Mobile Computing and Applications

Luis Fernando Castillo (ed.)
Universidad de Caldas

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

800 AÑOS
VNiVERSiDAD D SALAMANCA
1218 - 2018

Ediciones Universidad Salamanca

AQUILAFUENTE, 241

© Motivo de cubierta: María Alonso

1.º edición: febrero, 2019
ISBN: 978-84-9012-861-9 (PDF)

Ediciones Universidad de Salamanca
Plaza de San Benito, s/n - E-37008
Salamanca (España)
Telf: +34 923 294 598 - http://www.eusal.es
eus@eusal.es

Realizado en España – Made in Spain

Mobile Computing and Applications

Desarrollo de aplicaciones para sistemas móviles con tecnología.NET

Manuel-Jesús Prieto Martín [1]

[1] Telefónica Investigación y Desarrollo, Spain
mjprieto@telefonica.es

Resumen: En este capítulo se presente una introducción de la tecnología .NET para el desarrollo de aplicaciones móviles. Las capacidades de los terminales de acceso a Internet que tenemos hoy en día, van desde las pantallas sencillas hasta las pantallas de alta calidad de ordenadores de sobremesa, pasando incluso por las televisiones. Para diseñar una web que sea capaz de servir información e interactuar con todos estos dispositivos de forma adecuada y satisfactoria, tendremos que utilizar herramientas adecuadas o de lo contrario el esfuerzo de desarrollo será enorme. Microsoft ha creado el Mobile Internet Toolkit para el desarrollo de sistemas móviles. Este marco de trabajo se engloba dentro de la tecnología ASP.NET, y está orientado al desarrollo de entornos de información para Internet con capacidades de acceso para dispositivos móviles. En este capítulo describimos la tecnología y se muestra como aplicarla.

Palabras clave: .Net; ASP; programación móvil

Abstract. This chapter provides an introduction to .NET technology for mobile application development. The capabilities of today's Internet access terminals range from simple screens to high quality desktop screens and even televisions. To design a website that is capable of serving information and interacting with all these devices in an adequate and satisfactory way, we will have to use appropriate tools or else the development effort will be enormous. Microsoft has created the Mobile Internet Toolkit for the development of mobile systems. This framework is part of ASP.NET technology and is aimed at developing information environments for the Internet with access capabilities for mobile devices. In this chapter we describe the technology and show how to apply it.

Keywords: .Net; ASP; mobile programming

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

1 INTRODUCCIÓN

Hoy en día nos encontramos con una cantidad de dispositivos diferentes con capacidades muy diversas, y todos ellos con acceso a Internet. Esto nos obliga a tener en cuenta todos estos dispositivos a la hora de crear nuestras aplicaciones o nuestros entornos web. Cuando estos entornos están diseñados como sistemas de provisión de información genéricos (portales de noticias o páginas corporativas) deberemos contemplar todos estos dispositivos para no perder usuarios potenciales o hacer nuestra web más completa. Cuando afrontamos el diseño de un sistema destinado a ser un servicio corporativo, será también útil contemplar todos estos dispositivos, para poder así potenciar nuestros sistemas y crear un entorno más potente, profesional y con mayores capacidades.

Las capacidades de los terminales de acceso a Internet que tenemos hoy en día van desde las mínimas pantallas en blanco y negro de algunos terminales móviles hasta las pantallas de alta calidad de ordenadores de sobremesa, pasando incluso por las televisiones. Para diseñar un web que sea capaz de servir información e interactuar con todos estos dispositivos de forma adecuada y satisfactoria, tendremos que utilizar herramientas adecuadas o de lo contrario el esfuerzo de desarrollo será enorme. Cuando hace unos años se comenzó a extender la tecnología WAP, las empresas que deseaban ofrecer información bajo esta tecnología tendían a tener una duplicidad de información y sistemas, una orientada al mundo web y otra orientada al mundo WAP. Este hecho es por si bastante preocupante, ya que la duplicidad de sistemas e información provoca que el esfuerzo de mantenimiento del sistema sea mayor. Cuando a parte de este hecho, queremos ofrecer un buen servicio a todos los terminales, tendremos que adaptar nuestra información a cada uno de ellos, lo que hace crecer definitivamente de forma exponencial el esfuerzo. Algunas familias de terminales, especialmente los terminales móviles, presentan un rango tan amplio de capacidades que la presentación de información de manera adecuada en todos ellos obliga a tomar en consideración el uso de un entorno de trabajo a que nos ayude a afrontar este reto, o de lo contrario será un trabajo ímprobo la consecución de este objetivo. El problema no radica sólo en que los terminales tengan un tamaño de pantalla muy diferente o capacidades cromáticas diversas, lo que implicaría una adaptación de los contenidos, sino que además presentan unos formatos de presentación de información muy diferentes (WML, cHTML o HTML) lo que implica un proceso de adaptación mucho más profundo.

Como medida de apoyo para el desarrollo de sistemas con las características antes descritas y basados en tecnología ASP, Microsoft ofrece lo que denominó *Mobile Internet Toolkit*. Este marco de trabajo se engloba dentro de la tecnología ASP.NET, y está orientado al desarrollo de entornos de información para Internet con capacidades de acceso para dispositivos móviles [1-3].

1.1 LA TECNOLOGÍA ASP.NET

ASP.NET es la evolución de la tecnología ASP (*Active Server Pages*). Se podría definir como un marco de trabajo para la creación de aplicaciones web. En la primera época de Internet, el contenido de la misma estaba alojado en páginas HTML (*Hypertext Markup Language*) estáticas, es decir, páginas cuyo contenido estaba determinado desde el momento de la creación de la misma y siempre se mostraba la misma información con la misma estructura. A partir de 1993 el contenido de ciertas páginas comenzó a ser dinámico, es decir, se generaba en el momento en el que se solicitaba la información. Dentro de este modelo, se desarrollaron gran número de tecnologías y soluciones, entre ellas, el marco de trabajo ASP, creado por la empresa Microsoft.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

ASP se basa en un lenguaje de marcado similar a HTML o WML, pero que contiene partes de código, delimitadas por <%....%>. Los ficheros ASP son salvados e instalados dentro de una aplicación adecuada (servidor WEB con capacidad ASP), de tal manera que cuando se solicita la página en cuestión, las partes de código especial serán interpretadas y la versión definitiva de la página será creada y provista [4,5].
La tecnología ASP.NET es una evolución de ASP, con el mismo objetivo final, pero con una mayor funcionalidad y extensibilidad que su antecesor. Como parte del marco de trabajo más amplio y potente, conocido como .NET, ASP.NET tiene, entre otras, la capacidad de crear aplicaciones WEB utilizando cualquiera de los lenguajes de programación que integran .NET.
A grandes rasgos, el funcionamiento de ASP.NET es el siguiente. El cliente solicitará al servidor una página con extensión *aspx*. El servidor localizará esta página dentro de su sistema y la compilará, generando una versión compilada de la página. Esta página, una vez compilada, será ejecutada dentro del marco de trabajo del servidor y el resultado generado por la misma será remitido por el servidor al cliente. Las siguientes veces que se solicite esta página al servidor, este ejecutará directamente la versión compilada de la misma.
El código fuente de la página puede ser escrito en cualquier de los lenguajes soportados por ASP.NET, es decir, puede ser Visual Basic, C# o C++. Cuando la página es compilada, se genera un código intermedio, denominado código manejado (managed code) que estará en un lenguaje común. Este código será compilado a su vez por el JIT (Just In Time Compiler) de tal forma que finalmente tenemos código nativo.

1.2 WEB FORMS o FORMULARIOS WEB

Dentro de la tecnología ASP.NET, la base para la construcción de las páginas son los denominados controles para formularios web o *web forms*. Estos controles son abstracciones o representaciones de los diferentes elementos visuales que nos podemos encontrar dentro de las páginas WEB. El concepto es simple, pero es funcionalmente muy potente ya que podemos programar dichos controles dentro del entorno .NET, de tal forma que los eventos de las aplicaciones WEB pueden ser capturados y tratados.
Los controles que componen los formularios WEB de ASP.NET(concepto diferente al de los formularios WEB clásicos) se configuran componiendo páginas que serán mostradas al usuario cuando este lo solicite. Esta filosofía nos permite trabajar separando perfectamente la lógica de negocio de nuestra aplicación de lo que sería la interfaz de usuario de la misma. Los controles encapsulan totalmente el código y por lo tanto, de forma diferente a lo que ocurre con la tecnología ASP antigua, el código de presentación no se ve invadido por código de negocio. Esto propicia una mejora en el proceso de desarrollo de las aplicaciones, a parte de una mayor reutilización de los componentes [6-10].
El siguiente código fuente muestra un ejemplo de cuál es el formato básico de una página ASP.NET. El código fuente completo de la página será el siguiente, en el que se muestra tanto el código HTML como el código correspondiente propiamente a ASP.NET.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

<%@ Page Language="C#" %>

```
<script runat="server">
   private void Page_Load(object sender, System.EventArgs e){
      Título.Text = "Master en Comercio Electrónico";
      correo.Text = "Manuel J. Prieto";
   }

</script>
<html>
<head>
</head>
<body>
   <form runat="server">
      <div align="center"><asp:Label id="Título" runat="server"
        forecolor="Blue" font-size="Larger" font-names="Arial Black"
        font-bold="True"></asp:Label>
      </div>
      <div align="center">
         <asp:Image id="Image1" runat="server"
           ImageUrl="esccolmt.gif ">
         </asp:Image>
      </div>
      <div align="center">
         <asp:HyperLink id="correo" runat="server"
           NavigateUrl="mailto:mjpm@tid.es">
         </asp:HyperLink>
      </div>
   </form>
</body>
</html>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Como se puede ver en el código anterior, dentro de una página ASP.Net, tendremos elementos diferenciados mediante etiquetas XML y que indica la existencia de los controles. También se puede comprobar cómo se incluye dentro de una página código ejecutable similar a la forma de programar existente en el ASP clásico.
El resultado de este código una vez ejecutado y servido al usuario que lo ha demandado será el siguiente:

```
<html>
<head>
</head>
<body>
   <form name="_ctl0" method="post" action="NewFile.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwxMzY1NTgxMDMyO3Q8O2w8aT-
wxPjs+O2w8dDw7bDxpPDE+O2k8NT47PjtsPHQ8cDxwPGw8VGV4dDs+O2w8TWFzdGVyIGVuIENvb
WVyY2lvIEVsZWN0csOzbmljbzs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8TWFudWVsIEouIFByaW
V0bzs+Pjs+Ozs+Oz4+Oz4+Oz5FIGHXkhhcruJ24mPX+bfLT7SsYQ==" />

      <div align="center"><span id="TÃtulo"><b><font face="Arial
        Black" color="Blue">Master en Comercio
        ElectrÃ³nico</font></b></span>
      </div>
      <div align="center">
         <img id="Image1" src="/esccolmt.gif " border="0" />
      </div>
      <div align="center">
      <a id="correo" href="mailto:mjpm@tid.es">Manuel J. Prieto</a>
      </div>
   </form>
</body>
</html>
```

Los controles, que en último término se corresponden con un elemento visual dentro de la página, se pueden programar de tal forma que controlaremos exactamente qué información se va a mostrar en dicho control y qué repercusiones tiene cada una de las acciones del usuario sobre nuestro sistema. En el ejemplo anterior, se muestra cómo se establece el valor de unas variables de forma programática.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Una gran ventaja de los controles, es que al fin y al cabo son clases del sistema, de tal forma que podemos crear nuestras propias versiones de estas clases extendiendo a las mismas y así tendremos un comportamiento especial de nuestra WEB, gracias nuestras propias versiones de los controles. Esto sería ir un paso más allá de la *simple* programación de los eventos de los controles estándar.
Los controles que provee ASP.NET son muy variados, incluyendo:
Controles HTML. Estos controles se corresponden con los elementos disponibles dentro del lenguaje HTML.
Controles de ASP.NET. Estos controles permiten la introducción de datos de forma genérica. Un ejemplo de este tipo de controles sería un *check box*. Estos controles están pensados para ser usados de forma genérica. Dentro de este tipo de controles se englobarán lo controles que componen el denominado *Mobile Internet Toolkit*.
Controles de validación. Estos son controles destinados a comprobar los datos introducidos por el usuario en el resto de controles. Estos controles simplificarán nuestro código y evitarán ciertos errores. Por ejemplo, con este tipo de controles se podrá validar la información introducida por el usuario en un campo, de tal forma que esta se corresponda con un determinado formato (como una fecha), tenga un determinado tipo de datos (como una cadena), que no esté en blanco la información de dicho campo o que se corresponda con una determinada expresión regular [11-15].

1.3 MOBILE WEB FORMS

Dentro del modelo visto anteriormente de los *Web Forms*, tenemos una extensión del mismo que nos permite desarrollar páginas orientadas a los clientes móviles, es decir, a los terminales móviles como un teléfono o un PDA. Estos terminales, como se vio en la introducción, presentan unas características dispares y soportan una serie de formatos de página diversos.
El modelo propuesto dentro del marco de trabajo ASP.NET para trabajar con terminales móviles, consiste en la abstracción de la interfaz de usuario mediante el uso de componentes que modelan los elementos visuales básicos. En entorno de ejecución nos provee de forma automática la transformación de este componente abstracto en un formato adecuado y entendible para el terminal concreto que realiza la petición.
Dentro de un mismo lenguaje de marcado, diferentes terminales contienen diferentes características dentro de la interfaz de usuario, como puede ser el número de botones disponibles o las teclas especiales del terminal. Estos detalles también los tiene en cuenta el motor de ASP.NET, de tal forma que se adapta lo mejor posible a cada terminal concreto, con el objetivo final de proveer una experiencia del usuario satisfactoria.
En el código siguiente, se muestra un ejemplo muy sencillo de lo que sería un *Mobile Web Form*.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>

<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<mobile:Form runat="server">

   Hola, mundo móvil!<br />

   <mobile:Label runat="server" Text="Etiqueta" />

</mobile:Form>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Como vemos, en la página configurada en el ejemplo anterior se mostrará un texto literal y un control correspondiente al MIT. Las primeras líneas corresponden a las etiquetas obligatorias para la indicar que se trata de página orientada al mundo móvil. También cabe destacar dentro de la cabecera del formulario, la etiqueta *Language*, que nos permitirá especificar el lenguaje de programación usado para escribir el código correspondiente a la página. El resto del código de la página es muy sencillo. Como se puede comprobar, se define un formulario a través de la etiqueta *mobile:Form* y se introduce la información dentro del mismo. Cualquier página que trabaje con controles del MIT debe tener al menos un formulario. Por último, vemos que dentro del propio formulario tenemos un texto literal estático, alguna etiqueta HTML y un control que nos permite especificar un texto y que no es más que un control que simula una etiqueta de texto. La siguiente imagen muestra el resultado del formulario anterior una vez servida la página al dispositivo.

En el párrafo anterior hemos indicado que las páginas móviles basadas en MIT deben tener al menos un formulario. Dentro de los lenguajes de marcado orientados a móviles, es común tener lo que podríamos definir como varias páginas lógicas dentro de una única página física (modelo del *mazo de cartas* de WML). Por esto, a diferencia de las páginas ASP.NET estándar, en las que cada página tiene un único formulario, en el caso de los *Mobile Forms*, es común tener varios formularios en una única página.

Dentro de una página con varios formularios móviles, disponemos de la propiedad *ActiveForm* para indicar cuál de los formularios incluidos en la página debe ser el formulario activo en cada momento. Por supuesto, también se pueden navegar entre los formularios a través de enlaces. El siguiente ejemplo nos muestra una página con varios formularios y nos presenta un sistema de navegación entre dichos formularios basado en enlaces [16-18].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<mobile:Form id="FirstForm" runat="server">
   <mobile:Label id="Label1" runat="server" StyleReference="title">
      Form #1 de la página
   </mobile:Label>
   <mobile:Link id="Link1" runat="server" NavigateURL="#SecondForm">
     Siguiente Form
   </mobile:Link>
</mobile:Form>
<mobile:Form id="SecondForm" runat="server">
   <mobile:Label id="Label2" runat="server" StyleReference="title">
     Form #2 de la página
   </mobile:Label>
   <mobile:Link id="Link2" runat="server" NavigateURL="#FirstForm">
     Primer Form
   </mobile:Link>
</mobile:Form>
```

El resultado del ejemplo anterior es el siguiente.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

1.4 PROGRAMACIÓN DENTRO DE ASP.NET

Como se ha comentado anteriormente, la programación utilizada dentro de los proyectos desarrollados con ASP.NET puede realizar con cualquier de los lenguajes soportados dentro de .NET. Dentro de este documento se utilizará C# como lenguaje de programación.

Una vez que tenemos seleccionado el lenguaje de programación, deberemos decidir la técnica de programación a utilizar. Tendremos dos posibles métodos sobre los que elegir: código incrustado (*inline code*) o tener ficheros separados con el código. En este documento se utilizarán las dos técnicas indistintamente.

Dentro del modelo de programación de ASP.NET, podemos atrapar los diferentes eventos que se producen. Supongamos que dentro de una página ASP.NET tenemos un control correspondiente a un botón. Podemos indicar el código que se ejecutará al seleccionar dicho botón. Dicho código se ejecutará por supuesto en el servidor. El siguiente código muestra un ejemplo del método a escribir para capturar el evento de pulsación de un botón [19-24].

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Boton.aspx.cs" Inherits="MasterSalamanca.Boton" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="num" runat="server">

                  No se ha pulsado el bot&#243;n

                </mobile:Label>

                <mobile:Command id="comando" runat="server"

                     StyleReference="subcommand">

                  Ok

                </mobile:Command>

        </mobile:Form>

</body>
```

El código que se ejecuta en el servidor al pulsar un botón, es el que se presenta a continuación.

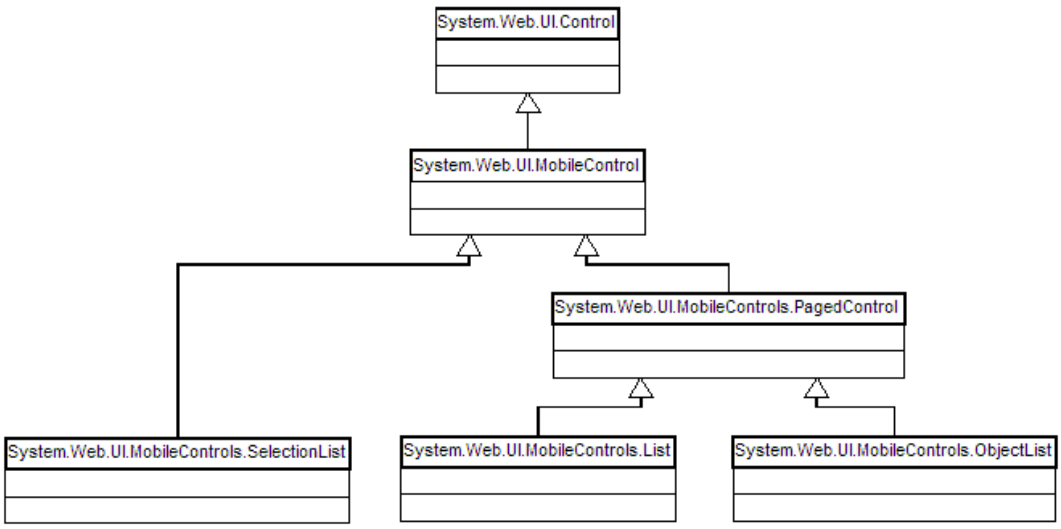
```
private void comando_Click(object sender, System.EventArgs e)

                {

                        num.Text = "Se ha pulsado el botón";

                }
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El resultado final en el terminal es el que se puede ver en la siguiente imagen.

2 LISTAS DE ELEMENTOS

Dentro de las aplicaciones móviles, es habitual el uso de los listados de elementos como principal forma de presentación de información. Esto es así debido a las importantes restricciones de la pantalla de los terminales y sobre todo a la pobre interfaz que presentan los dispositivos para la introducción de información por parte del usuario. Dentro del MIT disponemos de tres tipos diferentes de listas, modeladas cada una de ellas por una clase. Las clases de las que hablamos serán *SelectionList*, *ObjectList* y *List*. El siguiente diagrama de clases presenta la composición de dichas clases.



ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El control más simple es el correspondiente a *SelectionList*, ya que este, tal y como podemos descubrir en el diagrama de clases, no dispone de la capacidad de paginación. El control *SelectionList* es el único de los controles de listas que permite selección múltiple. Este control puede ser transformado en última instancia en una lista desplegable, en una serie de *radio buttons* o en otros elementos similares. Cualquier *SelectionList* debe ir acompañado de un botón de comando que nos permite enviar la selección del usuario al servidor.
El control *List* nos permitirá crear una lista estática o bien una lista interactiva. Cuando se utiliza este control de forma estática, el funcionamiento es similar a *SelectionControl*, es decir, se indican una serie de elementos y se acompaña el control *List* por un control correspondiente a un botón de comando, con el objetivo de enviar la selección de la lista al servidor. En cambio, si el control *List* funciona en modo interactivo, podremos utilizar cada uno de los ítems de la lista como un enlace, que generará un evento al ser seleccionado. Es decir, en este caso, los propios elementos de la lista son el punto de comunicación con el servidor y por lo tanto no es necesario un botón de comando auxiliar. Los ítems que componen la lista pueden ser especificados tanto de forma estática, como de forma dinámica a través de los que se denomina *data binding*.
Por último, el control *ObjectList* presenta la particularidad de no permitir la especificación estática de los elementos de la lista, debiendo realizar esta operación en todos los casos a través del enlace entre la lista y una fuente de datos. Esta fuente de datos será típicamente un conjunto de objetos (por ejemplo, un *array*) y se indicará qué campo del objeto se utilizará como clave, de tal forma que se mostrará dicha información en la lista. Una vez seleccionado un determinado elemento dentro de la lista, se mostrará el resto de información contenida en el objeto [25-30].

