado sobre todo en manuales y tutoriales obtenidos en Internet, y en el propio conocimiento y experiencia que ya tenemos sobre el lenguaje. La propia facilidad que Jbuilder ofrece en la programación de estos entornos también ha ayudado a que el desarrollo haya sido más rápido de lo que, en principio, se preveía. Los designers de Jbuilder, los distintos componentes (clases java) que nos han ido haciendo falta, jlabels, jbuttons, containers, etc., han sido conceptos familiares en la programación de este punto de acceso.

6.3.3. Packages Swing y AWT

El Abstract Windows Toolkit es una colección de clases orientadas a la construcción de interfaces gráficas de usuario (GUI) en Java.

Los componentes son aquella serie de objetos que pueden formar parte de nuestra interfaz como botones, menús, barras de desplazamiento, cajas, áreas de texto... Los componentes tienen que estar situados obligatoriamente en un contenedor de componentes (container). Los eventos son una forma de comunicar al programa todo lo que el usuario, mediante el ratón y el teclado, esta realizando sobre los componentes.

Cuando hacemos algo con algún componente se produce un determinado tipo de evento que el sistema operativo transmite al AWT. Este reacciona creando un objeto de una determinada clase de evento, derivada de AWTEvent.

El evento tendrá que ser gestionado por algún método y esto se consigue gracias a que el modelo de eventos de Java se basa en que los objetos sobre los que se producen los eventos (event sources) "avisan" a los objetos que gestionan los eventos (event listeners) para que actúen en consecuencia. Los objetos gestores de eventos deben de disponer de los métodos adecuados para saber responder. Java obliga a los event listeners a implementar los métodos de las interfaces Listener que Java proporciona para cada tipo de evento. Esto es que Java dispone de una interfaz Listener para cada evento con unos determinados métodos (cabeceras) que el usuario tendrá que implementar en sus objetos event listeners dentro de su aplicación interactiva.

Componentes y eventos soportados por el AWT:

- Todos los componentes excepto los contenedores de más alto nivel (Window y sus descendientes) deben de estar dentro de un contenedor.
- Un contenedor se puede poner dentro de otro contenedor.
- Un componente solo puede estar dentro de un contenedor.

Todos los componentes del AWT de Java son objetos que pertenecen a una jerarquía de clases según se muestra en la Ilustración 9.

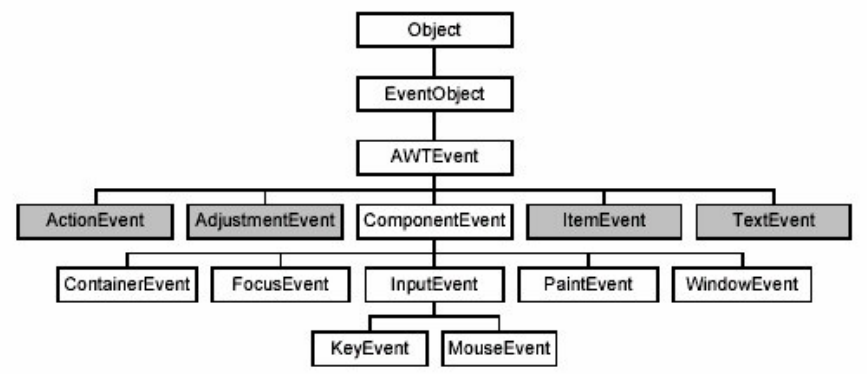


Ilustración 9. Jerarquía de componentes de AWT

Para añadir un componente a un contenedor se utiliza la el método add () de la clase Container. containerName.add (componentName);

Todos los eventos de AWT son objetos de clases que pertenecen a una jerarquía como la de la figura. Las clases de la jerarquía se encuentran se encuentran definidas en el package java.awt.event

6.3.4. Jerarquía de eventos en AWT

En AWT existen dos tipos de eventos, los eventos de alto nivel y los eventos de bajo nivel. Los eventos de alto nivel suelen implicar muchos de bajo nivel.

Los de alto nivel son:

- ActionEvent, clicar sobre un botón o elegir un elemento del menú.
- AdjustmentEvent, mover las barras de desplazamiento.
- ItemEvent, elegir valores.
- TextEvent, cambiar texto.

Los de bajo nivel son:

- ComponentEvent, eventos elementales relacionados con componentes.
- ContainerEvent, eventos elementales relacionados con contenedores.
- KeyEvent, eventos relacionados con las pulsaciones sobre el teclado.
- MouseEvent, eventos relacionados con las pulsaciones del ratón.
- FocusEvent, eventos relacionados con el focus.
- WindowEvent, eventos elementales relacionados con ventanas.

6.3.5. Relación entre componentes y eventos

Es muy importante conocer cual es la relación que existe entre eventos y componentes ya que a la hora de programar una interfaz gráfica de usuario necesitamos esta información. Por eso a continuación mostramos una tabla en donde se relaciona cada componente con sus correspondientes eventos más una pequeña explicación.

Button	ActionEvent	Clicar en el botón
Checkbox Seleccionar o deseleccionar un item	ItemEvent	Seleccionar o deseleccionar un item
CheckboxMenuItem	ItemEvent	Seleccionar o deseleccionar un item
Choice	ItemEvent	Seleccionar o deseleccionar un item
Component	ComponentEvent	Mover, cambiar tamaño, mostrar u ocultar un componente
	FocusEvent	Obtener o perder el focus
	KeyEvent	Pulsar o soltar una tecla
	MouseEvent	Pulsar o soltar un botón del ratón; entrar o salir de un componente; mover o arrastrar el ratón

Container	ContainerEvent	Añadir o eliminar un componente de un container
List	ActionEvent	Hacer List doble click sobre un item de la lista
	ItemEvent	Seleccionar o deseleccionar un item de la lista
MenuItem	ActionEvent	Seleccionar un item de un menú
Scrollbar	AdjustementEvent	Cambiar el valor de la scrollbar
TextComponent	TextEvent	Cambiar el texto
TextField	ActionEvent	Terminar de editar un texto pulsando Intro
Window	WindowEvent	Acciones sobre una ventana: abrir, cerrar, iconizar, restablecer e iniciar el cierre

Tabla 5. Tabla de componentes y eventos en AWT

Hay que considerar que porque un componente no se relacione con ningún evento esto no quiere decir que no los pueda recibir. Una clase que recibe un evento puede transmitírselo a sus subclases.