2.1 EL CONTROL SELECTIONLIST

Como hemos comentado, este control es el más sencillo dentro de los controles orientados a la generación de listas de elementos. En el siguiente ejemplo vemos un uso muy sencillo de este control en el que se presenta una lista de cuatro elementos, indicados estáticamente y se acompaña dicha lista de un botón destinado al envío de los datos al servidor.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="EjSelecionList1.aspx.cs" Inherits="MasterSalamanca.MobileWebForm1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:SelectionList id="lista" runat="server" SelectType="ListBox">

                        <Item Value="uno" Text="uno" Selected="True"></Item>

                        <Item Value="dos" Text="dos"></Item>

                        <Item Value="tres" Text="tres"></Item>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                    <Item Value="cuatro" Text="cuatro"></Item>
                </mobile:SelectionList>
                <mobile:Command id="enviar" runat="server">Enviar</mobile:Command>
            </mobile:Form>
</body>
```

En el fragmento de código anterior se añaden cuatro elementos a la lista, esto se realiza a través de la etiqueta *item*. Otro elemento a destacar, es la propiedad *SelectType* que nos permitirá especificar la apariencia de la lista una vez generado el código en el adecuado lenguaje de marcado. Los posibles valores que puede tomar dicha propiedad son los siguientes:

- *DropDown* – Se corresponde con una lista desplegable
- *ListBox* – Se corresponde con una lista normal de elementos
- *Radio* – Cada elemento de la lista será un *radio button* seleccionable
- *MultiSelectListBox* – En este caso se permiten selecciones múltiples dentro de una interfaz similar al caso de *ListBox*
- *CheckBox* – Cada elemento de la lista se podrá seleccionar mediante lo que se denomina *checkbox*, y en este caso, la selección también puede ser múltiple

Las siguientes imágenes muestran el resultado final en el terminal.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

A continuación, vamos a ampliar el código de ejemplo anterior para explicar cómo se capturaría la selección de un elemento de la lista, una vez enviada esta información. En este caso, al formulario anterior le vamos a añadir un segundo formulario, que será el que muestre el resultado. El código completo de la página con los dos formularios es el siguiente.

```
<%@ Page language="c#" Codebehind="EjSelecionList1.aspx.cs" Inherits="MasterSalamanca.MobileWebForm1" AutoEventWireup="false" %>

<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:SelectionList id="lista" runat="server" SelectType="ListBox">

                        <Item Value="uno" Text="uno" Selected="True"></Item>

                        <Item Value="dos" Text="dos"></Item>

                        <Item Value="tres" Text="tres"></Item>

                        <Item Value="cuatro" Text="cuatro"></Item>

                </mobile:SelectionList>

                <mobile:Command id="enviar" runat="server">Enviar</mobile:Command>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        </mobile:Form>

        <mobile:Form id="Form2" runat="server">
                <mobile:Label id="result" runat="server"></mobile:Label>
        </mobile:Form>
</body>
```

El siguiente fragmento de código muestra el código C# que se ejecutará al pinchar sobre el botón enviar del primer formulario. Como se puede comprobar en el código de la página (gracias a la propiedad *Codebehind*), el código asociado a la misma estará alojado en el fichero *EjSelecionList1.aspx.cs*.

```
using System;

namespace MasterSalamanca
{
        public class MobileWebForm1 : System.Web.UI.MobileControls.MobilePage
        {
                protected System.Web.UI.MobileControls.SelectionList lista;
                protected System.Web.UI.MobileControls.Command enviar;
                protected System.Web.UI.MobileControls.Form Form2;
                protected System.Web.UI.MobileControls.Label result;
                protected System.Web.UI.MobileControls.Form Form1;

                private void enviar_Click(object sender, System.EventArgs e)
                {
                        this.ActiveForm = Form2;
                        string seleccion = lista.Selection.Value;
                        result.Text = "El elemento seleccionado es: " + seleccion;
                }
        }
}
```

El resultado de la ejecución de este ejemplo en un terminal, sería muestra en las siguientes pantallas.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

En el caso de que tuviéramos una selección múltiple, el código C# que nos permitiría recuperar la información de todos los elementos seleccionados, sería el que se presenta a continuación. En este caso, sólo se muestra el método concreto, no se muestra toda la clase, ya que sería similar a lo visto en el último ejemplo. El único cambio que habría que realizar sería la inclusión de la línea que nos permite utilizar las clases *MobileListItemCollection* y *MobileListItem*, tal y como se muestra también en el listado siguiente.

```
Using System.Web.UI.MobileControls;
….
private void enviar_Click(object sender, System.EventArgs e)
{
 this.ActiveForm = Form2;
 MobileListItemCollection items = lista.Items;
 string seleccion = “”;
 foreach (MobileListItem item in items)
 {
   if (item.Selected)
   {
     seleccion += item.Value + “<br/>”;
   }
 }
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
  result.Text = "Los elementos seleccionados son: " + seleccion;
}
```

2.1.1 CONSTRUCCIÓN DE LISTAS CON DATA BINDING

Como se ha comentado anteriormente en varias ocasiones, es posible asignar o indicar los elementos de una lista de forma dinámica mediante el uso de lo que se denomina *data binding*. A pesar de que este ejemplo se basa en un control de tipo *SelectionList*, lo que veremos sobre *data binding* es totalmente aplicable al resto de controles de tipo lista. La fuente de datos que establecerá los elementos de la lista debe ser un objeto de tipo *IEnumerate* o *IListSource*. Dentro de las librerías de clases incluidas en el marco de trabajo de .NET, tenemos varias que implementan la interfaz *IEnumerate*, como por ejemplo *Array*, *ArrayList* o *MobileListItemCollection*. En el caso de la interfaz *IListSource*, sólo dos clases de .NET la implementan: *DataSet* y *DateTable*. *DataSet* no es más que una colección de objetos de tipo *DataTable*, y estos elementos están relacionados con la tecnología ADO.NET de acceso a bases de datos [31-34].

El siguiente ejemplo muestra cómo crear a una lista de tipo *SelectionList*, que es completada a partir de la información generada de forma programática. En primer lugar, vamos a ver el código correspondiente al propio control *SelectionList*. En este caso no se muestra el resto de código de la página, porque no es relevante y ya está explicado en anteriores ejemplos.

```
<mobile:SelectionList id=lista runat="server"
  SelectType="MultiSelectListBox"
  DataTextField="Campo" DataValueField="Valor">
</mobile:SelectionList>
```

En este fragmento de código, caben destacar las propiedades *DataTextField* y *DataValueField*, que nos permiten indicar a través de qué métodos seremos capaces de obtener la información correspondiente al texto y al valor, respectivamente, de cada uno de los ítems de la lista.

El código de las clases C# que servirán de soporte para la lista y para la página que contiene dicha lista, será el siguiente.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.Mobile;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.MobileControls;
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace MasterSalamanca
{
	class Elemento
	{
		private String campo, valor;

		public Elemento (String campo, String valor)
		{
			this.campo = campo;
			this.valor = valor;
		}

		public String Campo{get{return this.campo;}}
		public String Valor{get{return this.valor;}}
	}


	public class EjSelectionList2 : System.Web.UI.MobileControls.MobilePage
	{
		protected System.Web.UI.MobileControls.SelectionList lista;
		protected System.Web.UI.MobileControls.Form Form1;

		private void Page_Load(object sender, System.EventArgs e)
		{
			if (!IsPostBack)
			{
				ArrayList datos = new ArrayList();
				datos.Add(new Elemento("A", "1"));
				datos.Add(new Elemento("AB", "3"));
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                    datos.Add(new Elemento("ABC", "4"));
                    lista.DataSource = datos;
                    lista.DataBind();
                }
            }
        }
}
```

Acabamos de presentar dos clases, que servirán para crear la lista de forma dinámica. La primera clase se ha denominado *Elemento* y no es más que lo que podríamos denominar un contenedor de datos. Esta clase presenta dos atributos y dos métodos destinados a permitir recuperar la información almacenada en dichos atributos. En este ejemplo tan sencillo, los métodos simplemente retornan la información de los atributos, pero es obvio que se podrían realizar operaciones más complejas para así determinar qué información será mostrada en la lista.

En segundo lugar, tendremos la clase *EjSelectionList2*, que será la clase que soporte la lógica de la página ASP.NET. En el ejemplo que nos ocupa, lo único que se ha incluido en esta clase es una serie de líneas de código dentro del método *Page_Load*. Este método, tal y como indica su nombre, se ejecuta al cargar la página, es decir, cada vez que la página en cuestión sea solicitada, se ejecutará este método como lo que podríamos definir como un proceso de iniciación de dicha página. Lo que hacemos en este método en primer lugar, es comprobar si la página está siendo accedida por primera vez o se trata de una recarga debida a que el usuario ha retrocedido en la secuencia de navegación. Esto se hace gracias a *IsPostBack*. Si la página no es recargada, se genera la información y se incluye la misma dentro de la lista. Como vemos, lo que hacemos es crear un objeto de tipo *ArrayList*, que tal y como se ha indicado implementa la interfaz *IEnumerate* y se introducen objetos de tipo *Elemento* en esta matriz. Una vez hecho esto, se asigna el objeto de tipo *ArrayList* a la lista y por lo tanto esta quedará asociada desde el punto de vista de la fuente de datos, a la matriz de datos indicada [35-40].

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

2.2 EL CONTROL LIST

El control *List*, ya ha sido presentado en la introducción de los controles de tipo lista y tal y como se dijo entonces, es muy similar al control *SelectionList*, si bien, presenta la capacidad de paginación ausente en este último. El control *List*, puede ser utilizado para generar una lista estática o una lista interactiva. En este último caso, no hay posibilidad de hacer selección múltiple, ya que cada elemento de la lista sirve como enlace para la ejecución de una acción. En este caso, es obvio que no se necesita el botón de comando adicional. Por defecto, el control lista generará una lista en modo interactivo. Para modificar este comportamiento habrá que establecer la propiedad *ItemsAsLinks* con valor *false*. El siguiente ejemplo muestra un ejemplo sencillo sobre el comportamiento de un control de tipo *List*, funcionando en modo interactivo. En el siguiente fragmento de código muestra únicamente la definición de la lista.

```
<mobile:Form id="Form1" runat="server">
        <mobile:List id="lista" runat="server">
                <Item Value="Elto1" Text="Elto1"></Item>
                <Item Value="Elto2" Text="Elto2"></Item>
                <Item Value="Elto3" Text="Elto3"></Item>
                <Item Value="Elto4" Text="Elto4"></Item>
        </mobile:List>
</mobile:Form>
```

Como vemos definimos una lista de forma estática (sin inclusión programática de datos) y con comportamiento dinámico, es decir, cada ítem de la lista actuará como un enlace. El siguiente fragmento

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

de código muestra cómo sería el método que responde a la selección por parte del usuario de uno de los ítems de la lista.

```
private void List1_ItemCommand(object sender, System.Web.UI.MobileControls.ListCommandEventArgs e)
{
        this.ActiveForm = Form2;
        String seleccion = e.ListItem.Value;
        result.Text = "Ha seleccionado: " + seleccion;
}
```

Las siguientes imágenes muestran el resultado final.

2.2.1 PAGINACIÓN

Como se ha comentado en varias ocasiones anteriormente, el control *List* presenta la capacidad de paginación de la lista de forma automática. Para que la paginación sea posible, debemos indicar el número de ítems a mostrar por página y el atributo *Paginate* del formulario debe ser establecido con el valor *true*.

Para realizar una paginación más eficiente, se puede realizar el proceso de tal forma que sólo se suministre al control aquella información que va a mostrar, es decir, la información referente a la página que está generando en cada momento.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

2.3 EL CONTROL OBJECTLIST

Este control es quizás el más potente de todos los controles destinados a listas ya que permite un mayor número de tipos de datos como fuente y presenta un manejo más potente de los comandos o acciones a asociar con cada ítem de la lista. Como ya se mencionó anteriormente, el control *ObjectList* presenta las mismas capacidades de paginación que el control *List*. Las listas basadas en *ObjectList* no admiten, en cambio, la provisión estática de los elementos que componen dicha lista.

Durante el estudio del control *List*, hemos visto cómo se puede asociar un control a una fuente de datos. En este caso, con el control *ObjectList*, vamos a comprobar la potencia del mismo manejando fuentes de una forma más compleja. Veremos cómo, partiendo de una fuente de datos con elementos que se componen de varios campos, el control mostrará un listado de un determinado campo de los diferentes elementos de la fuente y al seleccionar uno de ellos, se mostrará una pantalla con la información completa del elemento en cuestión de la fuente de datos [41-43].

El siguiente ejemplo muestra un ejemplo sencillo de cómo funciona este control. En primer lugar, lo que hacemos es definir una clase contenedora de datos. Esta clase tendrá campos de varios tipos de datos. El código de esta clase es el siguiente.

```
class ElementoComplejo
        {
                private String id;
                private String campo1, campo2;
                private int campo3;
                private double campo4;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        public ElementoComplejo(String id, String campo1, String campo2, int campo3, double
campo4)
        {
                this.id = id;
                this.campo1 = campo1;
                this.campo2 = campo2;
                this.campo3 = campo3;
                this.campo4 = campo4;
        }

        public String Id{get{return this.id;}}
        public String Campo1{get{return this.campo1;}}
        public String Campo2{get{return this.campo2;}}
        public int Campo3{get{return this.campo3;}}
        public double Campo4{get{return this.campo4;}}
}
```

El formulario de la página, como es de esperar, tendrá un control de tipo *ObjectList*, que se llamará *lista*. Lo único que se presenta en este documento, es el método *Page_Load* asociado al formulario, ya que el resto del código del mismo es irrelevante.

```
private void Page_Load(object sender, System.EventArgs e)
        {
                if (!IsPostBack)
                {
                        ArrayList datos = new ArrayList();
                        for (int i=0; i<10; i++)
                        {
                                ElementoComplejo ec = new ElementoComplejo(
                                        "id" + i, "c1 - " + i, "c2 - " + i,
                                        i, i);
                                datos.Add(ec);
                        }

                        lista.DataSource = datos;
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                    lista.LabelField = "Id";
                    lista.DataBind();
                }


            }
```

Se puede visualizar más de un ítem en el listado de todos los elementos que componen la lista. Esta visualización se hará de forma tabular. Para realizar esta operación, lo único que tendríamos que hacer es añadir la siguiente línea al método *Page_Load*, que como se puede imaginar al verla, establece los campos *Id* y *Campo1* del elemento de datos de tipo *ElementoComplejo*, como elementos a visualizar en el listado.

```
lista.TableFields = "Id;Campo1";
```

Como se ha comentado ya, el control *ObjectList* permite una gestión más compleja de los comandos asociados a los diferentes ítems de la lista. En concreto, nos permite asociar más de un comando a un único ítem. Es decir, en todos los ejemplos que hemos visto hasta ahora, al seleccionar un ítem, se disparaba una determinada acción. En este caso, podremos también seleccionar qué acción se realizará, partiendo del hecho de que existe un elemento seleccionado, podremos realizar varias operaciones sobre este registro o elemento de la lista seleccionado.
El siguiente código muestra cómo sería el código correspondiente a la página. Es similar a ejemplos vistos anteriormente, pero podemos comprobar que tenemos la definición de dos comandos.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="ObjectListEj1.aspx.cs" Inherits="MasterSalamanca.ObjectListEj1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:ObjectList id="lista" runat="server" LabelStyle-StyleReference="title" CommandStyle-StyleReference="subcommand">

                        <Command Name="Campo1" Text="Muestra campo 1"></Command>

                        <Command Name="Campo2" Text="Muestra campo 2"></Command>

                </mobile:ObjectList>

        </mobile:Form>

        <mobile:Form id="Form2" runat="server">

                <mobile:Label id="Label1" runat="server">Label</mobile:Label>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                <mobile:Label id="Label2" runat="server">Label</mobile:Label>
        </mobile:Form>
</body>
```

El código C# asociado a esta página, es muy similar, por lo que único que se mostrará en el siguiente listado, será el método encargado de gestionar la selección de los diferentes comandos.

```
protected void Elto_OnItemCommand(Object sender, ObjectListCommandEventArgs  args)
{
        this.ActiveForm = Form2;
        if (args.CommandName == "Campo1")
        {
                Label1.Text = "Campo 1";
                Label2.Text = args.ListItem["Campo1"];
        }
        else
        {
                Label1.Text = "Campo 2";
                Label2.Text = args.ListItem["Campo2"];
        }
}
```

Las siguientes imágenes muestran cómo sería el resultado final de estos ejemplos.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3 EL CONTROL CALENDAR

El control *Calendar*, como su nombre indica, nos permite realizar de forma sencilla y flexible, la gestión de introducción y selección de fechas por parte del usuario. En aquellos casos en el que el terminal de acceso lo permita, el resultado será gráfico. El uso de este control para que el usuario indique fechas presenta dos ventajas importantes. La primera de ella es la uniformidad en las fechas,

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

lo que evita la necesidad de formateo de las fechas, o el control de posibles errores. Por otro lado, facilita el proceso de cara al usuario, ya que este no tiene que teclear la fecha de forma explícita.
A la hora de introducir este control dentro de un formulario, el programador podrá especificar gran cantidad de propiedades del control, que modificarán el comportamiento del mismo. El siguiente código muestra un ejemplo muy sencillo. En primer lugar, tenemos el código correspondiente al formulario que contiene el control *Calendar*.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Calendario.aspx.cs" Inherits="MasterSalamanca.Calendario" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Calendar id="cal" runat="server" SelectionMode="DayWeek" SelectedDate="2004-01-01"></mobile:Calendar>

                <mobile:Label id="Label1" runat="server">Label</mobile:Label>

        </mobile:Form>

</body>
```

El código que hay detrás de la página anterior es muy sencillo, y lo que hará será mostrar por pantalla, dentro de una etiqueta, la fecha seleccionada. El control *Calendar*, como es de esperar, nos permite recuperar la fecha en una gran variedad de formatos.

```
private void cal_SelectionChanged(object sender, System.EventArgs e)

{

        Label1.Text = cal.SelectedDate.ToShortDateString();

}
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

4 EL CONTROL PHONECALL

Estamos contemplado el desarrollo de servicios de provisión de información para terminales móviles. Típicamente, estos terminales móviles serán teléfonos móviles, lo que nos lleva a tener en nuestras manos un elemento natural de comunicación, de comunicación vocal. Muchos de nuestros servicios podrán ser mucho más completos y efectivos, si enlazan el proceso con una llamada estándar de voz.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El control *PhoneCall* permitirá la inicialización automática de llamadas si el terminal lo soporta. En otro caso, el control mostrará un enlace que puede ser seleccionado por el usuario con el fin de realizar la llamada.
El siguiente ejemplo muestra el uso sencillo de este control.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="PhoneCallEj1.aspx.cs" Inherits="MasterSalamanca.PhoneCallEj1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">


<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

   <mobile:Form id=Form1 runat="server">

<mobile:PhoneCall id=telef runat="server" PhoneNumber="923 12 34 56" AlternateUrl="http://gsii.usal.es">Llamar a</mobile:PhoneCall></mobile:Form>

</body>
```

5 EL CONTROL ADROTATOR

Dentro del marketing o publicidad *online*, es muy común la presentación dentro de las páginas de lo que se denominan *banners* y que se podría traducir por pancarta o banderola. Para hacer más efectivo este sistema publicitario, habitualmente es requerida una rotación en el mensaje mostrado, de tal manera que la cantidad de usuarios que contemplan los anuncios es mayor y también es mayor el número de mensajes publicitarios que se le muestran a un mismo usuario. Este comportamiento es tan común dentro de los entornos web, que *Microsoft* provee un control dentro del MIT que nos permite realizar estas operaciones de forma automática. Este control es el denominado *AdRotator*.
El control nos permite introducir dentro de nuestras páginas, mensajes publicitarios gráficos. La información sobre los mensajes publicitarios se debe proveer a través de un fichero XML, con el siguiente formato [44].

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

<Advertisements>

  <Ad>

    <ImageUrl>[Imagen a mostrar]</ImageUrl>

    <MonoImageUrl>[Imagen monocromática a mostrar. Será típicamente un fichero WBMP para navegadores WML]</MonoImageUrl>

    <NavigateUrl>[URL a mostrar al seleccionar el enlace]</NavigateUrl>

    <AlternateText>[Texto a mostrar si no se puede ]</AlternateText>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
  <Keyword>[Categoría del banner]</Keyword>
  <Impressions>[Indica cuantas veces debe ser mostrado el anuncio con respecto al resto]</Impressions>
 </Ad>
 <Ad>
 ……
</Advertisements>
```

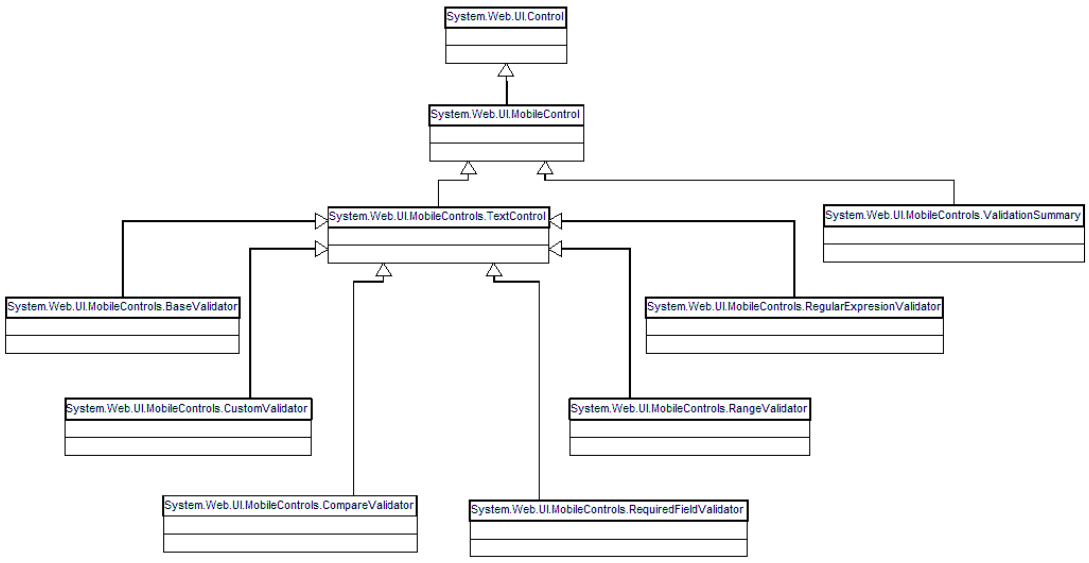
6 CONTROLES DE VALIDACIÓN

Dentro de cualquier aplicación, las validaciones de la información proporcionada por el usuario son muy importante, pero en el caso de las aplicaciones web es más importante si cabe, ya que la información, a pesar de ser errónea, es posible que sea enviada al servidor y esto supone un consumo de red. Este hecho, dentro de los entornos móviles, también cobra especial importancia, ya que las comunicaciones son más lentas y caras.

Algunas de las validaciones típicas a realizar serán el chequeo de que ciertos campos han sido completados, que tienen un formato correcto (correo electrónico o teléfono) o que dos campos contienen la misma información (común cuando se tiene que confirmar una clave de acceso, por ejemplo).

La realización de todas estas operaciones de validación, suele ser un trabajo costoso, tanto si hace en la parte del servidor, como si se hace en la parte del cliente (no siempre se puede comprobar todo en esta parte cliente). Para facilitar esta labor, ASP.NET proporciona un método flexible basado en lo que se denominan, controles de validación. Dentro del MIT, también nos encontramos con algunos controles de este tipo, que debido a las peculiaridades de los clientes a los que está orientado el MIT, no presentará procesos de validación en el cliente y todos ellos se realizarán en la parte del servidor.

La siguiente imagen presenta el diagrama de clases de los controles que proporciona el MIT para la realización de validaciones [45].



ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

6.1 EL CONTROL REQUIREDFIELDCONTROL

El control *RequiredFieldControl* es el más simple de los controles de validación y será el más común de todos. Este control simplemente comprueba que el usuario haya introducido alguna información dentro de un determinado campo.
El siguiente ejemplo muestra un sencillo ejemplo, en el que se le presenta al usuario lo que podría ser una pantalla de *login* o identificación y este debe especificar su nombre y clave de acceso. En caso de que alguno de los dos campos esté sin completar, se mostrará el error correspondiente. Por lo tanto, tendremos que introducir un validador para cada uno de los campos, el nombre y la clave de acceso. En caso de que ambos casos estén completos, iremos al segundo formulario de la página, que contiene simplemente un texto de bienvenida. El código de la página se muestra a continuación.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Valid1.aspx.cs" Inherits="MasterSalamanca.Valid1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="Label1" runat="server" BreakAfter="False">Nombre: </mobile:Label>

                <mobile:TextBox id="nombre" title="Nombre:" runat="server" EnableViewState="False"></mobile:TextBox>

                <mobile:Label id="Label2" runat="server" BreakAfter="False">Clave: </mobile:Label>

                <mobile:TextBox id="clave" runat="server" Password="True"></mobile:TextBox>

                <mobile:RequiredFieldValidator id="validaClave" runat="server" ErrorMessage="Indique su clave de acceso" ControlToValidate="clave"></mobile:RequiredFieldValidator>

                <mobile:RequiredFieldValidator id="validaNombre" runat="server" ErrorMessage="Indique su nombre" ControlToValidate="nombre"></mobile:RequiredFieldValidator>

                <mobile:Command id="ok" runat="server">ok</mobile:Command>

        </mobile:Form>

        <mobile:Form id="Form2" runat="server">

                <mobile:Label id="msg" runat="server">Bienvenido...</mobile:Label>

        </mobile:Form>

</body>
```