6.3.6. Pasos para crear una interfaz gráfica con AWT

Unos pasos sencillos para la elaboración de una primitiva interfaz con AWT serían los siguientes:

- Crear una clase con el método main () que es donde empezará a correr nuestro programa.
- Crear una clase derivada de Frame que responda al evento WindowClosing ().
- Añadir al contenedor todos los componentes que queremos que tenga nuestra interfaz gráfica. containerName.add (componentName);
- Implementar los objetos gestores (event listeners) que responden a nuestros eventos.

```
1. import java.awt.*;
2. import java.awt.event.*;
3. class CerrarVentana extends WindowAdapter
4. {
5. public void windowClosing(WindowEvent e)
```

```
6. {
7. System.exit(0);
8. }
9. }
10. class Ventana extends Frame
11. {
12. public Ventana()
13. {
14. CerrarVentana cv = new CerrarVentana();
15. addWindowListener(cv);
16. Button Boton = new Button();
17. this.add(Boton);//se puede omitir this
18. Boton.setLabel("Boton");
19. setSize(400, 400);
20. setTitle("Ventana");
21. setVisible(true);
22. }
23. public static void main(String args[])
24. {
25. Ventana mainFrame = new Ventana() ;
26. }
27. }
```

En las líneas 1 y 2 se importa a la aplicación los *packages* de clases java.awt y java.awt.event para incluir las clases de componentes y eventos respectivamente.

De la línea 3 a la 9 es donde se declara la clase gestora de eventos. Esta clase es una derivada de WindowAdapter osea que es una clase Adapter que se comentará más adelante.

De la 10 a la 22 es donde se define el *container*. Dentro de él en las líneas 14 y 15 se registran los eventos que nos interesan para la aplicación (de momento la aplicación solo responde al evento windowClosing()). También dentro del *container* se ha añadido un componente Button en la línea 17 y se han utilizado unos métodos heredaros de la jerarquía de clases de componentes y eventos en las líneas 19, 20, 21

De la 23 a la 26 se ha declarado el método main () que es donde comenzará a ejecutarse la aplicación. Es en este método donde se instancia la clase Ventana (contenedor de la aplicación).

El resultado es un botón que no desencadena ninguna acción, redimensionable y que ocupa toda la ventana de programa.

6.3.7. Interfaces Listener

La forma de gestionar eventos por AWT es que cada objeto que recibe un evento (event source) registra un objeto que lo gestione (event listener).

Para esto se utiliza el método addEventObject de la siguiente manera, eventSourceObject.addEventListener(eventListenerObject);

EventSourceObject es el objeto que recibe el evento y registra con el método addEventListener al objeto eventListenerObject para que gestione el evento cuando se produzca.

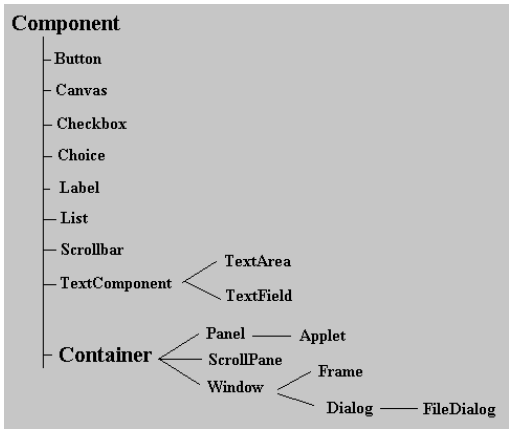
Las interfaces Listener son una forma de llevar esto a cabo. El eventListenerObject debe de implementar la interfaz Listener para el evento que se desee gestionar.

6.3.8. Clases componentes y eventos

A continuación vamos a ver brevemente las clases componentes y eventos de AWT. Hay que saber que cuando se produce un evento el AWT de Java genera automáticamente un objeto de ese evento que tendrá métodos que el usuario podrá utilizar.

- Clase Component : Una clase abstracta de la que derivan todas las clases del modelo de componentes de AWT.
- Clase EventObject y AWTEvent : Todos los métodos de las interfaces Listener relacionadas con el AWT tienen como argumento único una clase que desciende de la clase java.awt.AWTEvent. La clase AWTEvent no define ningún método pero hereda de EventObject el método getSource que devuelve una referencia al objeto que produjo el evento.
- Clase ComponentEvent : Se genera cuando un Component de cualquier tipo se muestra, se oculta o cambia de posición o tamaño.
- Clases InputEvent, MouseEvent, MouseMotionEvent: De la clase InputEvent descienden los eventos del ratón y teclado detectando si los botones del ratón o las teclas especiales han sido pulsadas. Se produce un MouseEvent cada vez que el ratón entra o sale de un componente visible en la pantalla, al clicar o cuando se pulsa o suelta un botón del ratón. Se produce un MouseMotionEvent cuando se mueve el ratón donde quiera que sea.
- Clase FocusEvent: Se produce cuando un componente pierde o gana el focus.
- Clase Container: Una clase muy general. Nunca se crea un objeto de esta clase pero heredan susmétodos las clases Frame y Panel.
- Clase ContainerEvent: Se genera cada vez que un Component se añade o se retira de un Container. Este evento solo tiene papel de aviso no es necesario gestionarlo.
- Clase Window: Los objetos de la clase Window son ventanas de máximo nivel sin bordes y sin barras de menú. Son más interesantes las clases que derivan de ella Frame y Dialog.
- Clase WindowEvent: Se produce cada vez que se abre, cierra, iconiza, restaura, activa o desactiva una ventana.
- Clase Frame: Es una ventana con un borde y que puede tener una barra de menús. Si una ventana depende de otra ventana es mejor utilizar una Window que un Frame.
- Clase Dialog: Es una ventana que depende de otra ventana (una Frame) si una Frame se cierra se cierra también los Dialogs que depende de ella. Si se minimiza sus Dialogs desparecen; si se restablece sus Dialogs aparecen de nuevo. Por defecto los Dialogs son no modales es decir, no requieren atención inmediata del usuario.
- Clase FileDialog: Muestra una ventana de dialogo en la cual se puede seleccionar un fichero. Las constantes enteras LOAD y SAVE definen el modo de apertura del fichero.
- Clase Panel: Un panel es un Container de propósito general se puede utilizar tal cual para contener otras componentes y para crear sub-clases de finalidad mas especificas. No tiene métodos propios y suele utilizar los heredados de Component y Container. Un Panel puede contener otros Panel una gran ventaja respecto a los otros tipos de containers.

- Clase Button: Lo más importante es que al clicar sobre él se genera un evento de la clase ActionEvent. El aspecto de un Button depende de la plataforma pero la funcionalidad es la misma. Se pueden cambiar el texto y el color del Button si se desea.
- Clase ActionEvent: Estos eventos se producen al clicar con el ratón en un botón (Button), al elegir un comando de un menú (MenuItem), al hacer doble clic en un elemento de una lista (List), y al pulsar intro para introducir texto en una caja de texto (TextField).
- Clase Canvas:Una Canvas es una zona rectangular de pantalla en la que se puede dibujar y en la que se pueden generar eventos. Las Canvas permiten realizar realizar dibujos, mostrar imágenes y crear componentes a medida de modo que muestren un aspecto similar en todas las plataformas.
- Component Checkbox y Clase CheckboxGroup: Los objetos de la clase Checkbox son botones de opción o de selección con dos posibles valores on y off. La clase CheckboxGroup permite la opción de agrupar varios Checkbox de modo que uno y solo uno este en on. Al cambiar la selección de un Checkbox se produce un ItemEvent.
- Clase ItemEvent: Se produce un ItemEvent cuando ciertos componentes cambian de estado (on/off).
- Clase Choice: Permite elegir un item de una lista desplegable los objetos Choice ocupan menosespacio en pantalla que los Checkbox. Al elegir un item se genera un ItemEvent.
- Clase Label: La clase Label introduce en un container un texto no editable y no seleccionable.
- Clase List:Viene definida por una zona de pantalla con varias líneas de las que se muestran solo algunas y entre las que se puede hacer una selección simple y múltiple. Las List generan eventos de la clase ActionEvent e ItemEvents.
- Clase Scrollbar: Es una barra de desplazamiento con un cursor que permite introducir y modificar valores, con pequeños y grandes incrementos. Al cambiar el valor de la Scrollbar se produce un AdjustementEvent.
- Clase AdjustementEvent: Hay cinco tipos de AdjustementEvent: track (se arrastra el cursor de la Scrollbar), unit increment y unit decrement (se clica en las flechas de la Scrollbar), block increment y block decrement (se clica encima o debajo del cursor).
- Clase ScrollPane: Es como una ventana de tamaño limitado en la que se puede mostrar un componente de mayor tamaño con dos Scrollbars que son visibles solo si son necesarias por defecto.
- Clase TextArea y TextField: Ambas componentes se heredan de la clase TextComponent y muestran texto seleccionable y editable. TextArea ofrece posibilidades de edición de texto seleccionables además puede estar compuesta de varias líneas. No se pueden crear objetos de la clase TextComponent porque su constructor no es public. Reciben eventos TextEvent y todos los de sus super-clase.
- Clase TextEvent: Se produce un TextEvent cada vez que cambia algo en un TextComponent. Se puede desear evitar ciertos caracteres y para eso es necesario gestionar los eventos correspondientes.
- Clase KeyEvent: Se produce un KeyEvent al pulsar sobre el teclado. Hay dos tipos key-typed que representa la introducción de un carácter Unicode; y key-pressed y key-released que representan pulsar o soltar una tecla.



6.3.9. Jerarquía AWT

Menús: Los menús de Java no descienden de Component sino de MenuComponent pero tienen un comportamiento similar pues aceptan Events. Para crear un menú se debe crear primero una MenuBar; después se crean los Menus y los MenuItem. Los MenuItems se añaden al Menu correspondiente; los Menus se añaden a la MenuBar y la MenuBar se añade a un Frame. También puede añadirse un Menu a otroMenu para crear un sub-menú, del modo que es habitual en Windows. La clase Menu es subclase de MenuItem. Esto es así precisamente para permitir que un Menu sea añadido a otro Menu.

- Clase MenuShortcut: Derivada de Object, representa las teclas aceleradoras que pueden utilizarse para activar los menús desde teclado, sin ayuda del ratón.
- Clase Menubar: Es un contenedor de objetos Menu.
- Clase Menu: El objeto Menu define las opciones que aparecen al seleccionar uno de los menús de la barra de menús. En un Menu se pueden introducir objetos MenuItem, otros Menu, objetos CheckboxMenuItem y separadores.
- Clase MenuItem: Representa las distintas opciones de un menú. Al seleccionar un objeto MenuItem se generan eventos del tipo ActionEvent.
- Clase CheckboxMenuItem: Son items de un Menu que pueden estar activados o no. No generan ActionEvent, generan ItemEvent de modo similar a la clase CheckBox.
- Menus Pop-Up: Son menús que aparecen en cualquier parte de la pantalla al cliclar con el botón derecho del ratón (pop-up trigger) sobre un componente determinado.