El código correspondiente al botón *ok* de la página, es muy sencillo y lo único que hará será comprobar que la página es válida, y en caso de que sea así, activar el segundo formulario.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
if (Page.IsValid)
{
        ActiveForm = Form2;
}
```

La siguiente imagen muestra la pantalla de error, resultado de la validación.

6.2 EL CONTROL COMPAREVALIDATOR

El control *CompareValidator*, tal y como mencionamos anteriormente, nos permite comparar dos campos relacionados. La posible comparación de estos campos, no se limita a comparaciones de igualdad, sino que también podremos, por ejemplo, comprobar si un campo tiene un valor mayor que el otro [46].

Vamos a ampliar el ejemplo anterior, de tal forma que el usuario tendrá que identificarse indicando la clave de acceso dos veces para la comprobación de la misma. Es decir, tendremos un campo más, y comprobaremos que todos los campos *nombre* y *clave* están completos, y que los campos correspondientes a la clave y la confirmación de esta, contienen la misma información.

La página tendría ahora el siguiente código.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Valid2.aspx.cs" Inherits="MasterSalamanca.Valid2" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        <mobile:Form id="Form1" runat="server">
                <mobile:Label id="Label1" runat="server" BreakAfter="False">Nombre: </mobile:Label>
                <mobile:TextBox id="nombre" title="Nombre:" runat="server" Ena-
bleViewState="False"></mobile:TextBox>
                <mobile:Label id="Label2" runat="server" BreakAfter="False">Clave: </mobile:Label>
                <mobile:TextBox id="clave" runat="server" Password="True"></mobile:TextBox>
                <mobile:Label id="Label3" runat="server" BreakAfter="False">Confirmaci&#243;n
Clave:</mobile:Label>
                <mobile:TextBox id="clave2" runat="server" Password="True"></mobile:TextBox>
                <mobile:RequiredFieldValidator id="validaNombre" runat="server" ControlToVali-
date="nombre" ErrorMessage="Indique su nombre"></mobile:RequiredFieldValidator>
                <mobile:RequiredFieldValidator id="validaClave" runat="server" ControlToVali-
date="clave" ErrorMessage="Indique su clave de acceso"></mobile:RequiredFieldValidator>
                <mobile:CompareValidator id="validClaves" runat="server" ControlToValidate="clave2"
ErrorMessage="Las claves indicadas no coinciden" ControlToCompare="clave"></mobile:CompareValida-
tor>
                <mobile:Command id="ok" runat="server">ok</mobile:Command>
        </mobile:Form>
        <mobile:Form id="Form2" runat="server">
                <mobile:Label id="msg" runat="server">Bienvenido...</mobile:Label>
        </mobile:Form>
</body>
```

El código correspondiente al control del botón que permite activar el segundo formulario sería exactamente el mismo que el visto en el ejemplo correspondiente al control *RequiredFieldValidator*.

6.3 EL CONTROL RANGEVALIDATOR

El control *RangeValidator*, tal y como indica su nombre, nos permite comprobar que el valor indicado por un usuario dentro de un determinado campo está dentro de un rango de valores predeterminado. En este caso, vamos a ver un ejemplo en el que se le presenta al usuario una página en la que este debe indicar su fecha de nacimiento. Esta fecha no debe ser menor que el 1 de Enero de 1900, ni mayor que la fecha actual. Estos valores mínimo y máximo, que delimitan el rango a evaluar, se establecerán de forma programática y pertenecerán a un control de tipo *RangeValidator*. El código de la página es el siguiente.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>
<%@ Page language="c#" Codebehind="Valid3.aspx.cs" Inherits="MasterSalamanca.Valid3" Au-
toEventWireup="false" %>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" content="C#">
<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">
<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
        <mobile:Form id="Form1" runat="server">
                <mobile:Label id="Label1" runat="server">Fecha de Nacimiento: </mobile:Label>
                <mobile:TextBox id="fecha" runat="server"></mobile:TextBox>
                <mobile:RangeValidator id="validFecha" runat="server" ErrorMessage="La fecha no puede
ser inferior a 1900 ni mayor que la actual" ControlToValidate="fecha" Type="Date"></mobile:RangeValida-
tor>
                <mobile:Command id="ok" runat="server">ok</mobile:Command>
        </mobile:Form>
        <mobile:Form id="Form2" runat="server">
                <mobile:Label id="msg" runat="server">Bienvenido...</mobile:Label>
        </mobile:Form>
</body>
```

El código para establecer los valores del rango, será el siguiente.

```
private void Page_Load(object sender, System.EventArgs e)
{
        validFecha.MinimumValue = "1/1/1900";
        validFecha.MaximumValue = System.DateTime.Today.Day +
                "/" + System.DateTime.Today.Month + "/" +
                System.DateTime.Today.Year;
}
```

El resultado de la validación sería el siguiente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

6.4 EL CONTROL REGULAREXPRESSIONVALIDATOR

Este control, tal y como su propio nombre indica, nos permite realizar comprobaciones del cumplimiento o no de las restricciones establecidas por una expresión regular, por parte de la información especificada por el usuario. Evidentemente, este control es sumamente potente y con él podemos comprobar correos electrónicos, teléfonos, cuentas bancarias o cualquier otra información que tenga un cierto formato reconocido.
El código correspondiente al formulario es el siguiente.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Valid4.aspx.cs" Inherits="MasterSalamanca.Valid4" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="Label1" runat="server" BreakAfter="False">Teclee su correo electr&#243;nico</mobile:Label>

                <mobile:TextBox id="correo" runat="server"></mobile:TextBox>

                <mobile:RegularExpressionValidator id="RegularExpressionValidator1" runat="server" ErrorMessage="El correo indicado es incorrecto." ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" ControlToValidate="correo"></mobile:RegularExpressionValidator>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                <mobile:Command id="ok" runat="server">Ok</mobile:Command>
        </mobile:Form>
</body>
```

En el código anterior, vemos que la expresión regular correspondiente al correo electrónico, indicada como valor para el atributo *ValidationExpression*.

6.5 EL CONTROL CUSTOMVALIDATOR

Este control, a diferencia del resto, no provee ningún sistema de validación automático. Este control nos permite crear nuestra propia validación, que el sistema invocará de forma automática. Para realizar la validación, deberemos sobrescribir el método que se encarga de gestionar el evento *ServerValidate*. El método que debemos sobrescribir es el siguiente.

```
void ServerValidate(Object source, ServerValidateEventArgs args)
```

En el ejemplo a realizar para ilustrar el uso de este control, vamos a realizar una validación que nos permite comprobar que el número indicado por el usuario es un número par.
El código correspondiente al formulario será el que se muestra a continuación.

```
<%@ Page language="c#" Codebehind="Valid5.aspx.cs" Inherits="MasterSalamanca.Valid5" AutoEventWireup="false" %>

<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">
<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
	<mobile:Form id="Form1" runat="server">
		<mobile:Label id="Label1" runat="server" BreakAfter="False">Escriba un n&#250;mero par</mobile:Label>
		<mobile:TextBox id="num" runat="server"></mobile:TextBox>
		<mobile:CustomValidator id="CustomValidator1" runat="server" ErrorMessage="Por favor, indique un número par" ControlToValidate="num"></mobile:CustomValidator>
		<mobile:Command id="ok" runat="server">Ok</mobile:Command>
	</mobile:Form>
</body>
```

Del código asociado a este control, cabe destacar únicamente el método de validación y la línea del método de inicialización de la página que establece el método *ServerValidate* como el método encargado de gestionar la validación correspondiente al control *CustomValidator1*, que será el control de validación para comprobar que el usuario introduce un número par.

```
private void InitializeComponent()
{
	this.CustomValidator1.ServerValidate += new System.Web.UI.WebControls.ServerValidateEventHandler(this.ServerValidate);
	this.Load += new System.EventHandler(this.Page_Load);

}

void ServerValidate(object source, System.Web.UI.WebControls.ServerValidateEventArgs args)
{
	try
	{
		int n = Int32.Parse(num.Text);
		if (n%2 == 0)
		{
			args.IsValid = true;
		}
		else
		{
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                    args.IsValid = false;
                }
        }
        catch (FormatException e)
        {
                args.IsValid = false;
        }
}
```

6.6 EL CONTROL VALIDATIONSUMMARY

El control *ValidationSummary*, agrupará todos los mensajes de error producidos por todos los controles de validación incluidos en una página. Así, podremos mostrar todos los mensajes de error juntos o tratar esta información de aquella forma que creamos más adecuada.

Evidentemente, este control deberá ir acompañando a otros controles de validación. Una vez realizadas las validaciones por parte de cada uno de estos controles de validación, el control *ValidationSummary* presentará un listado conjunto de todos los mensajes de error.

El control *ValidationSummary* se puede incluir tanto dentro del propio formulario que contiene el resto de los controles de validación, como en un formulario a parte.

El siguiente ejemplo, es una versión mejorada del ejemplo en que se solicitaba al usuario su nombre y la clave de acceso por duplicado. En este caso, para mostrar los mensajes de error lo que vamos a hacer es tener un formulario adicional, de tal forma que todos los mensajes de error se muestren conjuntamente.

El código correspondiente a la página que contiene tres formularios es el siguiente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Valid2.aspx.cs" Inherits="MasterSalamanca.Valid2" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

<mobile:Label id=Label1 runat="server" BreakAfter="False">Nombre: </mobile:Label>

<mobile:TextBox id=nombre title=Nombre: runat="server" EnableViewState="False"></mobile:TextBox>

<mobile:Label id=Label2 runat="server" BreakAfter="False">Clave: </mobile:Label>

<mobile:TextBox id=clave runat="server" Password="True"></mobile:TextBox>

<mobile:Label id=Label3 runat="server" BreakAfter="False">Confirmaci&#243;n Clave:</mobile:Label>

<mobile:TextBox id=clave2 runat="server" Password="True"></mobile:TextBox>

<mobile:RequiredFieldValidator id=validaNombre runat="server" ErrorMessage="Indique su nombre" ControlToValidate="nombre"></mobile:RequiredFieldValidator>

<mobile:RequiredFieldValidator id=validaClave runat="server" ErrorMessage="Indique su clave de acceso" ControlToValidate="clave"></mobile:RequiredFieldValidator>

<mobile:CompareValidator id=validClaves runat="server" ErrorMessage="Las claves indicadas no coinciden" ControlToValidate="clave2" ControlToCompare="clave"></mobile:CompareValidator>

<mobile:Command id=ok runat="server">ok</mobile:Command>

        </mobile:Form>

        <mobile:Form id="Form2" runat="server">

<mobile:Label id=msg runat="server">Bienvenido...</mobile:Label>

        </mobile:Form>

        <mobile:Form id="Form3" runat="server">

<mobile:ValidationSummary id=errores runat="server" FormToValidate="Form1"></mobile:ValidationSummary>

        </mobile:Form>

</body>
```

El código del botón encargado de enviar la información del primer formulario, comprobará si la página es válida. En caso de no serlo, nos mostrará el formulario que contiene el control *ValidationSummary* para mostrar todos los mensajes de error.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
private void ok_Click(object sender, System.EventArgs e)
{
        if (Page.IsValid)
        {
                ActiveForm = Form2;
        }
        else
        {
                ActiveForm = Form3;
        }
}
```

El resultado de la validación dentro del terminal es el siguiente.

7 APARIENCIA Y HOJAS DE ESTILO

Como hemos visto hasta este punto, el entorno MIT se encarga de *renderizar* en último término la página configurada en el formulario, de tal forma que pueda ser visualizada correctamente en el terminal concreto que ha realizado la petición. Si bien este sistema es automático, el propio MIT nos provee un procedimiento orientado a proporcionar al usuario un mayor control sobre el resultado final.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Para controlar a más bajo nivel el proceso de *renderizado*, tenemos dentro de los formularios móviles algunas propiedades que denominaremos, propiedades de estilo. Estas propiedades nos permiten modificar aspectos visuales de nuestra página, como puede ser la fuente de los textos o el color de los mismos. Por supuesto, todas estas características de visualización controladas con las propiedades de estilo sólo tienen efecto en aquellos terminales que soportan dichas características. Los terminales que no presenten pantalla en color evidentemente no podrán mostrar texto en color.
Las propiedades de estilo dentro de MIT, presentan ciertas similitudes con las hojas de estilo habituales (CSS), si bien, estas últimas presentan muchas más capacidades y son mucho más ricas.
A parte de estas capacidades de concreción en lo que a la apariencia se refiere, también tenemos ciertos estilos predefinidos que podemos aplicar a nuestros textos. Estos formatos predefinidos, son especificados a través del atributo *StyleReference*. Los terminales presentarán estos formatos predefinidos de la forma más adecuada. Los formatos predefinidos son aplicables únicamente a los controles *Label*. Los posibles valores a especificar son los siguientes:

- Title
- Error
- SubCommand

El siguiente ejemplo muestra cómo se especifican los formatos de texto. El siguiente ejemplo muestra varios casos, en algunos se indica un formato predefinido a través de *StyleReference* y en otros casos se hace de forma específica.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Formatos.aspx.cs" Inherits="MasterSalamanca.Formatos" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="Label1" runat="server" Font-Bold="True" Font-Size="Small">Mobile Internet Toolkit</mobile:Label>

                <mobile:Label id="Label2" runat="server" Font-Bold="True" Font-Size="Large" Font-Italic="True" Font-Name="Book Antiqua" ForeColor="Blue">Mobile Internet Toolkit</mobile:Label>

                <mobile:Label id="Label4" runat="server" Font-Italic="True" Font-Name="Forte" ForeColor="Red">Mobile Internet Toolkit</mobile:Label>

                <mobile:Label id="Label3" runat="server" StyleReference="error">Mobile Internet Toolkit</mobile:Label>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
            <mobile:Label id="Label5" runat="server" StyleReference="subcommand">Mobile Internet
Toolkit</mobile:Label>
            <mobile:Label id="Label6" runat="server" StyleReference="title">Mobile Internet
Toolkit</mobile:Label>
        </mobile:Form>
</body>
```

La siguiente captura muestra el resultado en el terminal de los diferentes formatos.

7.1 HOJAS DE ESTILO

Las hojas de estilo, nos permiten definir estilos a aplicar a los diferentes controles. Un estilo no será más que un conjunto de características referentes a la visualización del resultado de los controles. Para utilizar dentro de nuestras páginas una hoja de estilo, lo único que tendremos que hacer es introducir dentro de dicha página un control de tipo *StyleSheet*.

Dentro del control *StyleSheet* incluido en la página, definiremos los diferentes estilos que componen la hoja. Estos estilos serán los mismos que se pueden aplicar directamente sobre un control, pero de esta forma, los definiremos una vez y los utilizaremos las veces que deseemos. De esta forma, también se hace mucho más fácil el proceso de modificación de los estilos de una página.

Una vez que hemos definido un estilo, se puede aplicar a los controles a través del atributo *StyleReference* de los mismos, tal y como hemos visto anteriormente.

El siguiente ejemplo define dos estilos dentro de una hoja y aplica dichos estilos a una serie de controles.

```
<%@ Page language="c#" Codebehind="HojasEstilo.aspx.cs" Inherits="MasterSalamanca.HojasEstilo" AutoEventWireup="false" %>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="Label1" runat="server" StyleReference="Estilo 1">Mobile Internet
Toolkit</mobile:Label>

                <mobile:Label id="Label2" runat="server" StyleReference="Estilo 2">Mobile Internet
Toolkit</mobile:Label>

                <mobile:TextBox id="TextBox1" runat="server" StyleReference="Estilo 1"></mobile:Text-
Box>

        </mobile:Form>

        <mobile:StyleSheet id="StyleSheet1" runat="server">

                <mobile:Style Font-Size="Large" Font-Name="Gill Sans Ultra Bold" Font-Bold="True"
BackColor="Yellow" ForeColor="Red" Alignment="Center" Name="Estilo 1"></mobile:Style>

                <mobile:Style Font-Size="Large" Font-Italic="True" BackColor="Aqua" Alignment="Left"
Name="Estilo 2"></mobile:Style>

        </mobile:StyleSheet>

</body>
```

Es lógico que una aplicación web mantenga una apariencia similar en todos sus elementos, de tal forma que de la sensación de consistencia. Bien, para hacer más sencillo este proceso, el MIT no provee un sistema por el que podemos utilizar hojas de estilo definidas de forma externa, es decir, no es necesario que estén incrustadas en la propia página, tal y como se mostraba en el último ejemplo.
Este sistema, se basa en la definición de la hoja estilo en un fichero separado. El proceso comienza con la definición de un control propio (un fichero *ascx*), que tendrá un control correspondiente a una hoja de estilo incrustada. Esta hoja de estilo, definirá los formatos generales a aplicar en la aplicación. Una vez que tenemos esto, introduciremos en nuestra página un control del tipo *StyleSheet* y en este control, estableceremos el atributo *ReferencePath*, de tal forma que haga referencia al control creado con los estilos generales. A partir de este momento, podremos utilizar los estilos generales para los controles incluidos dentro de la página.
El resultado del anterior ejemplo en terminales con diferentes capacidades será el siguiente.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

8 CREACIÓN DE CONTROLES

A parte de los controles pertenecientes al MIT que hemos visto y estudiado hasta este punto, el marco de trabajo nos permite la creación de nuestros propios controles, encapsulando así ciertos comportamientos que necesitemos. Una vez creados estos controles, pueden ser usados dentro de los formularios, de forma similar a cómo se hace el uso de controles propios del MIT.

8.1 CONTROLES DE USUARIO

Los controles de usuario nos permitirán reutilizar de forma sencilla y rápida, lógica creada para otras páginas. Para crear un control de usuario, usaremos el mismo formato de fichero que utilizamos para la creación de páginas basadas en formularios, pero en este caso, el nombre del fichero no tendrá extensión *aspx*, sino que su extensión será *ascx*.

Para definir un control de usuario, simplemente introduciremos dentro del mismo, aquellos controles (pertenecientes al MIT o no) que deseemos agrupar en el control de usuario que estamos creando. Una vez que tengamos este hecho finalizado, podremos utilizar el control de usuario como un único elemento, de tal forma que en realidad estamos modelando varios controles unidos. Es sencillo imaginar la gran mejora desde el punto de vista la reutilización de código y desde el punto de vista del mantenimiento de nuestra aplicación.

El siguiente código muestra un ejemplo de un control de usuario, en el que se presentan dos etiquetas con un texto fijo. Este ejemplo nos permite tener una posible cabecera para todas las páginas de nuestra aplicación. El código correspondiente al control de usuario es el siguiente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Control Language="c#" AutoEventWireup="false" Codebehind="UserControl1.ascx.cs" Inherits="MasterSalamanca.UserControl1" TargetSchema="http://schemas.microsoft.com/Mobile/WebUserControl" %>

<mobile:Label id="Label2" runat="server" StyleReference="title" Alignment="Center">MOBILE INTERNET TOOLKIT</mobile:Label>

<mobile:Label id="Label1" runat="server" Alignment="Center" Font-Italic="True">Manuel J. Prieto</mobile:Label>
```

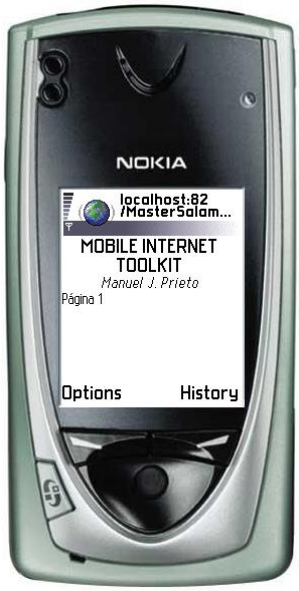
Una vez que tenemos este control definido, podremos utilizarlo dentro de cualquier página, insertándolo dentro de un formulario. Por cierto, en el ejemplo anterior se puede comprobar que los controles de usuario no necesitan contener un formulario, como si ocurre en las páginas estándar. El siguiente ejemplo muestra un ejemplo de utilización del control de usuario definido en el ejemplo anterior.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="UsoControlUsuario.aspx.cs" Inherits="MasterSalamanca.UsoControlUsuario" AutoEventWireup="false" %>

<%@ Register TagPrefix="uc1" TagName="UserControl1" Src="UserControl1.ascx" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <uc1:UserControl1 id="UserControl11" runat="server"></uc1:UserControl1>

                <mobile:Label id="Label1" runat="server">P&#225;gina 1</mobile:Label>

        </mobile:Form>

</body>
```

El resultado final será el siguiente.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



8.2 CONTROLES PROPIOS

Los controles propios (*custom controls*), son codificados directamente como clases. Esta codificación se realizará en cualquiera de los lenguajes de programación soportados por ASP.NET y será compilada. Esta forma de creación de controles es más complicada que la vista en el punto anterior, lo que hemos denominado controles de usuario, pero también es significativamente más potente.

Para la creación de controles propios, podremos partir de los controles propios del MIT, o crear el control partiendo desde cero. Evidentemente es mucho más sencillo y apropiado, en la mayoría de los casos, partir de un control ya creado y aprovechar así parte de su lógica y su comportamiento. En concreto, las técnicas por las que podemos optar para crear nuestros controles es el siguiente:

Herencia – En este caso nuestro control extenderá un control ya existente. Para realizar esto, evidentemente tendremos que extender la clase que define el control del que vamos a partir. Al extender la clase en cuestión, lo que haremos será añadir nuestros propios atributos, métodos o cualquier otra cosa que consideremos necesario, o modificar alguno de los asistentes.

Composición – En este caso lo que haremos será crear un control propio, basado en la combinación de varios controles ya existentes.

Composición dependiente de dispositivo – Este tipo de controles se compondrán de manera diferente, según el dispositivo que realice la petición. Por ejemplo, un control puede ser compuesto con una serie de imágenes para un determinado grupo de dispositivos, o puede ser compuesto como una lista de elementos textuales en el caso de que el grupo de dispositivos sea otro.

Partiendo de cero – En este caso, nuestro control será creado a partir directamente de la clase *System.Web.UI.MobileControl*, y proveerá de forma completa el propio control todos los comportamientos y funcionalidades.

Evidentemente, las dos primeras técnicas expuestas son mucho más sencillas que las dos últimas.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

8.2.1 USO DE LA HERENCIA

Esta forma de creación de controles, basada en la extensión o modificación de controles ya existentes, es muy útil y sencilla. Sobre todo, cabe destacar que el proceso de *renderización* o creación del contenido final que será enviado al terminal, ya está implementado y podemos utilizarlo de forma totalmente transparente.

En el siguiente ejemplo, vamos a crear un control que extiendo un control de tipo lista estándar. El nuevo control, en nuestro caso, presentará la capacidad de ordenar el listado de ítems en función de algún criterio concreto. Este ejemplo no se muestra de forma completa, pero sería un control de tipo lista con un comportamiento de personalización al usuario embebido. El código correspondiente a la clase que extiende la lista es el siguiente.