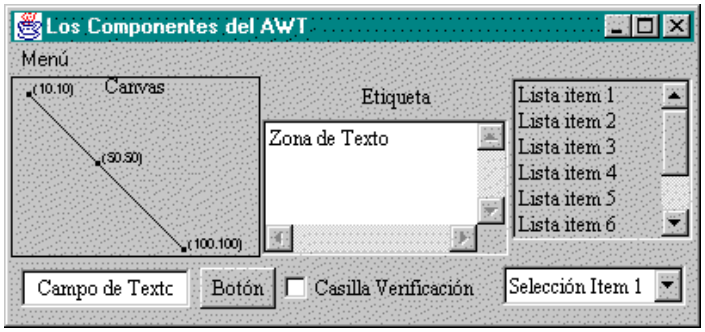


Ilustración 10. Componentes de AWT

6.3.10. Swing

Swing es un paquete *Java* para la generación de la GUI en aplicaciones reales de gran tamaño, viene a complementar y ampliar al modelo de componentes y eventos de *AWT*, basándose en este. Es una de las *API*'s del *JFC* (*Java Foundation Classes*).

Swing era el nombre clave del proyecto que desarrolló los nuevos componentes que vienen a sustituir o complementar a los de AWT. Aunque no es un nombre oficial, frecuentemente se usa para referirse a los nuevos componentes y al API relacionado. Está inmortalizado en los nombres de paquete del API Swing, que empiezan con javax.swing. Esta versión de las JFC fue publicada como JFC 1.1, que algunas veces es llamada 'Versión Swing'. El API del JFC 1.1 es conocido como el API Swing.

Swing constituye la característica más importante que se ha añadido a la plataforma 1.2, como Sun ha preferido llamarla, Java2. Con esta última incorporación por fin se completa totalmente la parte gráfica de la programación en Java, ofreciendo al programador acceso a todas las características existentes en un entorno gráfico actual, así como un conjunto de componentes gráficos completo y fácilmente ampliable con el que construir la interfaz gráfica de usuario, o GUI, de nuestras aplicaciones y applets, de una forma totalmente transparente e independiente de la plataforma en la que ejecutemos nuestro código.

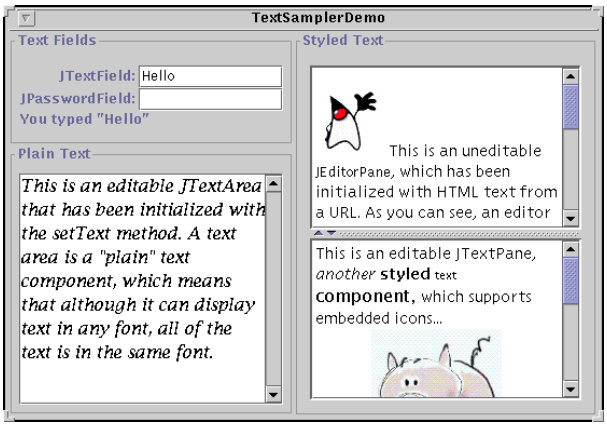


Ilustración 11. Algunos de los componentes Swing

El modelo de eventos que utiliza *Swing* es el mismo que *AWT*, el de *Java* 1.1, añadiendo algunos nuevos eventos para los nuevos componentes. Utilizando igualmente las interfaces *Listener*, las clases *Adapter* o las clases anónimas para registrar los objetos que se encargaran de gestionar los eventos.

En principio implementa de nuevo todos los componentes gráficos existente en el AWT, pero en este caso con implementaciones ligeras, o lighweight, con todas las ventajas que esto implica. Además añade nuevas y útiles funcionalidades a estos componentes, tales como la posibilidad de presentar imágenes o animaciones en botones, etiquetas, listas o casi cualquier elemento gráfico.

Este paquete nuevo está enteramente basado en AWT y más específicamente en el soporte para interfaz de usuario ligero. Debido a ello y a ser puro Java, es posible hacer aplicaciones basadas en Swing desde la plataforma 1.1.5 y que funcione sin ningún problema con la JVM de dicha plataforma, así como con la incluida junto con los navegadores más actuales, lo que asegura que un applet realizado usando estos nuevos componentes funcionará sin problemas en dichos navegadores.

Entre los componentes que se incorporan en Swing está la reimplementación de todos los componentes gráficos existentes en AWT y que, para no confundir con los antiguos, ahora empiezan todos por J. Así en vez de Button, tenemos JButton. La mayor diferencia entre los componentes AWT y los componentes Swing es que éstos últimos están implementados sin nada de código nativo. Esto significa que los componentes

Swing pueden tener más funcionalidad que los componentes AWT, porque no están restringidos al denominador común, es decir las características presentes en cada plataforma. El no tener código nativo también permite que los componentes Swing sean vendidos como añadidos al JDK 1.1, en lugar de sólo formar parte del JDK 1.2.

Incluso el más sencillo de los componentes Swing tiene capacidades que van más allá de lo que ofrecen los componentes AWT. Por ejemplo:

- Los botones y las etiquetas Swing pueden mostrar imágenes en lugar de o además del texto.
- Se pueden añadir o modificar fácilmente los bordes dibujados alrededor de casi cualquier componente Swing. Por ejemplo, es fácil poner una caja alrededor de un contenedor o una etiqueta.
- Se puede modificar fácilmente el comportamiento o la apariencia de un componente Swing llamando a métodos o creando una subclase.
- Los componentes Swing no tienen porque ser rectangulares. Por ejemplo, los botones pueden ser redondos.
- Bordes complejos: Los componentes pueden presentar nuevos tipos de bordes. Además el usuario puede crear tipos de bordes personalizados.

6.3.11. Integración con el sistema GCM

Usando un conector distinto, por consiguiente un puerto TCP distinto, el punto de acceso Java accederá al módulo proveedor valiéndose únicamente del propio estándar y protocolo desarrollado y pensado para el sistema GCM. Porque se decidió así, este punto de acceso usa los paquetes de mensajería del propio proveedor, es decir, se comunica con el proveedor usando directamente el protocolo interno GCM. Esto lo hemos podido implementar porque se trata de una aplicación desarrollada en Java, mismo lenguaje de programación que se ha usado para el desarrollo de nuestro proveedor.

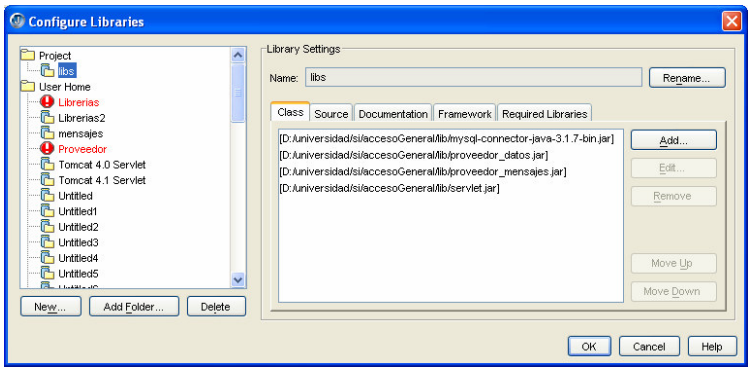
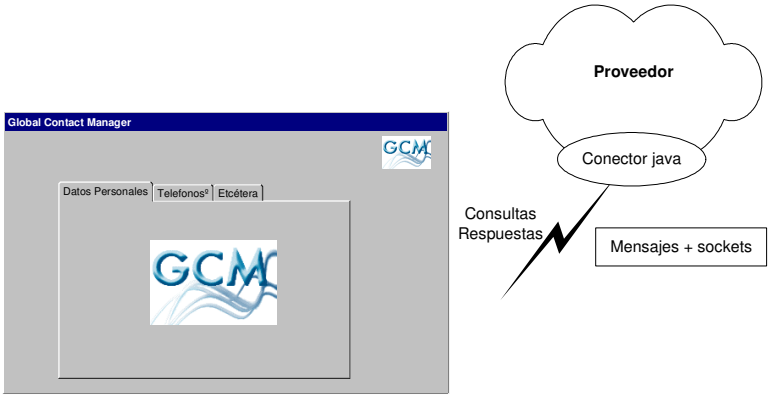
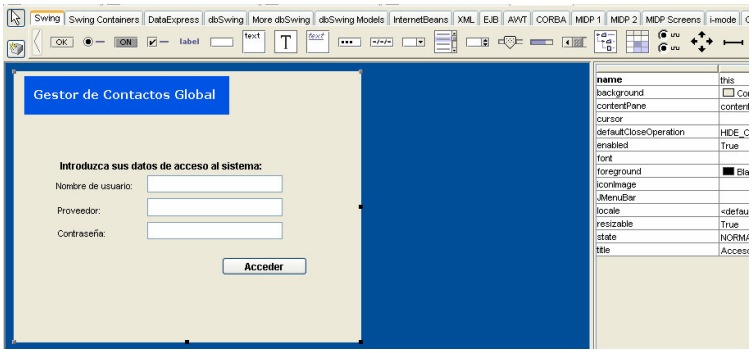


Ilustración 12. Librerías usadas para la aplicación

El envío y recepción de mensajes se ha implementado siguiendo el protocolo GCM, es decir usando sockets con TCP/IP.



Todas las ventanas se han diseñado usando los Designers que nos ofrece Jbuilder:



Las únicas dificultades que nos hemos encontrado han sido, por un lado, la gestión de los distintos containers que, debido a la gran variedad de datos que el sistema puede ofrecer para un mismo usuario, estos containers deben ser dinámicos y adaptarse a cada consulta. El soporte de este tipo de funcionalidad no esta incluido en estos avanzados diseñadores, que sin embargo sí que son perfectos para el diseño unitario de componentes.

El uso de arrays de componentes Swing y awt, ha sido algo habitual. El tamaño de estos arrays se asignaba dinámicamente en tiempo de ejecución y tras la lectura de los datos correspondientes de usuario.

Otras dificultades han sido las inherentes al sistema de programación visual, y en particular a la orientación a objetos de java, y es el gran número de clases que hay que crear e integrar (una por ventana), teniendo mucho cuidado con el flujo natural del programa en todas las direcciones.

Ilustración 13. Pestañas adaptables

Los datos son presentados por el programa en forma de pestañas adaptables. Es un módulo de consulta, es decir, nos va a permitir, consultar toda la información del usuario que se validase en el sistema, datos personales, perfiles, contactos, datos accesibles de esos contactos, etc.

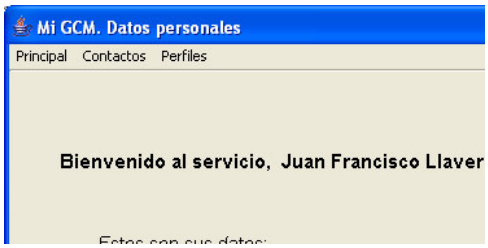


Ilustración 14. Módulo de consulta

6.3.12. Futuras aplicaciones

En software, informática y nuevas tecnologías no se puede hablar de un futuro predecible, y menos si se trata de lenguajes de programación que en principio están limitados por lo que los desarrolladores del lenguaje quieran ofrecer a los programadores.