```
namespace Shadow
{
	using System;
	using System.Data;
	using System.Drawing;
	using System.Web;
	using System.Web.Mobile;
	using System.Web.UI.MobileControls;
	using System.Web.UI.WebControls;
	using System.Web.UI.HtmlControls;
	public class MobileWebUserControl5 : System.Web.UI.MobileControls.List
	{
		public MobileWebUserControl5()
		{
		}
		private void Page_Load(object sender, System.EventArgs e)
		{
			// Cojo el userId inicializado por el constructor de "GeneralPage" y construyo
			// la página
			// Cambio el orden de los elementos
			// Borro la lista anterior y añado la nueva
		}
	}
}
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En el ejemplo anterior no se modifican los elementos, pero se presenta la arquitectura, de tal forma que el objetivo es que el lector comprenda como funcionaría un posible control que extienda al control *List* que ya conocemos.

8.2.2 CONTROL BASADO EN COMPOSICIÓN

Como ya se ha indicado, los controles de composición se crean a partir de la unión o composición de uno o más controles. En este caso, de forma similar a lo que ocurre con la herencia, son los controles ya existentes los encargados de crear el contenido que se envía al terminal en último término.
Cuando se crea un control de composición, generalmente es apropiado partir o heredar de un control de tipo *Panel*. Esto es debido a que el control *Panel* es tratado de forma especial por el entorno de ejecución del MIT. Por ejemplo, el uso de un control *Panel* como control base, nos asegura que el nuevo control no va a ser dividido en varias páginas durante el proceso de ejecución.
El resultado del uso de este tipo de controles, es similar a los controles de usuario. En este caso, lo que se hace es incluir los controles de forma programática. El siguiente ejemplo muestra un ejemplo sencillo de cómo sería el método principal en el proceso de creación de un control compuesto.

```
protected override void CreateChildControls()
{
        texto1 = new Label();
        texto2 = new TextBox();
        texto1.Text = "Su nombre";
        texto2.Text = "Escriba aquí";
        Controls.Add(texto1);
        Controls.Add(texto2);

        ChildControlsCreated = true;
}
```

8.2.3 CREACIÓN AVANZADA DE CONTROLES

Cuando un control necesita un resultado muy concreto, es decir, debemos controlar de forma muy cercana el proceso de *rederizado* del control. Este proceso se realiza dentro del método *Render*, y por lo tanto será en este punto dónde deberemos trabajar.
En el caso de los dispositivos móviles, como se ha indicado en repetidas ocasiones a lo largo de este documento, el proceso de *renderizado* es especialmente delicado, ya que deberemos adaptarnos lo mejor posible al terminal con el que se está trabajando en cada momento.
El entorno de ejecución del MIT provee lo que se denomina adaptadores al dispositivo, que nos permiten trabajar con varios grupos de terminales o dispositivos. Cuando se realiza una petición por parte de un determinado terminal, los adaptadores al dispositivo aplicados a cada control, son seleccionados

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

por parte del entorno de ejecución, de tal forma que sean los más adecuados para el terminal en cuestión. Los adaptadores de dispositivo, implementan una serie de métodos y propiedades, en los que el control relegará las funciones que necesitan de una gestión dependiente del dispositivo.
Para configurar nuestros propios adaptadores, tendremos que configurar dichos adaptadores para el control en cuestión, dentro del fichero *Web.config*. Este fichero configura totalmente la aplicación, entre otros puntos, los adaptadores que trabajarán sobre los controles.

9 ADAPTACIÓN AL DISPOSITIVO

Se ha hablado en varias ocasiones a lo largo de este documento sobre el proceso de adaptación al dispositivo. A parte de la adaptación automática que realiza el propio entorno, podremos configurar lo que denominaremos filtros de dispositivo para nuestra aplicación. Los filtros de dispositivo se definen dentro del fichero *Web.config* y estos filtros serán utilizados para seleccionar el contenido más adecuado para cada terminal. Un terminal puede encajar dentro de varios filtros. Los filtros se definen dentro de la sección *deviceFilters* del fichero *Web.config*. El siguiente listado muestra la definición de una serie de filtros.

```
<system.web>
  <deviceFilters>
    <filter name="IsColor" compare="IsColor" argument="true" />
    <filter name="IsPocketIE" compare="Browser" argument="Pocket IE" />
    <filter name="IsHTML" compare="PreferredRenderingMIME"
       argument="text/html" />
  </deviceFilters>
</system.web>
```

Dentro de la definición de un filtro, el nombre identifica el mismo y este nombre será el utilizado dentro de los formularios para hacer referencia a dicho filtro. El resto de atributos indican los datos necesarios para hacer la comprobación de aplicabilidad del filtro. Las características válidas para comprobar si un filtro es aplicable o no, están definidas dentro de la clase *System.Web.UI.Mobile.Controls.MobileCapabilities*.
Para definir un contenido específico para un determinado grupo de dispositivos, podemos utilizar la etiqueta *<DeviceSpecific>*. El siguiente ejemplo muestra cómo indicar un contenido específico, según el terminal desde el que se realiza la petición.

```
<mobile:Label runat="server" >
  <DeviceSpecific>
    <Choice Filter="IsPocketIE" Text="Corriendo en Pocket IE" />
    <Choice Filter="IsHTML" Text="Corriendo en un terminal HTML" />
    <Choice Text="Corriendo en un terminal genérico" />
  </DeviceSpecific>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<mobile:Label>
```

Cuando se realiza la petición de la página, el marco de trabajo comprueba cada una de las etiquetas *Choice*, en el orden en el que están definidos dentro del formulario. Si el filtro indicado corresponde con la petición, el filtro será seleccionado y será aplicado. En el ejemplo anterior, podemos comprobar cómo el último *Choice* no tiene filtro, por lo que será seleccionada como elemento por defecto. Es decir, en caso de que ninguno de los otros filtros sea aplicable, se aplicará este filtro. Este filtro por defecto es opcional.
Como queda plasmado en el ejemplo anterior, cuando se selecciona un filtro, lo que se hace es establecer el valor del atributo *Text* del control. Esta operación se denomina sobreescritura de propiedades. Esta técnica nos permite cambiar las características del control en función del terminal que hace la petición. Así, podemos cambiar el texto, tal y como acabamos de ver, podemos cambiar una imagen o incluso podríamos cambiar la apariencia del control, variando el valor del estilo del control.

9.1 USO DE PLANTILLAS

Con el objetivo de poder realizar personalización de una forma más sencilla dentro de los controles, las páginas compuestas por formularios, proveen una funcionalidad que permite el uso de plantillas. Las plantillas se pueden utilizar sobre los siguientes controles:

- Form
- Panel
- List
- ObjectList

Dentro de las plantillas aplicables a los formularios, tenemos la capacidad de modificar la cabecera y el pie del formulario. Estas operaciones se realizarán a través de las etiquetas *HeaderTemplate* y *FooterTemplate*. También tenemos la posibilidad de utilizar la etiqueta *ScriptTemplate*, de tal forma que insertaremos contenido inmediatamente después de la etiqueta *head* en HTML o *card* en WML. El siguiente ejemplo muestra un ejemplo de cómo sería el uso de plantillas con el fin antes descrito. Hay que destacar, que en este caso no se tiene en cuenta el dispositivo que hace la petición.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="PlantillasEj.aspx.cs" Inherits="MasterSalamanca.PlantillasEj1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

```
            <mobile:TextView id="TextView1" runat="server">Este es el texto de la página</mobile:TextView>
            <mobile:DeviceSpecific id="DeviceSpecific1" runat="server">
                <Choice>
                    <HeaderTemplate>
                        <mobile:Label id="Label1" runat="server">Cabecera de la p&#225;gina</mobile:Label>
                    </HeaderTemplate>
                    <FooterTemplate>
                        <mobile:Label id="Label2" runat="server">Pie de la p&#225;gina</mobile:Label>
                    </FooterTemplate>
                </Choice>
            </mobile:DeviceSpecific>
        </mobile:Form>
</body>
```

El resultado sería el siguiente en el terminal.

Podemos ampliar un poco más el ejemplo anterior, de tal manera que tengamos en consideración desde qué terminal se realiza la petición a la hora de crear el contenido. En el siguiente ejemplo se muestra esta funcionalidad, de tal forma que se cambia la cabecera en el caso de que el terminal corresponda con un determinado grupo de terminales.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>
<%@ Page language="c#" Codebehind="PlantillaEj2.aspx.cs" Inherits="MasterSalamanca.PlantillaEj2" Au-
toEventWireup="false" %>
<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" content="C#">
<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">
<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
        <mobile:Form id="Form1" runat="server">
                <mobile:DeviceSpecific id="DeviceSpecific1" runat="server">
                        <Choice Filter="isPocketIE" Xmlns="http://schemas.microsoft.com/mo-
bile/html32template">
                                <HeaderTemplate>
                                        Esto es un POCKET IE
                                </HeaderTemplate>
                        </Choice>
                        <Choice Xmlns="http://schemas.microsoft.com/mobile/html32template">
                                <HeaderTemplate>
                                        <mobile:Label id="Label1" runat="server">Cabecera
gen&#233;rica</mobile:Label>
                                </HeaderTemplate>
                                <FooterTemplate>
                                        <mobile:Label id="Label2" runat="server">Pie
gen&#233;rico</mobile:Label>
                                </FooterTemplate>
                        </Choice>
                </mobile:DeviceSpecific>
        </mobile:Form>
</body>
```

Como hemos indicado cuando enumerábamos los controles que aceptaban plantillas, estas también pueden aplicarse a los controles *List* y *ControlList*. Estas plantillas se utilizarán normalmente para mejorar la presentación en algunos terminales. En estos casos, además de la cabecera y el pie de la lista, también podemos modificar el modo en que se mostrarán los elementos de la lista. Así, si estamos trabajando con un terminal que soporta HTML, podremos mostrar la lista utilizando una tabla. En la cabecera de la lista escribiremos las etiquetas correspondientes a la creación de la tabla, cada

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

ítem podrá ser una fila de la tabla y para cerrar la tabla al final, usaremos el elemento de la plantilla que nos permite controlar el pie.
Trabajar con plantillas en el caso de los controles tipo *Panel*, lo que nos permite es insertar bloques de lenguaje de marcado final, dentro de la aplicación. Esto mismo se puede hacer usando los elementos de la plantilla del formulario, pero en el caso del control *Panel*, el elemento *ContentTemplate* reemplaza cualquier otro control o contenido que estuviera dentro del *Panel*.

10 GESTIÓN DEL ESTADO DE LA APLICACIÓN

Para mantener el estado de la aplicación, podemos usar cualquiera de las técnicas disponibles dentro de ASP.NET. En este documento veremos alguno de ellos. Por ejemplo, los formularios WEB disponibles dentro de ASP.NET son capaces de mantener su propio estado durante las diferentes peticiones del cliente. Cuando una propiedad adquiere un valor, bien programáticamente o bien mediante una acción del usuario, dicho valor es guardado automáticamente como parte del estado del control.
Normalmente, esta técnica se basa en el envío al cliente de una variable oculta que es retornada al servidor como parte de la respuesta. Si bien esto es el funcionamiento habitual dentro de ASP.NET, en el caso del trabajo con terminales móviles, el estado de la aplicación no se envía al cliente, debido entre otras cosas, a las limitaciones de ancho de banda que se sufren en las comunicaciones móviles. En lugar de esto, lo que hacen es salvar el estado como parte de la sesión del usuario dentro del servidor.
El siguiente ejemplo muestra de forma sencilla este tipo de funcionamiento. Tenemos dos etiquetas dentro de un formulario y vemos cómo el contenido de las etiquetas se mantiene entre peticiones.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="GestionEstado1.aspx.cs" Inherits="MasterSalamanca.GestionEstado1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="val" runat="server" Visible="False">1</mobile:Label>

                <mobile:Label id="Label1" runat="server">Label</mobile:Label>

                <mobile:Command id="Reload" runat="server" StyleReference="subcommand">Reload</mobile:Command>

        </mobile:Form>

</body>
```

El código que tenemos detrás de esta página para que funcione, es el que se muestra a continuación, si bien, se han eliminado las partes sin relevancia.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
private void Page_Load(object sender, System.EventArgs e)
{
        int valor = int.Parse(val.Text);
        valor++;
        val.Text = "" + valor;
        Label1.Text = "Es la " + valor + " que ves esta página";

}
private void Reload_Click(object sender, System.EventArgs e)
{
        ActiveForm = Form1;
}
```

El resultado en ejecución es el siguiente.

Como hemos visto, el estado de la aplicación se mantiene en el servidor. Esto presenta un problema en el caso de que el usuario seleccione la funcionalidad de atrás o *back* del navegador. En este caso, es posible que se pierda la sincronización entre el servidor y el cliente.

Para solucionar este problema en la medida de lo posible, los formularios móviles mantienen una pequeña historia del estado, dentro de la sesión del usuario, tal y como ya hemos dicho. Cada identificador enviado al cliente, se corresponde con una determinada posición dentro de dicha historia. Así, con el identificador de la página, el servidor es capaz de sincronizar la historia con la página activa en cada momento. El tamaño de esta historia es configurable por el desarrollador. El tamaño por

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

defecto es seis y este valor se puede modificar en el fichero *web.config*, modificando el valor del atributo *sessionStateHistorySize*.
Debido a que el estado de la aplicación es salvado dentro de la sesión, es posible que dicho estado expire o caduque. Cuando hay problemas para cargar la información correspondiente al estado de la sesión, se llama al evento *OnViewStateExpire*. En este lugar podemos reconstruir el estado o dejar la implementación por defecto, que lanzará una excepción. Por ejemplo, imaginemos que tenemos un control de tipo lista en el que los elementos son creados al comienzo. En este caso, tendríamos el contenido de la lista gestionado por el entorno de ejecución y en el caso de que se llamara a *OnViewStateExpire*, podríamos volver a crear dinámicamente los elementos de la lista.
El método de gestión del estado de la aplicación que acabamos de ver, es muy sencillo, pero puede causar una cierta merma en el rendimiento del servidor, si el número de peticiones es elevado y la cantidad de información que contienen los controles también es muy alta. Para desactivar la gestión del estado de la aplicación en un control y en todos sus hijos, simplemente tenemos que establecer la variable *EnableViewState* de dicho control a *false*.
Por defecto, la gestión de la sesión en ASP.NET necesita la escritura por parte del servidor de una *cookie* en el cliente. El problema se presenta cuando nos enfrentamos a algunos dispositivos móviles, que no tiene soporte para *cookies*. En este caso, para que la aplicación sea capaz de trabajar correctamente con estos terminales, deberemos configurar la aplicación para que mantenga el estado de la sesión sin utilizar *cookies*. Cuando una aplicación trabaja sin *cookies*, automáticamente inserta una clave o identificador de sesión dentro de las diferentes URLs de la aplicación.

11 GESTIÓN DE LAS CAPACIDADES DEL DISPOSITIVO

A estas alturas no debe haber dudas sobre la orientación del MIT a cubrir las necesidades de un abanico muy amplio de terminales. Dentro de este amplio abanico, tenemos terminales con capacidades muy diversas en lo que se refiere a la resolución de pantalla, el número de colores o el tipo de elementos de interacción con el usuario disponibles.
Para permitir al desarrollador el trabajo con estas diferentes capacidades, el MIT incluye un componente que nos permite acceder y gestionar dichas capacidades. Este componente identifica el dispositivo desde el que se hace la petición, recupera toda la información posible sobre dicho dispositivo y pone a disposición del desarrollador dicha información. Esto se hace a través del objeto *MobileCapabilities*.
Entre otras informaciones, el objeto *MobileCapabilities* nos proporciona información sobre el modelo del terminal que hace la petición. El siguiente ejemplo muestra cómo podríamos recuperar dicho modelo.

```
MobileCapabilities capabilities =
   (MobileCapabilities)Request.Browser;
String deviceModel = capabilities.MobileDeviceModel;
```

12 ANEXO A – REFERENCIA DE LOS CONTROLES DEL MIT

12.1 CONTROL Form

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
Method	Método usado para enviar el formulario al servidor.	Post Get
PagerStyle	El estilo usado para las propiedades de paginación del control	
Paginate	Indica si los contenidos deben ser divididos en páginas.	True False
StyleReference	Estilo que usa el control.	error subcommand title
Títle	Título usado para identificar el formulario	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

12.2 CONTROL Panel

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Paginate	Indica si los contenidos deben ser divididos en páginas.	True False
StyleReference	Estilo que usa el control.	error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.3 CONTROL Label

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
StyleReference	Estilo que usa el control.	error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.4 CONTROL TextBox

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
MaxLength	El máximo número de caracteres que se puede introducir	True False
Numeric	Indica si la entrada se restringe a datos numéricos	

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Password	Indica si el texto debe ser ocultado	True False
Size	El tamaño esperado de texto introducido	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.5 CONTROL TextView

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.6 CONTROL Command

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
CausesValidation	Indica si se debe activar la validación en el formulario cuando se active	True False
CommandArgument	El argumento asociado con el comando	
CommandName	El nombre asociado con el comando	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
Format	Indica la apariencia visual del control	Button Link
ImageUrl	La url de la imagen a mostrar	
SoftKeyLabel	Etiqueta usada para el comando cuando se muestra como un *tecla*. Sólo se aplica a dispositivos que usen teclas para los comandos	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.7 CONTROL Link

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
NavigateUrl	El URL o el formulario al que enlaza	
SoftKeyLabel	Etiqueta usada para el comando cuando se muestra como un *tecla*. Sólo se aplica a dispositivos que usen teclas para los comandos	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.8 CONTROL PhoneCall

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
AlternateFormat	Formato del texto que aparecerá para los dispositivos que no soportan llamadas	
AlternateURL	La URL o formulario de destino para dispositivos que no soportan llamadas	
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
PhoneNumber	Número de teléfono a llamar para teléfonos que soporten las llamadas	
SoftKeyLabel	Etiqueta usada para el comando cuando se muestra como un *tecla*. Sólo se aplica a dispositivos que usen teclas para los comandos	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.9 CONTROL Image

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
AlternateText	El texto alternativa a mostrar en el caso de que no se pueda mostrar la imagen	
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ImageURL	La URL de la imagen	
NavigateURL	La URL o el form al que navegar	
SoftKeyLabel	Etiqueta usada para el comando cuando se muestra como un *tecla*. Sólo se aplica a dispositivos que usen teclas para los comandos	
StyleReference	Estilo que usa el control.	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.10 CONTROL List

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
DataMember	Tabla usada como fuente cuando un DataSet es usado como fuente de datos	
DataSource	DataSource del que se obtienen los datos de la lista	
DataTextField	Especifica qué propiedad del item de datos se usa para determinar el texto del item en la lista	
DataValueField	Especifica qué propiedad del item de datos se usa para determinar el valor del item en la lista	
Decoration	Formato de la lista	Bulleted Numbered
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ItemCount	Número de items a mostrar en el control. El valor 0 indica *paginación automática*	
Items	Colección de elementos de la lista	
ItemAsLink	Indica si los elementos de la lista deben ser interpretados como hiperenlaces	True False
ItemsPerPage	Indica el número de items a mostrar en cada página del control. El valor 0 indica *paginación automática*	
StyleReference	Estilo que usa el control.	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.11 CONTROL SelectionList

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
DataMember	Tabla usada como fuente cuando un DataSet es usado como fuente de datos	
DataSource	DataSource del que se obtienen los datos de la lista	
DataTextField	Especifica qué propiedad del item de datos se usa para determinar el texto del item en la lista	
DataValueField	Especifica qué propiedad del item de datos se usa para determinar el valor del item en la lista	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ItemCount	Número de items a mostrar en el control. El valor 0 indica *paginación automática*	
Items	Colección de elementos de la lista	
Rows	El número de líneas visibles a mostrar	
SelectType	El tipo de selección de la lista	DropDown ListBox Radio MultiSelectListBox Checkbox
StyleReference	Estilo que usa el control.	Error subcommand title
Title	El título mostrado sobre el área de entrada de datos en los dispositivos que lo soporten	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.12 CONTROL ObjectList

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
AutoGenerateFields	Indica si los campos son generados automáticamente en tiempo de ejecución, en bsae a una fuente de datos asocidad	True False
BackColor	Color de fondo del control	
BackCommandText	Texto para el enlace de retorno en la vista de detalles	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
Commands	Colección de comandos para cada elemento de la listas	
DataMember	Tabla usada como fuente cuando un DataSet es usado como fuente de datos	
DataSource	DataSource del que se obtienen los datos de la lista	
DefaultCommand	Comando lanzado cuando un elemento es seleccionado	
DetailsCommandText	Texto para el enlace a la vista de detalles	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Fields	Colección de campos que componen la vista de detalles	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ItemCount	Número de items a mostrar en el control. El valor 0 indica *paginación automática*	
ItemsPerPage	Número de items a mostrar en cada página del control. El valor 0 indica *paginación por defecto*	
LabelField	Campo de la fuente de datos a usar como representación compacta de los elementos de la lista	
LabelStyle	El estilo aplicado a la etiqueta de cabecera	
MoreText	Texto para el enlace a *más detalles y comandos*	

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

StyleReference	Estilo que usa el control.	Error subcommand title
TableFields	Campos de la fuente de datos a usar en una representación extensa de los elementos de la lista	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.13 CONTROL StyleSheet

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
ReferencePath	Camino relativo al fichero de usuario que contiene el control StyleSheet	
TemplateStyle	El estilo usado como plantilla	

12.14 CONTROL Calendar

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
CalendarEntryText	Texto para el enlace que permite la selección de una fecha. El enlace es sólo usado cuando se necesitan varios pasos	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
FirstDayOfWeek	El día de la semana que aparece primero	Sunday Monday...
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
SelectedDate	La fecha seleccionada en cada momento	
SelectionMode	Indica si se seleccionan dias, semanas o meses	Day DayWeek DayWeekMonth

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

ShowDayHeader	Será *true* si muestra la cabecera de días de la semana	True False
StyleReference	Estilo que usa el control.	Error subcommand title
TableFields	Campos de la fuente de datos a usar en una representación extensa de los elementos de la lista	
Visible	Indica si el control está visible y se ha procesado.	True False
VisibleDate	El mes que aparece	
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.15 CONTROL AdRotator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
AdvertisementFile	Fichero XML que contiene los anuncios	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ImageKey	Nombre del elemento XML que especifica la URL de la imagen a descargar	
KeywordFilter	Palabra clave para limitar la selección de anuncios	
NavigateURLKey	Nombre del elemento XML que indica la URL de la página web que recuperar	
StyleReference	Estilo que usa el control.	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.16 CONTROL RequiredFieldValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
InitialValue	Valor inicial del campo a validar	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.17 CONTROL CompareValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToCompare	Id del control con el que comparar	
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
Operator	Operación de comparación a aplicar	Equal Not equal GreaterThan GreaterThanEqual LessThan LessThanEqual DataTypeCheck
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Type	Tipo de datos de los valores a comparar	String Integer Double Date Currency
ValueToCompare	Valor con el que comparar	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.18 CONTROL RangeValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
MaximumValue	Valor máximo del control a validar	
MinimumValue	Valor mínimo del control a validar	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Type	Tipo de datos de los valores a comparar	String Integer Double Date Currency
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.19 CONTROL RegularExpressionValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
ValidationExpression	Expresión regular que determinad la validez	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.20 CONTROL CustomValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.21 CONTROL ValidationSummary

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BackLabel	Etiqueta del enlace al formulario que causo el error	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
FormToValidate	Identificador del formulario a validar	
HeaderText	Texto que aparece en la cabecera	
StyleReference	Estilo que usa el control.	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Bogdan Okresa Durik. (2017) Organisational Metamodel for Large-Scale Multi-Agent Systems: First Steps Towards Modelling Organisation Dynamics. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
2. Jörg Bremer, Sebastian Lehnhoff. (2017) Decentralized Coalition Formation with Agent-based Combinatorial Heuristics. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
3. Rafael Cauê Cardoso, Rafael Heitor Bordini. (2017) A Multi-Agent Extension of a Hierarchical Task Network Planning Formalism. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
4. Enyo Gonçalves, Mariela Cortés, Marcos De Oliveira, Nécio Veras, Mário Falcão, Jaelson Castro (2017). An Analysis of Software Agents, Environments and Applications School: Retrospective, Relevance, and Trends. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
5. Eduardo Porto Teixeira, Eder M. N. Goncalves, Diana F. Adamatti (2017). Ulises: A Agent-Based System For Timbre Classification. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
6. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
7. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
8. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. *IEEE Access*.
9. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
10. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. *International Journal of Robust and Nonlinear Control*, *28*(16), 5087-5102.
11. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
12. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
13. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
14. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
15. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
16. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
17. Choon, Y. W., Mohamad, M. S., Deris, S., Illias, R. M., Chong, C. K., Chai, L. E., … Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. *PLoS ONE*, *9*(7). https://doi.org/10.1371/journal.pone.0102744
18. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In *FUSION 2014 - 17th International Conference on Information Fusion*. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
19. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. *Expert Systems with Applications*, *41*(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003
20. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. *Applied Mathematics and Computation*, *270*, 756–767. https://doi.org/10.1016/j.amc.2015.08.059

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

21. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). Random finite set-based Bayesian filters using magnitude-adaptive target birth intensity. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637788&partnerID=40&md5=bd8602d6146b014266cf07dc35a681e0
22. Casado-Vara, R., de la Prieta, F., Prieto, J., & Corchado, J. M. (2018, November). Blockchain framework for IoT data quality via edge computing. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (pp. 19-24). ACM.
23. Costa, Â., Novais, P., Corchado, J. M., & Neves, J. (2012). Increased performance and better patient attendance in an hospital with the use of smart agendas. *Logic Journal of the IGPL*, *20*(4), 689–698. https://doi.org/10.1093/jigpal/jzr021
24. Rodríguez, S., De La Prieta, F., Tapia, D. I., & Corchado, J. M. (2010). Agents and computer vision for processing stereoscopic images. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6077 LNAI). https://doi.org/10.1007/978-3-642-13803-4_12
25. Rodríguez, S., Gil, O., De La Prieta, F., Zato, C., Corchado, J. M., Vega, P., & Francisco, M. (2010). People detection and stereoscopic analysis using MAS. In *INES 2010 - 14th International Conference on Intelligent Engineering Systems, Proceedings*. https://doi.org/10.1109/INES.2010.5483855
26. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. *Information Sciences*, *180*(10), 2029–2043. https://doi.org/10.1016/j.ins.2009.12.032
27. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. *International Journal of Ambient Computing and Intelligence*, *v 1*, *n 1*(1), 15–26. https://doi.org/10.4018/jaci.2009010102
28. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. *Expert Systems with Applications*, *36*(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
29. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. *BMC Bioinformatics*, *10*. https://doi.org/10.1186/1471-2105-10-187
30. Fernández-Riverola, F., Díaz, F., & Corchado, J. M. (2007). Reducing the memory size of a Fuzzy case-based reasoning system applying rough set techniques. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *37*(1), 138–146. https://doi.org/10.1109/TSMCC.2006.876058
31. Méndez, J. R., Fdez-Riverola, F., Díaz, F., Iglesias, E. L., & Corchado, J. M. (2006). A comparative performance study of feature selection methods for the anti-spam filtering domain. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4065 LNAI*, 106–120. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-33746435792&partnerID=40&md5=25345ac884f61c182680241828d448c5
32. Di Mascio, T., Vittorini, P., Gennari, R., Melonio, A., De La Prieta, F., & Alrifai, M. (2012, July). The Learners' User Classes in the TERENCE Adaptive Learning System. In 2012 IEEE 12th International Conference on Advanced Learning Technologies (pp. 572-576). IEEE.
33. Chamoso, P., Rivas, A., Martín-Limorti, J. J., & Rodríguez, S. (2018). A Hash Based Image Matching Algorithm for Social Networks. In *Advances in Intelligent Systems and Computing* (Vol. 619, pp. 183–190). https://doi.org/10.1007/978-3-319-61578-3_18
34. Sittón, I., & Rodríguez, S. (2017). Pattern Extraction for the Design of Predictive Models in Industry 4.0. In *International Conference on Practical Applications of Agents and Multi-Agent Systems* (pp. 258–261).
35. García, O., Chamoso, P., Prieto, J., Rodríguez, S., & De La Prieta, F. (2017). A serious game to reduce consumption in smart buildings. In *Communications in Computer and Information Science* (Vol. 722, pp. 481–493). https://doi.org/10.1007/978-3-319-60285-1_41
36. Palomino, C. G., Nunes, C. S., Silveira, R. A., González, S. R., & Nakayama, M. K. (2017). Adaptive agent-based environment model to enable the teacher to create an adaptive class. Advances in Intelligent Systems and Computing (Vol. 617). https://doi.org/10.1007/978-3-319-60819-8_3

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

37. Canizes, B., Pinto, T., Soares, J., Vale, Z., Chamoso, P., & Santos, D. (2017). Smart City: A GECAD-BISITE Energy Management Case Study. In *15th International Conference on Practical Applications of Agents and Multi-Agent Systems PAAMS 2017, Trends in Cyber-Physical Multi-Agent Systems* (Vol. 2, pp. 92–100). https://doi.org/10.1007/978-3-319-61578-3_9
38. Chamoso, P., de La Prieta, F., Eibenstein, A., Santos-Santos, D., Tizio, A., & Vittorini, P. (2017). A device supporting the self-management of tinnitus. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 10209 LNCS, pp. 399–410). https://doi.org/10.1007/978-3-319-56154-7_36
39. Tiago Pinto, Luis Marques, Tiago M Sousa, Isabel Praça, Zita Vale, Samuel L Abreu. (2017) Data-Mining-based filtering to support Solar Forecasting Methodologies. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
40. Casado-Vara, R., & Corchado, J. (2019). Distributed e-health wide-world accounting ledger via blockchain. *Journal of Intelligent & Fuzzy Systems*, *36*(3), 2381-2386.
41. M.ª Belén Aige (2017). The online tourist fraud: the new measures of technological investigation in Spain. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
42. Silvia Susana Toscano (2017). Freedom of Expression, Right to Information, Personal Data and the Internet in the view of the Inter-American System of Human Rights. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1
43. Víctor Corcoba Magaña, Mario Muñoz Organero, Juan Antonio Álvarez-García, Jorge Yago Fernández Rodríguez. (2017) Design of a Speed Assistant to Minimize the Driver Stress. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
44. Miguel Oliver, José Pascual Molina, Antonio Fernández-Caballero, Pascual González. (2017) Collaborative Computer-Assisted Cognitive Rehabilitation System. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
45. Miki Ueno, Toshinori Suenaga, Hitoshi Isahara (2017). Classification of Two Comic Books based on Convolutional Neural Networks. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1
46. Eduardo Munera, Jose-Luis Poza-Lujan, Juan-Luis Posadas-Yagüe, Jose-Enrique Simó-Ten, Francisco Blanes (2017). Integrating Smart Resources in ROS-based systems to distribute services. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Mobile Internet Toolkit

Manuel-Jesús Prieto Martín [1]

[1] Telefónica Investigación y Desarrollo, Spain
mjprieto@telefonica.es

Resumen: En este capítulo se presenta el "mobile internet tooolkit", se introduce Flash Lite 1.1 y se analiza su potencial. También se se presenten ejemplos prácticos. Macromedia Flash Lite es una versión del repro-ductor de Macromedia Flash, diseñada y desa-rrollada con el objetivo de funcionar sobre dispositivos mó-viles, especialmente teléfonos mó-viles. Con este objetivo y teniendo en cuenta las importantes limitaciones que presentan este tipo de dispositivos, en cuanto a memoria, espacio de almacenamiento o capacidad de proce-so, Flash Lite presenta un compromiso de equilibrio entre estas limitaciones, y las caracterísi-cas y funcionalidades habituales en Macromedia Flash. Teniendo en cuenta la reciente aparición del producto y lo relativamente costoso que es conse-guir un estándar que los fabricantes instalen e integren en sus termi-nales, parece claro que, en el futuro, Macromedia Flash será una de las plataformas habituales de ejecución de aplicacio-nes dentro de los terminales móviles.

Palabras clave: Programación móvil; Macromedia Flash

Abstract. This chapter introduces the "mobile internet tooolkit", introduces Flash Lite 1.1 and analyses its potential. Practical examples are also presented. Macromedia Flash Lite is a version of the Macromedia Flash player, designed and developed to work on mobile devices, especially mobile phones. With this objective and taking into account the important limitations of this type of device, in terms of memory, storage space or process capacity, Flash Lite presents a compromise of balance between these limitations, and the usual features and functionalities of Macromedia Flash. Considering the recent appearance of the product and how relatively expensive it is to guide a standard that manufacturers install and integrate into their terminals, it seems clear that in the future, Macromedia Flash will be one of the usual platforms for running applications within mobile terminals.

Keywords: Mobile programming; Macromedia Flash

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

INTRODUCCIÓN A FLASH LITE 1.1

Macromedia Flash Lite es una versión del reproductor de Macromedia Flash, diseñada y desarrollada con el objetivo de funcionar sobre dispositivos móviles, especialmente teléfonos móviles. Con este objetivo y teniendo en cuenta las importantes limitaciones que presentan este tipo de dispositivos, en cuanto a memoria, espacio de almacenamiento o capacidad de proceso, Flash Lite presenta un compromiso de equilibrio entre estas limitaciones, y las características y funcionalidades habituales en Macromedia Flash. Actualmente existen dos versiones de Flash Lite, la versión 1.0 y la versión 1.1.

A grandes rasgos, y como idea genérica, podemos estructurar a Macromedia Flash Lite de la siguiente manera. Por una parte, proporciona un entorno de procesado de imágenes y recursos gráficos en general, que se encarga de gestionar y mostrar estos elementos en la pantalla del dispositivo. Como es habitual en Macromedia Flash, se soportan campos de texto para mostrar textos estáticos, dinámicos o como punto de entrada de texto por parte del usuario.

Ambas versiones de Flash Lite (1.0 y 1.1) soportan los formatos de audio típicos de los dispositivos móviles, como pueden ser MIDI o MFi, así como la gestión estándar de sonidos de Macromedia Flash.

También dispone de un interprete de *actionscript* que soporta la versión de este lenguaje usada en la versión 4 del reproductor estándar de Macromedia Flash, y que soporta además una serie de comandos específicos de la versión Lite, que permiten interactuar con algunas de las capacidades del dispositivo. Esta versión de actionscript, es denominada Flash Lite 1.x ActionScript.

Una característica importante que presenta Macromedia Flash Lite y que sin duda es adecuada para dispositivos móviles, es la conectividad de red. Con esta versión de Flash podremos cargar tanto ficheros *swf* externos como datos externos. También podremos realizar conexiones *http*, lo que sin duda aumenta de forma significativa la riqueza de las aplicaciones que se desarrollen sobre esta plataforma [1-5].

Junto con la conectividad, que es una característica básica de los dispositivos móviles y que Macromedia Flash Lite explota, también podremos aprovechar otras características como es el envío de mensajes cortos (SMS) o la realización de llamadas telefónicas.

Macromedia Flash Lite es soportado en la actualidad por un conjunto significativo de terminales móviles en el mercado. Teniendo en cuenta la reciente aparición del producto y lo relativamente costoso que es conseguir un estándar que los fabricantes instalen e integren en sus terminales, parece claro que en el futuro, Macromedia Flash será una de las plataformas habituales de ejecución de aplicaciones dentro de los terminales móviles. Hay varios puntos clave que apoyan esta idea y que el mercado no puede obviar:

- Cada día los terminales móviles tienen mayores capacidades y permiten ejecutar aplicaciones más complejas.

- Los usuarios demandan aplicaciones complejas, especialmente de entretenimiento, que doten de valor añadido sus terminales.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- Flash es una herramienta perfecta para desarrollar aplicaciones con un coste bajo y con unos resultados gráficos espectaculares. Este punto es importante en cualquier entorno, pero en el caso de los teléfonos móviles y el uso masivo de los mismos, cobra especial importancia.

- Es enorme la base de desarrolladores y diseñadores que conocen Macromedia Flash, y que por lo tanto pueden comenzar en un tiempo muy corto a crear aplicaciones que funcionen sobre dispositivos móviles.

- Los terminales móviles demandan aplicaciones sencillas, pero con unos resultados gráficos buenos. Teniendo en cuenta las limitaciones en la interfaz de usuario de los terminales (teclado y pantalla, principalmente), es obvio que un teléfono móvil no es el entorno adecuado para un procesador de texto o para un juego de estrategia 3D. Sin duda, es un entorno más adecuado para una aplicación de consulta de la meteorología gráficamente atractiva o para un juego sencillo y adictivo.

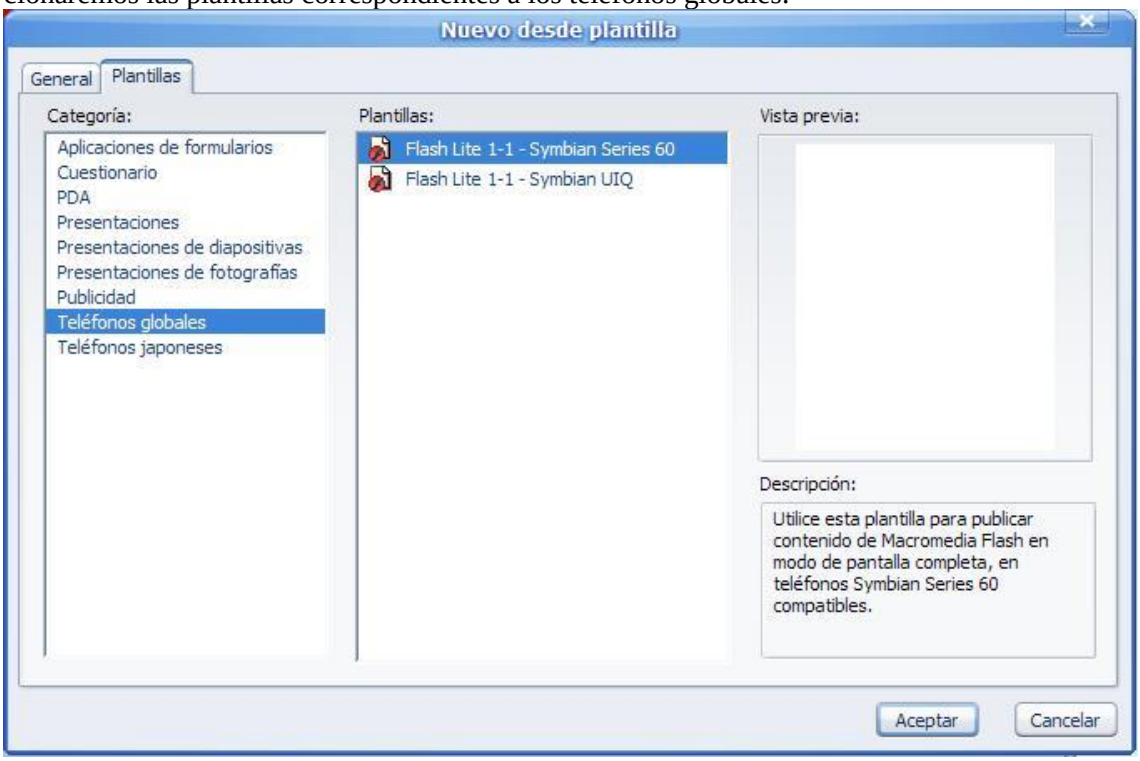
Actualmente el reproductor de Flash Lite se ejecuta dentro de los terminales como una aplicación independiente (*standalone*). De forma contrario a lo que ocurre en entorno de ordenadores habitual de Macromedia Flash, en el entorno de los terminales móviles, la heterogeneidad entre estos es un hecho y las características en las que difieren unos terminales de otros son muchas e importantes. Así, como ocurre en todos los desarrollos destinados a ser ejecutados en un teléfono móvil, el diseñador o desarrollador tiene que tener en cuenta qué familia de terminales será su objetivo de tal forma que el resultado final sea lo suficientemente bueno y que aproveche todas las capacidades y características del terminal. También se debe conocer y tener presente desde el primer momento el tipo de contenidos que el terminal o terminales objetivos permite o soporta. Por ejemplo, algunos dispositivos permiten realizar salvapantallas, otros permiten la integración de las aplicaciones Flash Lite dentro de páginas *web*, pero no todas estas características están presentes en todos los terminales. Este punto es especialmente importante porque en función del tipo de contenido y del propio dispositivo, dispondremos dentro de la aplicación de algunas capacidades. Así, una aplicación que es ejecutada como salvapantallas, no tendrá permiso para realizar conexiones de red. Para ayudar al desarrollador en este entorno tan diverso, la versión 8 de Macromedia Flash permite probar las aplicaciones sobre los diferentes entornos y así facilitar de forma importante este trabajo [6-11].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

LA PRIMERA APLICACIÓN: HOLA MUNDO

Como método de acercamiento al entorno de desarrollo que provee Macromedia Flash Professional 8 para las aplicaciones Flash Lite, vamos a realizar una primera aplicación. Esta aplicación tan sencilla, se limitará a mostrar por pantalla un texto y una imagen, ya que el principal objetivo es conocer el entorno de Macromedia Flash.

Para crear la aplicación comenzaremos por seleccionar la opción *Nuevo* en el menú de *Fichero*. En la pantalla que aparece seleccionaremos la plantilla a aplicar al nuevo documento Flash a crear, y seleccionaremos las plantillas correspondientes a los teléfonos globales.

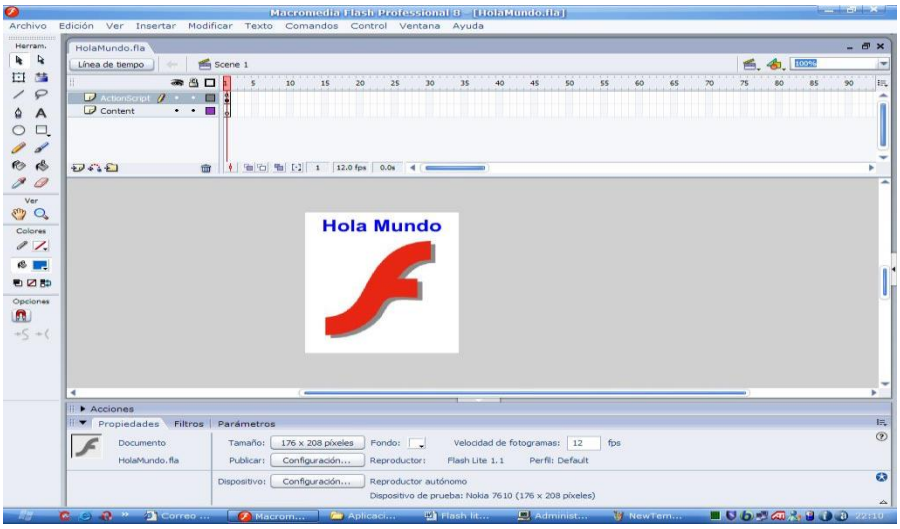


En esta pantalla nueva, y tal como se puede ver en la imagen anterior, seleccionamos la plantilla correspondiente a la serie 60 de Symbian. Así, se creará el nuevo documento y este estará preconfigurado y preparado para la plataforma que le hemos indicado. En este momento ya tenemos ante nosotros el entorno habitual de Macromedia Flash y podremos comenzar a trabajar de igual forma que cuando se desarrollan las aplicaciones Flash habituales.

En el espacio configurado para el documento, insertaremos el texto “Hola Mundo” y una imagen, que es el objetivo que nos hemos propuesto para esta primera aplicación. El resultado de estas acciones es el que muestra la siguiente imagen.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



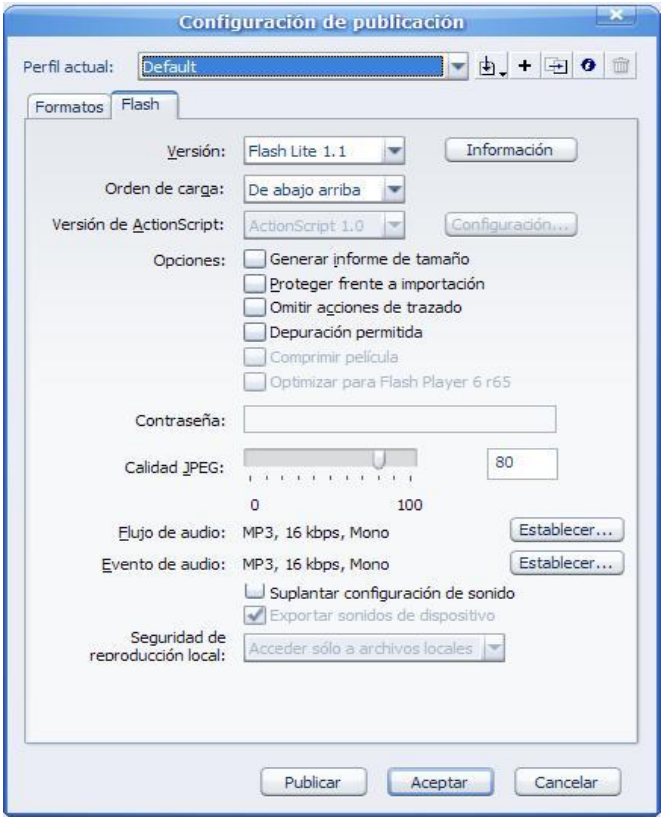
En este momento ya estamos dispuestos a probar la ejecución de nuestra aplicación dentro del entorno. Es obvio que la aplicación no hará nada y simplemente mostrará el contenido descrito por pantalla, pero es suficiente en estos momentos para nuestros propósitos.

Para probar la aplicación únicamente tendremos que seleccionar el menú Control y la opción Probar Película. Se abrirá una ventana que nos permite configurar el entorno en el que queremos probar la aplicación. La siguiente imagen muestra cómo sería la siguiente imagen.

En la parte izquierda de la pantalla, podemos seleccionar el dispositivo que queremos utilizar para probar la aplicación, dentro de la lista que se nos presenta.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Con estos pasos ya conocemos de qué forma podemos crear nuestras aplicaciones para Flash Lite y ya sabemos cómo probarlas. Como vemos, la herramienta Macromedia Flash Professional 8, nos ayuda de forma importante en esta labor [12-15].
A continuación, vamos a ver cómo podemos configurar de manera manual una aplicación, para Flash Lite y para un terminal concreto. Comenzaremos igual que antes, creando un documento nuevo, pero sin recurrir en este caso a las plantillas. Una vez creado el documento, iremos a la configuración de la publicación en el menú de fichero. En la ventana que aparece, dentro de la pestaña correspondiente a Flash, seleccionaremos la versión de flash que queremos utilizar como base para el nuevo documento que acabamos de crear.

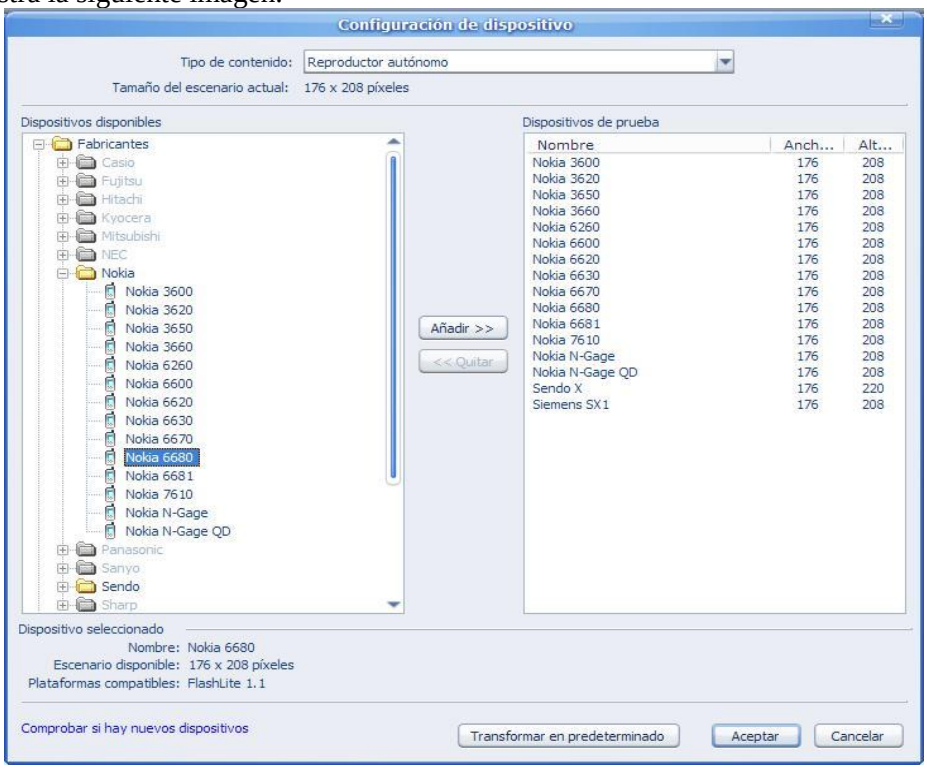


Una vez que volvemos a la pantalla principal, seleccionaremos la opción de configuración del dispositivo dentro del menú de propiedades de la película. Esta operación también se puede hacer a través del menú de fichero (*Fichero/Configuración de publicación*).

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En la nueva pantalla que se abre para permitirnos la configuración del dispositivo, tendremos la posibilidad de seleccionar el tipo de contenido que queremos crear, que en nuestro caso será una aplicación independiente (*Reproductor anónimo*). En la lista de posibles dispositivos que se presenta, buscaremos aquel que deseemos y lo seleccionaremos. En la parte inferior de la pantalla, se nos informa de las características del dispositivo que hayamos seleccionado en cada momento. Esta información comprende el nombre del dispositivo, el tamaño de la pantalla y las versiones de la plataforma soportadas por el dispositivo. En nuestro caso seleccionaremos un terminal de la marca Nokia, tal y como muestra la siguiente imagen.



Es importante que el tamaño de nuestra película coincida con el tamaño de la pantalla del terminal, por lo que tendremos que cambiar el tamaño de la película a través del botón correspondiente en la ventana de propiedades.

Ahora podremos comenzar a trabajar en nuestro documento o película y podremos probarlo tal y como hicimos anteriormente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

GESTIÓN DE COMANDOS EN EL EMULADOR

El emulador que utilizaremos para el desarrollo de nuestras aplicaciones presenta algunas teclas de comando que podremos programar, a parte de las teclas alfanuméricas que tiene cualquier teléfono. Estas teclas son las que tienen todos los terminales, para descolgar o moverse por la pantalla, entre otros. La siguiente imagen muestra estas teclas. Las teclas alfanuméricas son obvias y las teclas especiales que usaremos en la mayoría de las aplicaciones, son las teclas de la fila superior, tanto la tecla grande con las cuatro pequeñas flechas (izquierda, derecha, arriba y abajo), como las teclas que están a cada lado de esta. A estas últimas las llamaremos teclas programables. Es importante destacar que el punto central de la tecla grande, será la tecla de selección y se utilizará de forma masiva en las aplicaciones para seleccionar botones.

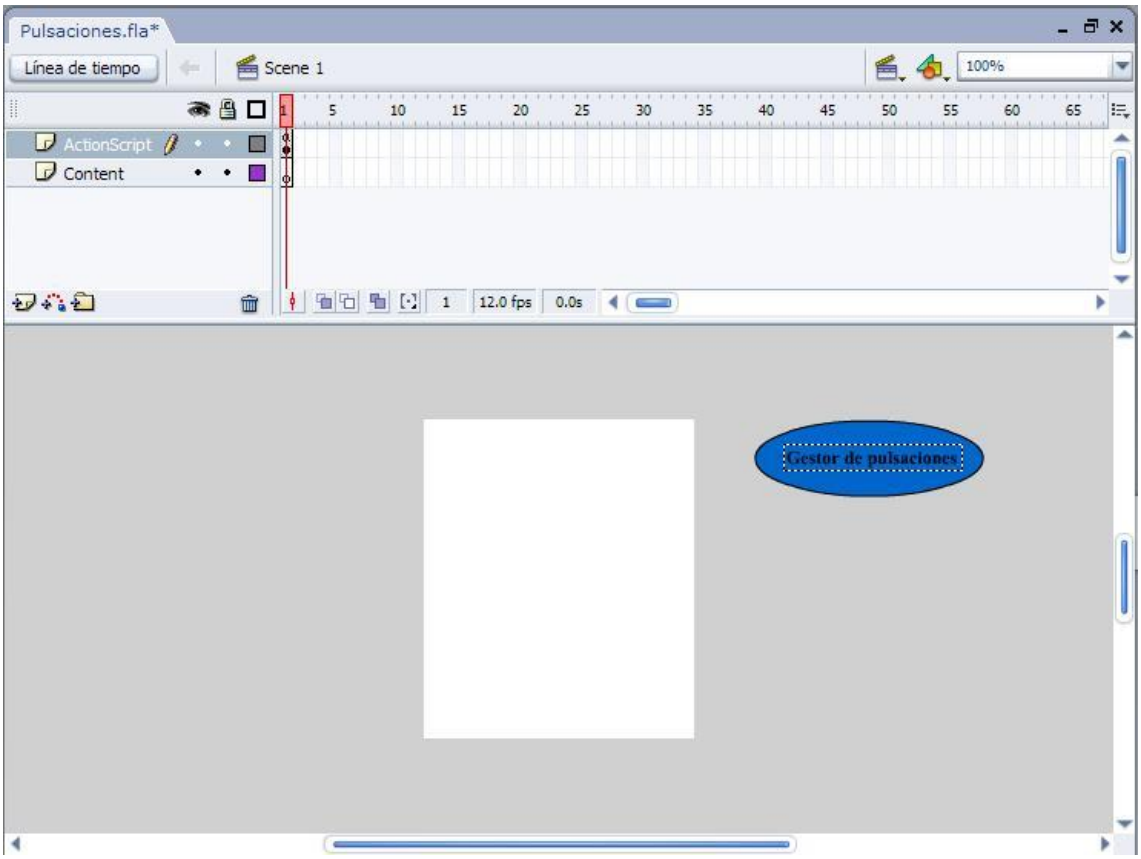
En la mayoría de las aplicaciones, tendremos que programar alguna acción sobre las teclas, de tal forma que la aplicación responda a ciertas acciones por parte del usuario. Para gestionar la pulsación de una tecla por parte del usuario tendremos que controlar, tal y como es común en flash, el siguiente evento:

```
on(keyPress “<Right>")
```

Este caso es para recoger una pulsación de la tecla con la flecha hacia la derecha. Para el resto de teclas, la función a escribir es similar, pero cambiando el texto *<Right>* por otro, para indicar otra tecla. El siguiente ejemplo muestra como capturar los eventos de pulsación de teclas.
Para comenzar, y debido a que esta aplicación tiene como única finalidad el aprendizaje del modo en que se capturan las pulsaciones de las teclas, tendremos un solo botón en la aplicación, y nada más que este botón. Además, este botón no se mostrará por pantalla y nunca será visible. La siguiente imagen muestra esta situación [16-20].

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



Como vemos, la pantalla está en blanco y el botón está fuera de dicha pantalla. Ahora lo único que haremos será añadir código a este botón, de tal manera que podamos comprobar de forma práctica cómo se capturan los eventos de teclado. El código que incluiremos en el botón se muestra en el listado a continuación.

```
on(keyPress "<Down>") {

        trace("Se ha pulsado la tecla de desplazamiento hacia abajo");

}

on(keyPress "<Up>") {

        trace("Se ha pulsado la tecla de desplazamiento hacia arriba");

}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
on(keyPress "<Left>"){
        trace("Se ha pulsado la tecla de desplazamiento hacia la izquierda");
}
on(keyPress "<Right>"){
        trace("Se ha pulsado la tecla de desplazamiento hacia la derecha");
}
on(keyPress "<Enter>"){
        trace("Se ha pulsado la tecla de selección");
}
on(keyPress "0"){
        trace("Se ha pulsado el 0");
}
on(keyPress "5"){
        trace("Se ha pulsado el 5");
}
on(keyPress "9"){
        trace("Se ha pulsado el 9");
}
on(keyPress "*"){
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        trace("Se ha pulsado la tecla de asterisco");

}
```

Este código únicamente mostrará una traza en la ventana de salida del entorno de desarrollo de Flash, pero es suficiente para los objetivos didácticos de este sencillo ejemplo.

SetSoftKeys

El método *SetSoftKeys* nos permite modificar la asignación de las teclas programables del dispositivo. Estas teclas son las que se han comentado anteriormente y que se encontraban a ambos lados del teclado central. No todos los dispositivos nos permiten trabajar con estas teclas. Además, este método únicamente podrá utilizarse cuando el reproductor de Flash Lite se ejecute en modo autónomo. Es decir, cuando el reproductor se inicie dentro del contexto de otra aplicación, como puede ser un navegador, este comando no podrá ser invocado y por lo tanto estas teclas no serán aplicables.
Este comando recibe como parámetros dos textos, que serán los textos asociados a los botones programables y que podremos utilizar para capturar los eventos. El primer texto hacer referencia a la tecla programable izquierda y el segundo hace referencia a la tecla programable derecha. El método retornará un valor indicando si el comando es admitido o no es admitido en el terminal en el momento de ejecución. Este valor será 0 para indicar que está permitido y -1 para lo contrario [21-24].
El siguiente ejemplo muestra cómo activar las teclas programables y como utilizarlas. Para comenzar, crearemos un clip que represente un círculo azul y colocaremos dos en la pantalla del dispositivo. A uno de estos clips lo llamaremos *puntoizquierda* y al otro *puntoderecha*. Para comenzar, en la capa denominada *ActionScript*, incluiremos este código:

```
fscommand2("SetSoftKeys", "Izquierda", "Derecha");

fscommand2("FullScreen", "true");

_root.puntoizquierda._alpha = 0;

_root.puntoderecha._alpha = 0;
```

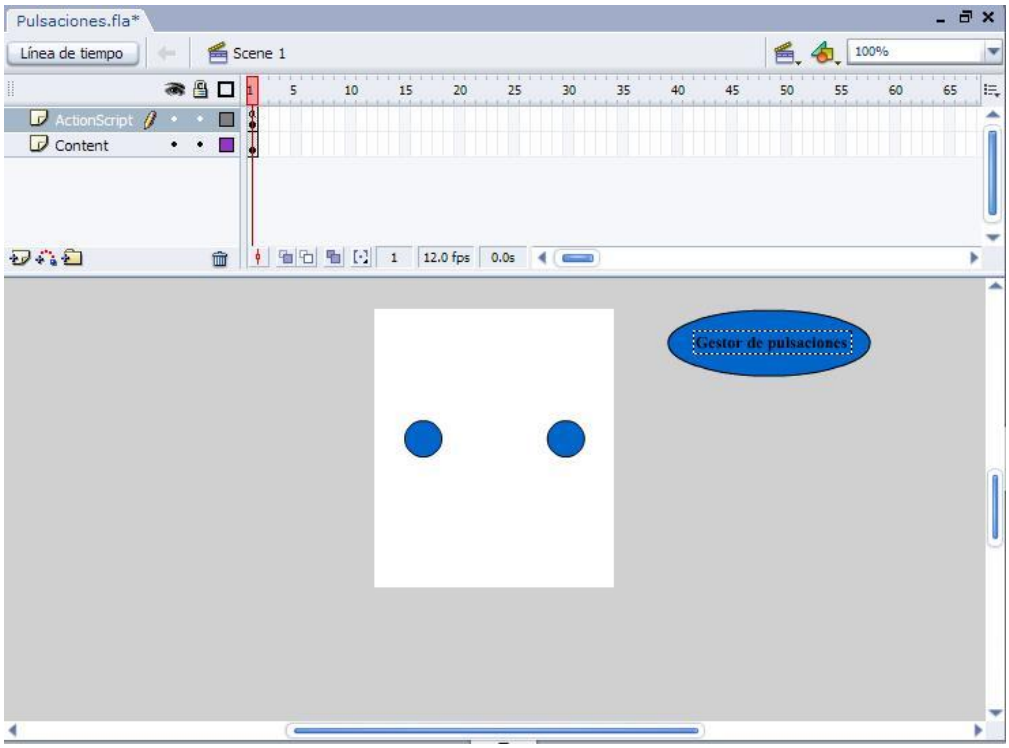
Este sencillo código activa las teclas programables, establece el modo pantalla-completa y hace invisibles los puntos de los que hemos hablado. También incluiremos un botón con el objetivo de controlar los eventos de ratón, igual que ya hicimos anteriormente. El código que asociaremos a este botón es:

```
on (keyPress "<PageUp>") {

        _root.puntoizquierda._alpha = 100;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

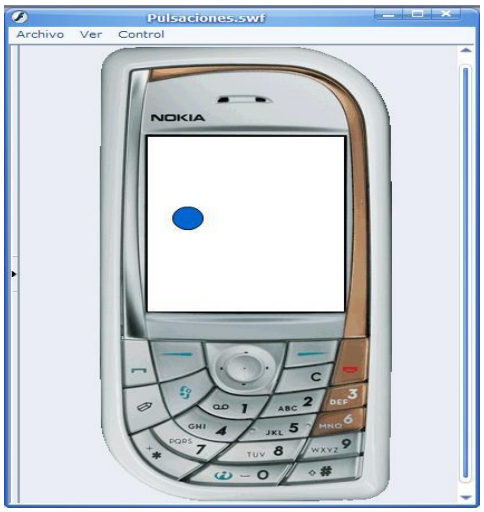
```
        _root.puntoderecha._alpha = 0;

}

on (keyPress "<PageDown>") {

        _root.puntoizquierda._alpha = 0;

        _root.puntoderecha._alpha = 100;

}
```

La tecla *PageUp* corresponde a la tecla programable de la izquierda y *PageDown* corresponde a la tecla programable de la derecha. El código lo que hace es mostrar y ocultar un punto u otro, según se pulse una u otra tecla programable. Las siguientes imágenes ilustran el proceso de desarrollo y ejecución.



ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



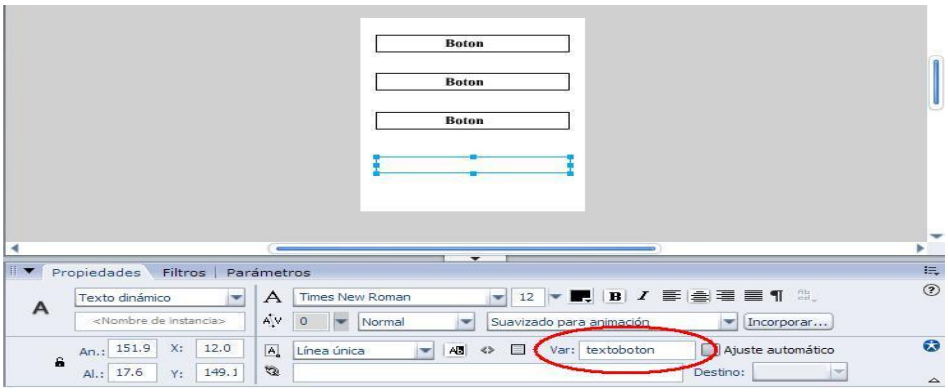
Uso de botones

Los botones dentro de las aplicaciones Flash Lite, funcionan de manera similar a como lo hacen en las aplicaciones flash clásicas. Tenemos los siguientes eventos a controlar en los botones:

- *press* – Se produce al pulsar la tecla de selección sobre el botón.
- *release* – Se produce cuando el usuario suelta la tecla de selección.
- *rollOver* – Se produce cuando el botón es seleccionado.
- *rollOut* – Se produce cuando el botón pierde la selección.

La siguiente aplicación muestra cómo utilizar botones dentro de las aplicaciones Flash Lite. Es una aplicación muy sencilla que presenta tres botones en la pantalla y un campo de texto dinámico, que mostrará el valor de una variable. El contenido de esta variable será modificado en función del botón que se pulse en cada momento. La variable se llama *textoboton* tal y como muestra la imagen siguiente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc



El código de los botones se muestra a continuación. El listado muestra el código de los tres botones, pero es evidente que cada una de las funciones está asociada a un botón.

```
// Botón superior

on(press){

        textoboton = "Botón superior";

}



//Botón intermedio

on(press){

        textoboton = "Botón intermedio";

}



// Botón inferior

on(press){

        textoboton = "Botón inferior";
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
}
```

Por último, vamos a incluir dos capturas de pantalla del emulador, para ver cómo sería la ejecución de esta sencilla aplicación.