En este caso, el proveedor, siempre que se siga el protocolo correspondiente, sera accesible por cualquier aplicación con medios de seguir ese protocolo, es decir, capaz de abrir sockets y enviar y recibir consultas estandarizadas en XML.

En el caso de aplicaciones java, la situación se facilita ya que podemos aprovechas las librerías de comunicación del propio proveedor para facilitar el tratamiento de los mensajes. En cuanto a posibilidades, teniendo una aplicación y disponiendo de los datos, se podrían realizar gestión, facturación, envíos personalizados a mail o dirección física, gestor telefónico, etc., lo que el sistema nos proporcionaría es un acceso a datos personales muy avanzado y que nos permitiría cualquier tipo de tratamiento sobre esos datos.

CAPÍTULO 7. Aspectos pendientes

Una vez finalizado el proyecto, todavía existen detalles del sistema que habría que ampliar.

En primer lugar, el procedimiento de comunicación del protocolo debería mejorarse utilizando sockets SSL que encriptasen todos los datos transmitidos entre los proveedores, además de usar certificación para asegurar la identificación de ambas partes. Con estas medidas se conseguiría una mejora sustancial de la seguridad del servicio.

También se debería ampliar el conjunto de datos que contempla el protocolo, de forma que se pudieran compartir más tipos de datos para mejorar el servicio que se ofrece a los usuarios. Posible tipos de datos serían: Imágenes (retratos de los contactos), datos de número de cuentas, currículo vital, etc.

También sería necesario introducir una serie de contadores en aquellas clases que puedan quedarse a la espera de una respuesta, estos contadores de tiempo se encargaría de devolver un respuesta de 'Respuesta fuera de tiempo' en caso de que se excediera un cierto límite esperando. Una de las funcionalidades de estos contadores sería eliminar las tareas que se queden almacenadas en e proveedor por que no les llega la respuesta necesaria.

En cuanto el punto de acceso web, se debería mejorar la compatibilidad con los diversos navegadores web más utilizados, ya que en algunos, por poner un ejemplo el Mozilla Firefox no detecta automáticamente el formato UTF8.

También se debería controlar mejor la aparición de caracteres potencialmente peligrosos para HTML en los datos de usuarios que se muestren, ya que a algunos caracteres ha de dárseles cierto formato para ser mostrados en las páginas HTML sin que rompan la estructura de la página, dichos caracteres son las dobles comillas, '"', que pueden terminar inesperadamente los atributos de las etiquetas; y los caracteres menor, '<' que pueden introducir nuevas etiquetas o cerrar las existentes.

APÉNDICE A. Glosario

A.1. Usuario

- **Usuario**: Persona física que hace uso del servicio, cada usuario ha de tener una cuenta en un servidor con un nombre valido para el sistema.
- **Nombre de usuario**: Nombre que identifica al usuario, ha de cumplir un determinado formato: Nombre dentro del servidor, #, nombre del servidor.
- **Nombre de servidor**: Es el dominio o dirección IP que utiliza cada servidor del sistema.
- **Contacto**: Usuario sobre el se tiene derecho de acceso a alguno de sus perfiles.
- **Datos del usuario**: Datos propios de un usuario, que este puede modificar o ampliar.
- **Perfil**: Registro que indica el conjunto de datos del usuario propietario al que da acceso, un perfil tiene una contraseña que debe ser rellenada por cualquier usuario que quiera obtener el permiso de acceso al perfil, un perfil tiene una lista con los contactos que actualmente tienen derecho de acceso al perfil.
- **Lista de contactos**: Conjunto de los contactos de un usuario distribuidos en grupos.
- **Contraseña de un perfil**: La contraseña de un perfil es requisito indispensable para poder obtener el permiso de usar un perfil, sin embargo una vez obtenido no es necesario seguir recordando la contraseña, esta se puede cambiar sin que afecte a los usuarios que ya tienen derecho de acceso al perfil.
- **Datos personalizados**: Campos especiales de los datos del usuario a los que pueden ser usados para contener cualquier información que el usuario considere necesaria.

A.2. Técnico

- **Servicio GCM**: Sistema completo formado por el conjunto de todos los servidores.
- **Servidor**: Cada uno de los sistemas que forman la red del servicio.
- **Servidor tipo**: Servidor implementado en el proyecto como ejemplo del funcionamiento del servicio.
- **Proveedor**: Capa de negocio del servidor tipo.
- **Punto de acceso**: Cada una de las aplicaciones del servidor tipo que dan acceso al sistema para el usuario.
- **Protocolo XML**: Sistema de comunicación entre los servidores del servicio.
- **Consulta**: Mensaje del protocolo cuyo fin es realizar una solicitud a otro servidor.
- **Respuesta**: Mensaje del protocolo consistente en una contestación a una consulta recibida anteriormente.
- **Tarea**: Cada instancia de un caso de uso del servicio, cada tarea tiene un identificador único en todo el servicio.

APÉNDICE B. Tabla ISO 3166-3

A continuación se expone la tabla ISO 3166-3 de códigos de tres letras de países. Hay un código especial (el XXX) que aunque no pertenece al ISO, se emplea en GCM para indicar que no se ha especificado y debe emplearse como valor por defecto del tipo *País*.