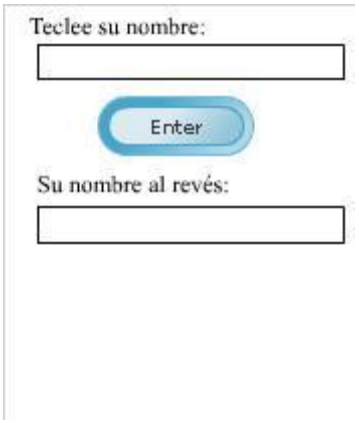
La figura anterior indica que se ha pulsado el botón intermedio, y por eso se muestra el texto al final de la pantalla, y también indica que en este momento, una pulsación de la tecla de selección actuaría sobre el botón inferior.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

TEXTO EN FLASH LITE

Flash lite permite la utilización de campos de texto de tres modos diferentes. Por una parte, podremos utilizar textos estáticos, en los que, como es habitual en flash, se muestra un texto y este no puede ser modificado a lo largo de la ejecución de la película. Este tipo de campos de texto, son utilizamos comúnmente como sencillas etiquetas de texto. Los campos de texto de contenido dinámico, el segundo tipo de campo de texto, nos permite modificar el texto en tiempo de ejecución, a través de la asociación de este campo de texto a una variable [25-30]. El contenido de esta variable será el texto que se muestre en la pantalla. Este tipo de campos de texto lo hemos utilizado ya en el documento en alguno de los ejemplos. Por último, tenemos los campos de texto que permiten interactuar con el usuario, de tal forma que este pueda escribir en el campo de texto e introducir así información en la aplicación.

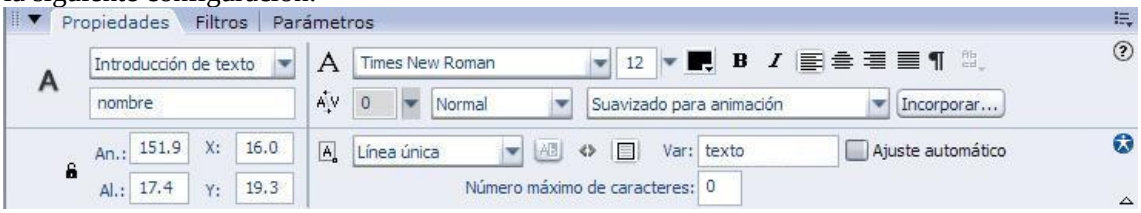
Para mostrar de forma práctica y sencilla cómo es la utilización de estos campos de texto y cuál es el cometido de cada uno de ellos, vamos a escribir un sencillo ejemplo. El ejemplo contiene los tres tipos de campos de texto:



Por una parte tenemos dos etiquetas de texto estático que corresponden con los textos:

- Teclee su nombre:
- Su nombre al revés:

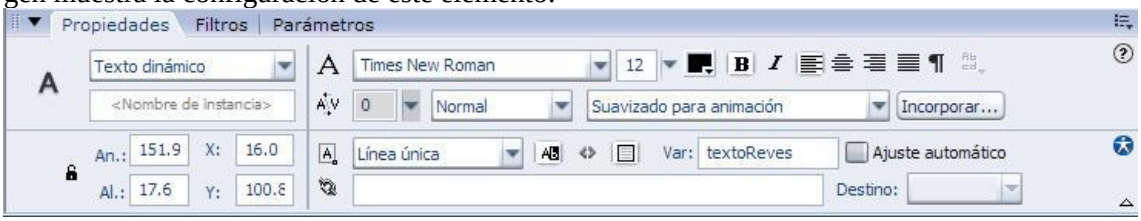
El campo de texto que está debajo de la primera etiqueta, es para introducir texto, tal y como muestra la siguiente configuración:



ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Como vemos en la imagen, el campo de texto es del tipo “Introducción de texto”, y la variable a la que está asociado es texto. Es decir, el texto que escriba el usuario en este elemento, será almacenado dentro de la variable texto. Esta variable no necesita ser definida en ningún sitio.
El campo de texto de la parte inferior de la pantalla, es un campo de texto dinámico, y su cometido es mostrar el contenido de una variable. Antes hemos visto como introducir información en una variable a través de un campo de texto, y ahora vemos el proceso contrario. Este campo de texto, mostrará el contenido de la variable pero no permitirá que se modifique dicha variable. La siguiente imagen muestra la configuración de este elemento:



Como vemos, el tipo es “Texto dinámico” y la variable a la que está asociado este campo de texto es “textoReves”.
Para completar el ejemplo, tenemos un botón con el texto “Enter”, que tendrá el código actionscript que nos permite aportar un poco de lógica al ejemplo. En concreto, el objetivo de esta sencilla aplicación es tomar un texto introducido por el usuario, darle la vuelta a dicho texto, y mostrarlo en el campo de texto correspondiente a través de la variable textoReves. El código actionscript del botón es el siguiente:

```
on(press){

	resultado = "";

	for (i=length(texto);i>0; i--){

		resultado = resultado add substring(texto, i, 1);

	}

	textoReves = resultado;

}
```

Como vemos, el código es muy sencillo. Simplemente define una variable llamada “resultado”, en la que va añadiendo las letras de la cadena original (texto), empezando por la última y finalizando en la primera. Por último, se introduce el resultado final en la variable textoReves.
La siguiente imagen muestra el resultado final en el emulador.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Restricciones en los campos de introducción de texto

Se pueden configurar los caracteres que el usuario puede introducir dentro de un campo de texto. A través del comando *SetInputTextType* se puede realizar este trabajo. Así, por ejemplo, si quisiéramos que en el ejemplo anterior, el usuario únicamente pudiera escribir números dentro del campo de texto, tendríamos que escribir el siguiente comando dentro de la aplicación:

```
fscommand2("SetInputTextType", "texto", "Numeric");
```

Si escribimos esta línea e intentamos introducir algún carácter no numérico dentro del campo de texto, tendremos un aviso por parte del emulador, tal y como muestra la siguiente imagen.

Los posibles modos que se pueden indicar en el control de los caracteres son los que se listan a continuación:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- Numeric – Únicamente se permiten números (0-9).
- Alpha – Sólo se permiten caracteres alfabéticos (a-z, A-Z).
- Alphanumeric – Caracteres alfanuméricos (0-9, a-z, A-Z).
- Latin – Caracteres latinos, es decir, los alfanuméricos y los símbolos de puntuación.
- NonLatin – Únicamente permite los caracteres no latinos.
- NoRestriction – Es el modo predeterminado, sin limitación alguna.

El método fscommand2 mostrado anteriormente, devuelve una variable que nos permite comprobar si la orden se ha ejecutado correctamente o no. Esto es debido a que no todos los dispositivos admiten la gestión de estas restricciones y por lo tanto, a través de esta variable podremos comprobar si nuestra orden tendrá efecto o si por el contrario será ignorada por el terminal. Los posibles valores de la variable son:

- 0 – En caso de error
- 1 – Si la orden se ha ejecutado correctamente y por lo tanto la restricción tendrá efecto.

Fuentes de texto

En Flash Lite disponemos de varias formas para representar las fuentes de los campos de texto. La primera y más sencilla es aplicar alguna de las fuentes de texto que están disponibles de forma genérica:

- _sans
- _serif
- _typewriter

Aún así, el resultado final depende en cierta medida del dispositivo final, pero en cualquier caso este tratará de adecuarse a lo indicado en la aplicación. Esta configuración se realiza a través del menú de propiedades del campo de texto. Tendremos que seleccionar el tipo de fuentes a utilizar y la fuente concreta.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El menú de selección del tipo de texto tiene cinco opciones, pero únicamente tres están disponibles para Flash Lite.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

SONIDOS

Dentro de la versión Lite de Flash tenemos dos tipos de sonido:

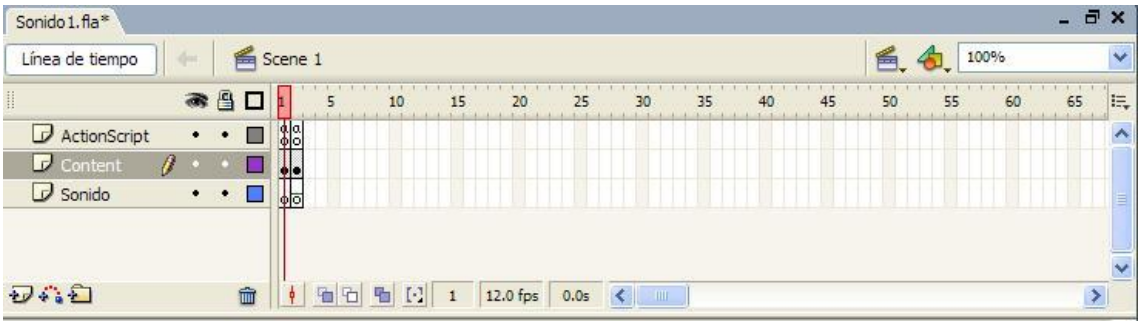
- Sonidos de dispositivo.
- Sonidos estándar o nativo.

Los sonidos de dispositivo se guardan dentro del fichero publicado (con extensión swf) y su formato es el nativo del dispositivo (MIDI o MFi). El sonido será reproducido por el dispositivo, ya que Flash se lo pasa a este para su descodificación y reproducción. En cuanto a las limitaciones de reproducción de este tipo de sonidos, debemos tener en cuenta que Flash Lite no puede sincronizar las animaciones y el sonido del dispositivo y que la versión 1.0 únicamente admite este tipo de sonidos, sonidos de dispositivo. La reproducción de este tipo de sonidos es responsabilidad del dispositivo o terminal, y no del propio entorno Flash, lo que limita sus capacidades [31-25].
En cuanto a los sonidos estándar o nativos únicamente están disponibles en la versión 1.1, tal y como se indicó anteriormente. Flash Lite permite sincronizar este tipo de sonidos con las animaciones, lo que provee a este tipo de sonidos de mucha más utilidad que los sonidos de dispositivo. Además de los formatos de ficheros de sonido que ya hemos comentado antes y que son admitidos como tipos de sonido de dispositivo, los formatos admitidos en Flash Lite 1.1 son SMAF, PCM, WAV, ADPCM y MP3.
Otra clasificación que podemos realizar en los sonidos, son sonidos de evento y sonidos de flujo. Los primeros se reproducen independientemente de la línea de tiempo y se reproducen hasta el final del sonido o hasta que la reproducción se detiene mediante ActionScript. Estos sonidos deben descargarse de forma completa antes de comenzar a reproducirse y una vez más tenemos que decir que su utilidad en las aplicaciones es menor que la de los sonidos de flujo [36-38].
El segundo tipo de sonidos, los sonidos de flujo, están sincronizados con la línea de tiempo y se utilizan para reproducir sonidos sincronizados con la animación. El sonido se detiene cuando la línea de tiempo se detiene y Flash Lite omite fotogramas de la animación si es necesario, para mantener la sincronización. En este caso, sincronizando la línea de tiempo y los sonidos, la utilidad en las aplicaciones es mucho mayor, ya que se puede controlar cuándo comienza a reproducirse el sonido y cuando se para, siempre en sincronía con la línea de tiempo y por tanto con la evolución de la aplicación.
A continuación vamos a realizar un sencillo ejemplo, que nos permitirá comprender de forma más práctica y rápida el trabajo con los sonidos dentro de Flash Lite.

Ejemplos de uso de sonidos

Lo primero que haremos para realizar este ejemplo con sonidos, será modificar la línea de tiempo, para insertar una nueva capa que llamaremos Sonido y extender la línea de tiempo de las capas a dos fotogramas. La siguiente imagen muestra esta estructura:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc



Una vez que tenemos esta estructura de capas, insertaremos el siguiente código en el primer fotograma de la capa denominada ActionScript:

```
fscommand2("FullScreen", true);

stop();
```

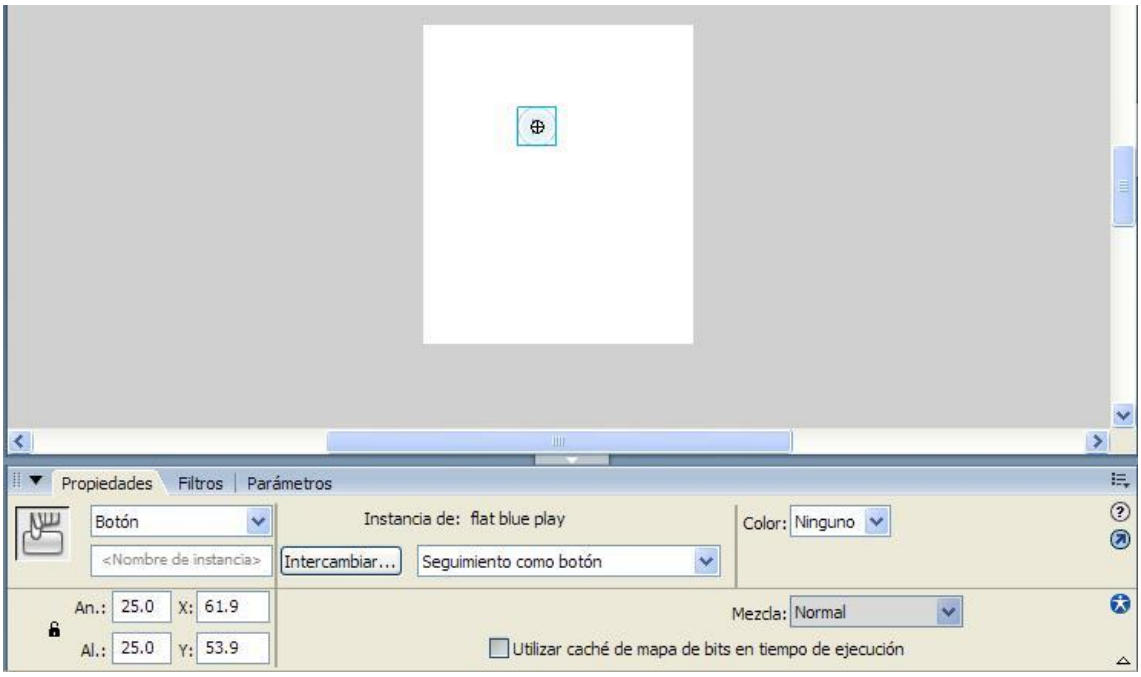
En el segundo fotograma de esta misma capa insertaremos la siguiente línea:

```
stop();
```

Con estas líneas de código simplemente paramos la línea de tiempo de ejecución de la aplicación, en cada fotograma. De esta forma, podemos pasar del fotograma uno al dos y viceversa y así tendremos dos pantallas dentro de la aplicación.
Las dos pantallas de la aplicación, se configuran en la siguiente capa, la que se denomina Content. La siguiente imagen muestra el contenido de este primer fotograma de la capa Content:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



La capa contiene únicamente un botón, tal y como muestra la imagen, que pertenece a la librería estándar de Flash, y que corresponde con el botón de play estándar de cualquier equipo de sonido. Este botón nos permite avanzar hasta la siguiente pantalla, el fotograma dos, con el siguiente código insertado en dicho botón:

```
on(press){

        gotoAndPlay(2);

};
```

Al pasar al segundo fotograma, comenzará a sonar música. Este proceso de gestión del sonido, lo veremos un poco más adelante, pero antes vamos a ver el contenido del segundo fotograma de la capa denominada Content. La siguiente imagen muestra dicho contenido:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El botón que contiene esta capa, es también un botón de la librería de botones estándar de Flash y corresponde con el botón de stop de cualquier equipo de sonido. Junto con el botón, tenemos un campo de texto estático que muestra el mensaje: "reproduciendo sonido".
El código que debe ser insertado en el botón, será el siguiente:

```
on(press){

        stopAllSounds();

        gotoAndPlay(1);

}
```

Como vemos, este botón lo que hace en primer lugar es parar la reproducción de cualquier sonido que esté en ese momento siendo reproducido, y a continuación vuelve a la primera pantalla de la aplicación, que ya hemos explicado anteriormente.
Para finalizar, vamos a ver qué serie de operaciones debemos realizar para conseguir que el sonido comience a reproducirse cuando pulsemos en el botón correspondiente en el primer fotograma. Lo primero que debemos hacer es insertar el sonido a reproducir en la aplicación. En nuestro caso, deseamos reproducir un sonido MIDI, y será un sonido de dispositivo.
Para disponer de este fichero MIDI dentro de nuestra aplicación, primero tendremos que insertar dentro de la biblioteca lo que denominaremos un sonido proxy, que será un sonido nativo, ya que Flash Lite no permite importar sonidos de dispositivo directamente. El formato de los sonidos proxy

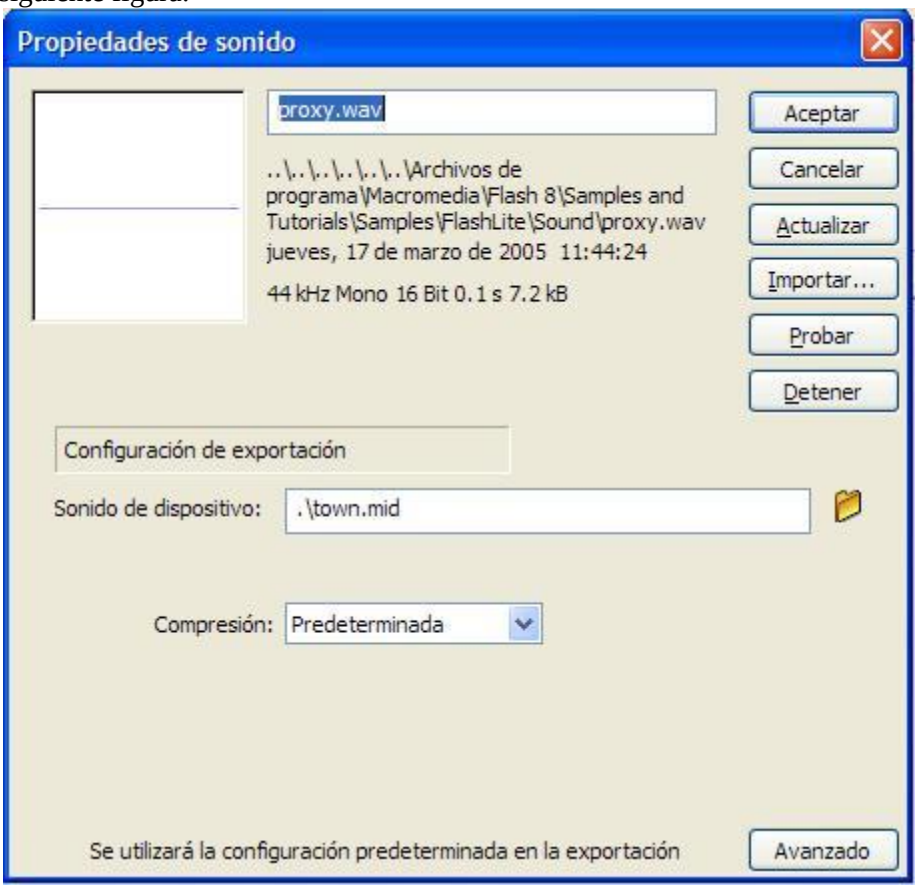
ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

será mp3, wav o aiff. En nuestro caso, el sonido proxy que vamos a utilizar es el fichero proxy.wav, que se distribuye junto con la aplicación Flash (versión 8), como parte de los ejemplos correspondientes a Flash Lite. En concreto, podemos encontrar este fichero en el directorio:

[Directorio_instalación]\Samples and Tutorials\Samples\FlashLite\Sound

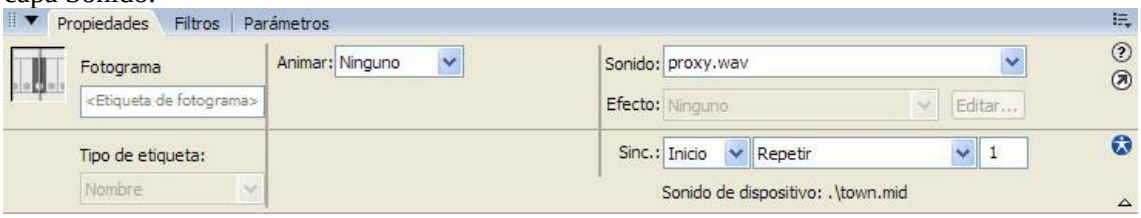
Insertamos este fichero dentro de la biblioteca de la aplicación, a través de la opción Archivo/Importar/Importar a biblioteca.
Una vez que tenemos el sonido de proxy insertado en la biblioteca, procederemos a asociar este sonido al sonido de dispositivo real que utilizaremos en nuestra aplicación. Tal y como hemos dicho anteriormente, este fichero a reproducir, será un fichero MIDI. Para realizar la asociación, abriremos la ventana de propiedades del elemento proxy.wav dentro de la ventana de la biblioteca de la aplicación y asociaremos el fichero MIDI a través del campo denominado Sonido de dispositivo, tal y como muestra la siguiente figura:



En nuestro caso, hemos utilizado el fichero town.mid. El lector, podrá seleccionar cualquier otro fichero MIDI que tenga accesible en su máquina. A partir de este momento, ya tenemos podremos utilizar el fichero proxy, para manejar el sonido dentro de la aplicación.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Para realizar la configuración final de la reproducción del sonido dentro de la aplicación, volveremos a la línea de tiempo. En el segundo fotograma de la capa denominada Sonido, asociaremos el sonido a esta capa, de tal forma que comience a reproducirse al entrar en este fotograma [39].
Debido a que estamos trabajando con un sonido de dispositivo, que tal y como hemos indicado, se reproducen de forma ininterrumpida hasta el final una vez que son activamos, tendremos que utilizar la orden stopAllSounds de ActionScript para detener la reproducción del sonido. Esta orden, está insertada en el botón de parar tal y como se mostró.
La siguiente imagen muestra cómo está asociada la reproducción del sonido con el fotograma 2 de la capa Sonido.