AFGHANISTAN	AFG
ALBANIA	ALB
ALGERIA	DZA
AMERICAN SAMOA	ASM
ANDORRA	AND
ANGOLA	AGO
ANGUILLA	AIA
ANTARCTICA	ATA
ANTIGUA AND BARBUDA	ATG
ARGENTINA	ARG
ARMENIA	ARM
ARUBA	ABW
AUSTRALIA	AUS
AUSTRIA	AUT
AZERBAIJAN	AZE
BAHAMAS	BHS
BAHRAIN	BHR
BANGLADESH	BGD
BARBADOS	BRB
BELARUS	BLR
BELGIUM	BEL
BELIZE	BLZ
BENIN	BEN
BERMUDA	BMU
BHUTAN	BTN
BOLIVIA	BOL
BOSNIA AND HERZEGOWINA	BIH
BOTSWANA	BWA
BOUVET ISLAND	BVT
BRAZIL	BRA
BRITISH INDIAN OCEAN TERRITORY	IOT
BRUNEI DARUSSALAM	BRN
BULGARIA	BGR
BURKINA FASO	BFA
BURUNDI	BDI
CAMBODIA	KHM
CAMEROON	CMR
CANADA	CAN
CAPE VERDE	CPV
CAYMAN ISLANDS	CYM
CENTRAL AFRICAN REPUBLIC	CAF
CHAD	TCD
CHILE	CHL
CHINA	CHN

CHRISTMAS ISLAND	CXR
COCOS (KEELING) ISLANDS	CCK
COLOMBIA	COL
COMOROS	COM
CONGO	COG
COOK ISLANDS	COK
COSTA RICA	CRI
COTE D'IVOIRE	CIV
CROATIA (local name: Hrvatska)	HRV
CUBA	CUB
CYPRUS	CYP
CZECH REPUBLIC	CZE
DENMARK	DNK
DJIBOUTI	DJI
DOMINICA	DMA
DOMINICAN REPUBLIC	DOM
EAST TIMOR	TMP
ECUADOR	ECU
EGYPT	EGY
EL SALVADOR	SLV
EQUATORIAL GUINEA	GNQ
ERITREA	ERI
ESTONIA	EST
ETHIOPIA	ETH
FALKLAND ISLANDS (MALVINAS)	FLK
FAROE ISLANDS	FRO
FIJI	FJI
FINLAND	FIN
FRANCE	FRA
FRANCE, METROPOLITAN	FXX
FRENCH GUIANA	GUF
FRENCH POLYNESIA	PYF
FRENCH SOUTHERN TERRITORIES	ATF
GABON	GAB
GAMBIA	GMB
GEORGIA	GEO
GERMANY	DEU
GHANA	GHA
GIBRALTAR	GIB
GREECE	GRC
GREENLAND	GRL
GRENADA	GRD
GUADELOUPE	GLP

GUAM	GUM
GUATEMALA	GTM
GUINEA	GIN
GUINEA-BISSAU	GNB
GUYANA	GUY
HAITI	HTI
HEARD AND MC DONALD ISLANDS	HMD
HONDURAS	HND
HONG KONG	HKG
HUNGARY	HUN
ICELAND	ISL
INDIA	IND
INDONESIA	IDN
IRAN (ISLAMIC REPUBLIC OF)	IRN
IRAQ	IRQ
IRELAND	IRL
ISRAEL	ISR
ITALY	ITA
JAMAICA	JAM
JAPAN	JPN
JORDAN	JOR
KAZAKHSTAN	KAZ
KENYA	KEN
KIRIBATI	KIR
KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	PRK
KOREA, REPUBLIC OF	KOR
KUWAIT	KWT
KYRGYZSTAN	KGZ
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LAO
LATVIA	LVA
LEBANON	LBN
LESOTHO	LSO
LIBERIA	LBR
LIBYAN ARAB JAMAHIRIYA	LBY
LIECHTENSTEIN	LIE
LITHUANIA	LTU
LUXEMBOURG	LUX
MACAU	MAC
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF	MKD
MADAGASCAR	MDG
MALAWI	MWI
MALAYSIA	MYS
MALDIVES	MDV
MALI	MLI
MALTA	MLT
MARSHALL ISLANDS	MHL
MARTINIQUE	MTQ
MAURITANIA	MRT
MAURITIUS	MUS

MAYOTTE	MYT
MEXICO	MEX
MICRONESIA, FEDERATED STATES OF	FSM
MOLDOVA, REPUBLIC OF	MDA
MONACO	MCO
MONGOLIA	MNG
MONTSERRAT	MSR
MOROCCO	MAR
MOZAMBIQUE	MOZ
MYANMAR	MMR
NAMIBIA	NAM
NAURU	NRU
NEPAL	NPL
NETHERLANDS	NLD
NETHERLANDS ANTILLES	ANT
NEW CALEDONIA	NCL
NEW ZEALAND	NZL
NICARAGUA	NIC
NIGER	NER
NIGERIA	NGA
NIUE	NIU
NORFOLK ISLAND	NFK
NORTHERN MARIANA ISLANDS	MNP
NORWAY	NOR
OMAN	OMN
PAKISTAN	PAK
PALAU	PLW
PANAMA	PAN
PAPUA NEW GUINEA	PNG
PARAGUAY	PRY
PERU	PER
PHILIPPINES	PHL
PITCAIRN	PCN
POLAND	POL
PORTUGAL	PRT
PUERTO RICO	PRI
QATAR	QAT
REUNION	REU
ROMANIA	ROM
RUSSIAN FEDERATION	RUS
RWANDA	RWA
SAINT KITTS AND NEVIS	KNA
SAINT LUCIA	LCA
SAINT VINCENT AND THE GRENADINES	VCT
SAMOA	WSM
SAN MARINO	SMR
SAO TOME AND PRINCIPE	STP
SAUDI ARABIA	SAU
SENEGAL	SEN
SEYCHELLES	SYC
SIERRA LEONE	SLE

SINGAPORE	SGP
SLOVAKIA (Slovak Republic)	SVK
SLOVENIA	SVN
SOLOMON ISLANDS	SLB
SOMALIA	SOM
SOUTH AFRICA	ZAF
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	SGS
SPAIN	ESP
SRI LANKA	LKA
ST. HELENA	SHN
ST. PIERRE AND MIQUELON	SPM
SUDAN	SDN
SURINAME	SUR
SVALBARD AND JAN MAYEN ISLANDS	SJM
SWAZILAND	SWZ
SWEDEN	SWE
SWITZERLAND	CHE
SYRIAN ARAB REPUBLIC	SYR
TAIWAN, PROVINCE OF CHINA	TWN
TAJIKISTAN	TJK
TANZANIA, UNITED REPUBLIC OF	TZA
THAILAND	THA
TOGO	TGO
TOKELAU	TKL
TONGA	TON
TRINIDAD AND TOBAGO	TTO
TUNISIA	TUN
TURKEY	TUR

TURKMENISTAN	TKM
TURKS AND CAICOS ISLANDS	TCA
TUVALU	TUV
UGANDA	UGA
UKRAINE	UKR
UNITED ARAB EMIRATES	ARE
UNITED KINGDOM	GBR
UNITED STATES	USA
UNITED STATES MINOR OUTLYING ISLANDS	UMI
URUGUAY	URY
UZBEKISTAN	UZB
VANUATU	VUT
VATICAN CITY STATE (HOLY SEE)	VAT
VENEZUELA	VEN
VIET NAM	VNM
VIRGIN ISLANDS (BRITISH)	VGB
VIRGIN ISLANDS (U.S.)	VIR
WALLIS AND FUTUNA ISLANDS	WLF
WESTERN SAHARA	ESH
YEMEN	YEM
YUGOSLAVIA	YUG
ZAIRE	ZAR
ZAMBIA	ZMB
ZIMBABWE	ZWE
UNDEFINED	XXX

APÉNDICE C. Tabla ISO 639-2

A continuación se expone la tabla de códigos de idiomas según el ISO 639-2 y que se corresponde con el tipo de datos *Idioma*.

AFAR	AA
ABKHAZIAN	AB
AFRIKAANS	AF
AMHARIC	AM
ARABIC	AR
ASSAMESE	AS
AYMARA	AY
AZERBAIJANI	AZ
BASHKIR	BA
BYELORUSSIAN	BE
BULGARIAN	BG
BIHARI	BH
BISLAMA	BI
BENGALI (Bangla)	BN
TIBETAN	BO
BRETON	BR
CATALAN	CA
CORSICAN	CO
CZECH	CS
WELSH	CY
DANISH	DA
GERMAN	DE
BHUTANI	DZ
GREEK	EL
ENGLISH (American)	EN
ESPERANTO	EO
SPANISH	ES
ESTONIAN	ET
BASQUE	EU
PERSIAN	FA
FINNISH	FI
FIJI	FJ
FAEROESE	FO
FRENCH	FR
FRISIAN	FY
IRISH	GA
GAELIC (Scots gaelic)	GD
GALICIAN	GL
GUARANI	GN
GUJARATI	GU
HAUSA	HA
HINDI	HI
CROATIAN	HR

HUNGARIAN	HU
ARMENIAN	HY
INTERLINGUA	IA
INTERLINGUE	IE
INUPIAK	IK
INDONESIAN	IN
ICELANDIC	IS
ITALIAN	IT
HEBREW	IW
JAPANESE	JA
YIDDISH	JI
JAVANESE	JW
GEORGIAN	KA
KAZAKH	KK
GREENLANDIC	KL
CAMBODIAN	KM
KANNADA	KN
KOREAN	KO
KASHMIRI	KS
KURDISH	KU
KIRGHIZ	KY
LATIN	LA
LINGALA	LN
LAOTHIAN	LO
LITHUANIAN	LT
LATVIAN (Lettish)	LV
MALAGASY	MG
MAORI	MI
MACEDONIAN	MK
MALAYALAM	ML
MONGOLIAN	MN
MOLDAVIAN	MO
MARATHI	MR
MALAY	MS
MALTESE	MT
BURMESE	MY
NAURU	NA
NEPALI	NE
DUTCH	NL
NORWEIGIAN	NO
OCCITAN	OC
OROMO (Afan)	OM
ORIYA	OR

PUNJABI	PA
POLISH	PL
PASHTO (Pushto)	PS
PORTUGUESE	PT
QUECHUA	QU
RHAETO-ROMANCE	RM
KIRUNDI	RN
ROMANIAN	RO
RUSSIAN	RU
KINYARWANDA	RW
SANSKRIT	SA
SINDHI	SD
SANGRO	SG
SERBO-CROATIAN	SH
SINGHALESE	SI
SLOVAK	SK
SLOVENIAN	SL
SAMOAN	SM
SHONA	SN
SOMALI	SO
ALBANIAN	SQ
SERBIAN	SR
SISWATI	SS
SESOTHO	ST
SUDANESE	SU

SWEDISH	SV
SWAHILI	SW
TAMIL	TA
TEGULU	TE
TAJIK	TG
THAI	TH
TIGRINYA	TI
TURKMEN	TK
TAGALOG	TL
SETSWANA	TN
TONGA	TO
TURKISH	TR
TSONGA	TS
TATAR	TT
TWI	TW
UKRAINIAN	UK
URDU	UR
UZBEK	UZ
VIETNAMESE	VI
VOLAPUK	VO
WOLOF	WO
XHOSA	XH
YORUBA	YO
CHINESE	ZH
ZULU	ZU

APÉNDICE D. Bibliografía

- Documentación de tecnologías JAVA:
  - **http://java.sun.com/**
- Documentación de MySQL;
  - **http://www.mysql.com**
- Documentación sobre el servidor Tomcat de aplicaciones web:
  - **http://jakarta.apache.org/tomcat/**
- Documentación de XML:
  - **http://www.w3.org/XML/**

APÉNDICE E. Palabras clave

Agenda

Contacto

Global

Perfil

Protocolo

Proveedor

Punto de acceso

Servicio

Servidor

XML