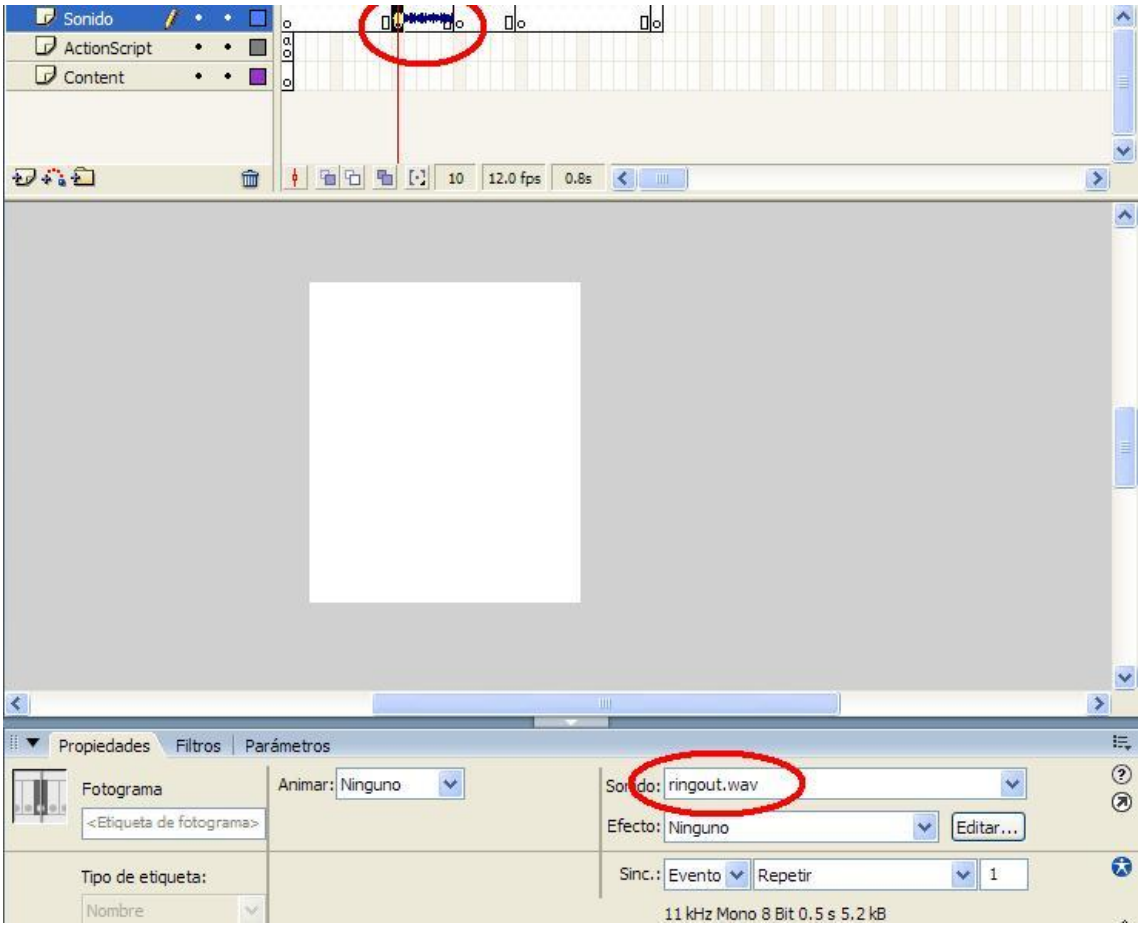
En la parte inferior derecha de la ventana de propiedades de la capa, tal y como se ve en la figura, tenemos el sonido (proxy.wav) y la acción.
Por último, y para completar el ejemplo, simplemente tendremos que ejecutar la aplicación seleccionando como dispositivo de ejecución un dispositivo que soporte el sonido. Las siguientes imágenes muestran la aplicación en ejecución.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Para utilizar sonidos nativos dentro de una aplicación, lo primero que deberemos hacer será importar el fichero en cuestión a la librería de la aplicación. En nuestro caso, hemos usado un fichero wav que insertaremos en una capa de la línea de tiempo, denominada Sonido, a través de la ventana de propiedades de la capa, tal y como hemos hecho en el ejemplo anterior. Una vez hecho esto, sencillamente con la gestión de los fotogramas de esta capa, tendremos la posibilidad de sincronizar el sonido con la an imación.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc



ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

ACTIONSCRIPT EN FLASH LITE 1.1

Aunque ya hemos hecho algún uso de actionscript dentro de Flash Lite, a continuación vamos a ver de forma más detallada cuáles son las capacidades del lenguaje dentro de esta versión de Flash y sus posibles usos.
La versión de actionscript disponible en la versión 1.1 de Flash Lite contiene ciertos elementos presentes en la versión del lenguaje actionscript que incorporaba Flash 4, junto con ciertos elementos específicos de Flash Lite, propios del tipo de dispositivos al que está orientada la plataforma: iniciar llamadas, mensajes de texto…
Hay algunos elementos correspondientes al lenguaje actionscript de la versión 4 de Flash, que no están presentes en la versión Lite:

- Las funciones startDrag y stopDrag.
- La propiedad _dropTarget.
- La propiedad _soundBufTime.
- La propiedad _url.
- La propiedad de conversión String().

Junto con estas restricciones, la gestión de pulsaciones de botones y teclas también difiere en cierta medida, pero ya hemos visto como trabajar con estos elementos [40].
Algunos otros puntos que conviene resaltar como ausentes en Flash Lite, debido a su presencia en las versiones para PC son:

- Funciones definidas por el usuario.
- Matrices, objetos y otros tipos de datos complejos.
- La carga en tiempo de ejecución de archivos externos de imagen y sonido.

Para especificar la ruta a un determinado clip o línea de tiempo, se utiliza la sintaxis de barras (/), tal y como se utiliza en la gestión de directorios en sistemas de ficheros. También se pueden utiliza dos puntos (..) y los elementos _levelN, _root y _parent. Con estos elementos, conseguiremos seleccionar aquel elemento de la línea de tiempo al que queremos dirigirnos. Por ejemplo:

```
tellTarget("/prueba/clip1") {
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
   _x = 100;

}
```

Este código cambiaría el valor de la propiedad _x del clip denominado clip1, que está contenido en el clip prueba, que a su vez está contenido en la línea de tiempo principal de la película.
Para utilizar variables dentro del actionscript, tendremos que utilizar una sintaxis similar a la descrita en el párrafo anterior, usando también los dos puntos (:). También se puede utilizar una notación basada en puntos. Es decir, la variable var1 del elemento clip1 del que hablábamos en el párrafo anterior, Tendríamos que poner cualquiera de estas dos líneas:

```
/prueba/clip1/:var1

_root.prueba.clip1.var1
```

Debido a la restricción en el uso de matrices de la que hablábamos, tendremos que gestionar este tipo de estructuras de datos de otra forma. Una emulación del mismo es lo que tenemos a continuación. Para hacerlo utilizaremos la función eval, que permite acceder a las variables, propiedades o clips de película por su nombre. Así, aunque con ciertas limitaciones sobre el uso de matrices estándar tal y como se hace en la versión para Pc de Flash, podremos resolver en cierta medida el problema utilizando un determinado formato para el nombre de las mismas. Tendremos que definir los datos como:

```
dato_0 = “valor de la posición 1”;

dato_1 = “valor de la posición 2”;

dato_2 = “valor de la posición 3”;

dato_3 = “valor de la posición 4”;
```

Como podemos ver, los nombre de las variables que componen lo que podríamos denominar la matriz virtual, tienen un parte del nombre común y al final tienen un número que podríamos identificar con el índice del dato dentro de la matriz. Para hacer referencia ahora a estos datos almacenados, tendríamos que usar un fragmento de código similar al siguiente, teniendo en cuenta que queremos acceder a la posición determinada por el valor de la variable pos, dentro de la matriz virtual:

```
eval (“dato_” add pos);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Para mostrar todo el contenido de la matriz en la ventana de salida de la aplicación, podríamos usar lo siguiente:

```
for (i = 0; i <4; i++) {

   trace (eval ("dato_" add i));

}
```

También podríamos crear variables nuevas, e ir así aumentando el tamaño de nuestra matriz virtual. Por ejemplo, si tenemos una variable tam_dato que contiene en tamaño de la matriz virtual en cada momento, podríamos añadir un nuevo elemento a la matriz con el siguiente fragmento de código:

```
eval (“dato_” add tam_dato) = “Valor nuevo”;

tam_dato++;
```

Con este sencillo truco, podemos utilizar matrices en nuestras aplicaciones de una forma sencilla y efectiva, a pesar de que Flash Lite no soporte este tipo de datos de forma nativa.
Otra restricción de la que hemos hablado antes y que nos encontramos en Flash Lite, es la utilización de funciones personalizadas. Esta restricción se puede también subsanar de una forma sencilla a través de la función call(). Esta función nos permite la ejecución del código insertado en un determinado fotograma, sin necesidad de desplazar la línea de tiempo hasta el mismo. A través de esta función se puede invocar un fotograma de la misma línea de tiempo, o un fotograma de cualquier clip de la aplicación. De esta forma, insertando código en ciertos fotogramas y utilizando la función call para invocar dicho código, podremos simular el uso de funciones personalizadas [41].

Elementos propios del actionscript de Flash Lite 1.1

A continuación vamos a ver aquellas características propias del lenguaje actionscript para la versión Lite de Flash. Comenzaremos por ver cómo se inicia una llamada telefónica, un mensaje de texto o un mensaje multimedia:

```
getURL(“tel:123456789”);

getURL(“sms:123456789”);

getURL(“sms:123456789?body=Texto del mensaje. Saludos”);

getURL(“mms:123456789”);
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Dos elementos importantes dentro del actionscript de Flash Lite y que ya hemos visto en algún momento a lo largo del documento son fscommand y fscommand2. El primero de ellos nos permite lanzar otras apliciones:

```
status = fscommand("launch",

  "d:\\system\\apps\\browser\\browser.app,http://www.usal.es");

}
```

Como vemos, el formato del segundo parámetro de la función, contiene en primer lugar el nombre de la aplicación a lanzar, seguido de los parámetros que desean pasarse a esta aplicación, separados por “,”.
La instrucción fscommand2 es similar a fscommand en su sintaxis. También devuelve un valor y recibe unos parámetros. Estos parámetros son un comando en primer lugar y separados por comas del comando y entre sí, la información adicional que este comando necesite. Los comandos que se le pueden pasar a fscommand2 son los siguientes:

- Escape – Codifica una cadena de texto para ser enviada a través de la red, sustituyendo los caracteres no alfanuméricos por su secuencia de escapa hexadecimal.

- FullScreen – Establece el tamaño de visualización de la aplicación, para que ocupe la pantalla completa o no (true o false). Sólo funciona cuando el reproductor de Flash Lite se ejecuta en modo autónomo, evidentemente, cuando el reproductor está embebido dentro de otra aplicación, no se puede aplicar.

- GetBatteryLevel – Esta llamada retorno el nivel de batería del que dispone el dispositivo en ese momento. El valor irá desde 0 hasta el valor máximo, que se puede averiguar a través de la llamada a fscommand2 con el comando GetMaxBatteryLevel.

- GetDateDay – Devuelve el día correspondiente a la fecha actual, es decir, puede tomar valores entre 1 y 31.

- GetDateMonth – Devuelve el mes correspondiente a la fecha actual, es decir, puede tomar valores entre 1 y 12.

- GetDateWeekday – Devuelve un valor numérico, correspondiente al día de la semana correspondiente a la fecha actual:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

  - 0 – Domingo
  - 1 – Lunes
  - 2 – Martes
  - 3 – Miércoles
  - 4 – Jueves
  - 5 – Viernes
  - 6 – Sábado
  - -1 – Error

- GetDateYear – Devuelve el año correspondiente a la fecha actual.

- GetDevice – Indica a través de una variable cuyo nombre es una cadena de texto que se pasa como parámetro, el nombre identificador del dispositivo que está ejecutando la aplicación. Un ejemplo de uso de este comando se presenta a continuación y la variable devicename, sería la que contendría el valor una vez realizada la llamada:

```
fscommand2("GetDevice", "devicename");
```

- GetDeviceId – Indica a través de una variable cuyo nombre es la cadena de texto que se pasa como parámetro, un identificador único del dispositivo en el que se está ejecutando la aplicación, como puede ser el número de serie.

- GetFreePlayerMemory – Devuelve la cantidad de memoria disponible para Flash Lite, expresada en kilobytes.

- GetLanguage – Indica a través de una variable cuyo nombre es la cadena de texto que se pasa como parámetro, el idioma utilizado por el dispositivo.

- GetLocaleLongDate – Indica a través de una variable cuyo nombre es la cadena de texto que se pasa como parámetro, la fecha actual, expresada en formato largo a través de una cadena de texto.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- GetLocaleShortDate – Indica a través de una variable cuyo nombre es la cadena de texto que se pasa como parámetro, la fecha actual, expresada en formato corto a través de una cadena de texto.

- GetLocaleTime – Indica a través de una variable cuyo nombre es la cadena de texto que se pasa como parámetro, la hora actual.

- GetMaxBatteryLevel – Retorna el nivel máximo de batería.

- GetMaxSignalLevel – Retorna el nivel máximo de intensidad de señal.

- GetMaxVolumeLevel – Retorna el nivel máximo de volumen.

- GetNetworkConnectStatus – Indica el estado actual de la conexión de red. Los posibles valores que puede retornar esta llamada son:

    - 0 – Hay una conexión activa.

    - 1 – El dispositivo está conectándose.

    - 2 – No hay conexión activa.

    - 3 – Se ha interrumpido la conexión de red.

    - 4 – No de puede determinar el estado de la conexión de red.

    - -1 – Comando no admitido por el terminal.

- GetNetworkName – Indica a través de una variable cuyo nombre es la cadena de texto que se pasa como parámetro, el nombre de la red.

- GetNetworkRequestStatus – Retorna el estado de la última petición http realizada. Los posibles valores son:

    - 0 – Hay una solicitud pendiente, se ha establecido una conexión de red, se ha resuelto el nombre del servidor y se ha realizado conectado con este.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

  - 1 – Hay una solicitud pendiente y se ha establecido una conexión de red.
  - 2 – Hay una solicitud pendiente, pero todavía no se ha establecido una conexión de red.
  - 3 – Hay una solicitud pendiente, se ha establecido una conexión de red y se está resolviendo el nombre de host del servidor.
  - 4 – La solicitud ha fallado debido a un error de la red.
  - 5 – La solicitud ha fallado debido a un error de conexión al servidor.
  - 6 – El servidor ha devuelto un error de HTTP (por ejemplo, 404).
  - 7 – La solicitud ha fallado debido a un error al acceder al servidor DNS o al resolver el nombre del servidor.
  - 8 – La solicitud se ha realizado correctamente.
  - 9 – La solicitud ha fallado debido a un error de tiempo límite agotado.
  - 10 – La solicitud aún no se ha realizado.
  - -1 – Comando no admitido.

- GetNetworkStatus – Devuelve un valor indicando el estado de la red telefónica. Los posibles valores son:
  - 0 – No hay ninguna red registrada.
  - 1 – Red doméstica.
  - 2 – Red doméstica ampliada.
  - 3 –Lejos de la red doméstica.
  - -1 – No se admite el comando.

- GetPlatform – Indica a través de una variable cuyo nombre es la cadena de texto que se pasa como parámetro, la descripción extendida del dispositivo.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- GetPowerSource – Indica el tipo de alimentación que se está usando. Los posibles valores son:
    - 0 – Batería.
    - 1 – Fuente externa de alimentación.
    - -1 – Comando no admitido.
- GetSignalLevel – Indica la intensidad de la señal a través de un valor numérico que va desde 0 hasta el valor determinado por el comando GetMaxSignalLevel.
- GetTimeHours – Indica las horas en el hora actual, y sus valores van desde 0 hasta 23.
- GetTimeMinutes – Indica los minutos en la hora actual, y sus valores van desde 0 hasta 59.
- GetTimeSeconds – Indica los segundos en la hora actual, y sus valores van desde 0 hasta 59.
- GetTimeZoneOffset – Indica a través de una variable cuyo nombre es la cadena de texto que se pasa como parámetro, el número de minutos de diferencia entre la hora local y la universal.
- GetTotalPlayerMemory – Indica la cantidad total de memoria, expresada en kilobytes.
- GetVolumeLevel – Indica el nivel actual de volumen. Este valor puede variar entre 0 y el valor indicado por GetMaxVolumeLevel.
- Quit – Este comando ordena la parada de la reproducción y cierra el reproductor. Sólo es aplicable cuando el reproductor funciona en modo autónomo.
- ResetSoftKeys – Este comando reestablece la configuración original de las flechas programables.
- SetInputTypeText – Este comando establece el modo en que debe abrirse el campo de introducción de texto, cuyo nombre recibe como segundo parámetro (el primero es el nombre del propio comando). El tercer parámetro indica el modo en cuestión y puede ser uno de los siguientes valores:
    - Numeric – 0-9.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

    - Alpha – a-z y A-Z.
    - AlphaNumeric – 0-9, a-z y A-Z.
    - Latin – Alfanuméricos y puntuación.
    - NonLatin – Sólo caracteres no latinos.
    - NoRestriction

- SetQuality – Establece la calidad de la presentación. Sus valores pueden ser:
    - high
    - medium
    - low

- SetSoftKeys – Permite modificar la asignación de las teclas programables. Este comando se explicó con detalle en un fragmento anterior del documento.

- StartVibrate – Ordena al dispositivo que comience a vibrar. Se puede indicar el tiempo (en milisegundos) que el terminal debe estar vibrando y las veces que debe repetir la vibración, así como el tiempo que debe parar entre repeticiones.

- StopVibrate – Ordena al dispositivo que detenga la vibración, si está activa.

- Unescape – Decodifica una cadena de texto codificada anteriormente.

A continuación, vamos a ver las capacidades y variables de la plataforma:

- _capCompoundSound – Indica si Flash Lite puede procesar ficheros de sonido compuesto. En caso afirmativo su valor será 1 y undefined en caso contrario.

- _capEmail – Indica si se pueden enviar mensajes de correo electrónico a través del método getURL. En caso afirmativo su valor será 1 y undefined en caso contrario.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- _capLoadData – Indica si se pueden cargar datos externos a través de las funciones loadMovie, loadMovieNum, loadVariables y loadVariablesNum. En caso afirmativo su valor será 1 y undefined en caso contrario.

- _capMFi – Indica si el dispositivo puede reproducir sonidos en formato MFi. En caso afirmativo su valor será 1 y undefined en caso contrario.

- _capMIDI - Indica si el dispositivo puede reproducir sonidos en formato MIDI. En caso afirmativo su valor será 1 y undefined en caso contrario.

- _capMMS - Indica si se pueden enviar mensajes multimedia a través del método getURL. En caso afirmativo su valor será 1 y undefined en caso contrario.

- _capMP3 - Indica si el dispositivo puede reproducir sonidos en formato MP3. En caso afirmativo su valor será 1 y undefined en caso contrario.

- _capSMAF - Indica si el dispositivo puede reproducir archivos multimedia en formato SMAF. En caso afirmativo su valor será 1 y undefined en caso contrario.

- _capSMS - Indica si se pueden enviar mensajes de texto (sms) a través del método getURL. En caso afirmativo su valor será 1 y undefined en caso contrario.

- _capStreamSound – Indica si el dispositivo puede reproducir flujos de sonido. (sincronizado). En caso afirmativo su valor será 1 y undefined en caso contrario.

- _cap4WayKeyAS – Indica si actionscript ejecuta código asociado a las teclas de flecha (derecha, izquierda, arriba y abajo). En caso afirmativo su valor será 1 y undefined en caso contrario.

- $version – Es un texto que indica el número de versión de Flash Lite.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

References

1. Lucas Fernando Souza De Castro, Gleifer Vaz Alves, André Pinz Borges (2017). Using trust degree for agents in order to assign spots in a Smart Parking. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
2. Rafhael Cunha, Cleo Billa, Diana Adamatti (2017). Development of a Graphical Tool to integrate the Prometheus AEOlus methodology and Jason Platform. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
3. Jaime Rincón, Jose Luis Poza, Juan Luis Posadas, Vicente Julián, Carlos Carrascosa (2016). Adding real data to detect emotions by means of smart resource artifacts in MAS. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 4
4. Christian Paulo Villavicencio, Silvia Schiaffino, J. Andrés Díaz-Pace, Ariel Monteserin (2016). A Group Recommendation System for Movies based on MAS. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 3
5. Asset Management System through the design of a Jadex Agent System (2016). Javier Carbó, José M. Molina, Miguel A. Patricio. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
6. Xiomara Patricia Blanco Valencia, M. A. Becerra, A. E. Castro Ospina, M. Ortega Adarme, D. Viveros Melo, D. H. Peluffo Ordóñez (2017). Kernel-based framework for spectral dimensionality reduction and clustering formulation: A theoretical study.
7. Juan Bullón, Angélica González Arrieta, Ascensión Hernández Encinas, Araceli Queiruga Dios (2017). Manufacturing processes in the textile industry. Expert Systems for fabrics production. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1
8. David Griol, Jose Manuel Molina (2016). Simulating heterogeneous user behaviors to interact with conversational interfaces. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 4
9. Casado-Vara, R., & Corchado, J. (2019). Distributed e-health wide-world accounting ledger via blockchain. Journal of Intelligent & Fuzzy Systems, 36(3), 2381-2386.
10. Gustavo Isaza, Maria H. Mejía, Luis Fernando Castillo, Adriana Morales, Nestor Duque (2012). Network Management using Multi-Agents System. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
11. Ana Cristina Bicharra, Nayat Sanchez-Pi, Luis Correia, José Manuel Molina (2012). Multi-agent simulations for emergency situations in an airport scenario. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
12. Valérian Guivarch, Valérie Camps, André Péninou (2012). AMADEUS: an adaptive multi-agent system to learn a user's recurring actions in ambient systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
13. Ichiro Satoh (2012). Bio-inspired Self-Adaptive Agents in Distributed Systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
14. Román, J. A., Rodríguez, S., & de da Prieta, F. (2016). Improving the distribution of services in MAS. Communications in Computer and Information Science (Vol. 616). https://doi.org/10.1007/978-3-319-39387-2_4
15. Buciarelli, E., Silvestri, M., & González, S. R. (2016). Decision Economics, In Commemoration of the Birth Centennial of Herbert A. Simon 1916-2016 (Nobel Prize in Economics 1978): Distributed Computing and Artificial Intelligence, 13th International Conference. Advances in Intelligent Systems and Computing (Vol. 475). Springer.
16. González-Briones, A., De La Prieta, F., Mohamad, M., Omatu, S., & Corchado, J. (2018). Multi-agent systems applications in energy optimization problems: A state-of-the-art review. Energies, 11(8), 1928.
17. Prieto, J., Alonso, A. A., de la Rosa, R., & Carrera, A. (2014). Adaptive Framework for Uncertainty Analysis in Electromagnetic Field Measurements. Radiation Protection Dosimetry, ncu260.
18. Chamoso, P., Raveane, W., Parra, V., & González, A. (2014). Uavs Applied to the Counting and Monitoring Of Animals. In Advances in Intelligent Systems and Computing (Vol. 291, pp. 71–80). https://doi.org/10.1007/978-3-319-07596-9_8

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

19. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. Information Sciences, 180(10), 2029–2043. https://doi.org/10.1016/j.ins.2009.12.032
20. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. International Journal of Ambient Computing and Intelligence, v 1, n 1(1), 15–26. https://doi.org/10.4018/jaci.2009010102
21. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. Expert Systems with Applications, 36(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
22. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. BMC Bioinformatics, 10. https://doi.org/10.1186/1471-2105-10-187
23. Fernández-Riverola, F., Díaz, F., & Corchado, J. M. (2007). Reducing the memory size of a Fuzzy case-based reasoning system applying rough set techniques. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 37(1), 138–146. https://doi.org/10.1109/TSMCC.2006.876058
24. Méndez, J. R., Fdez-Riverola, F., Díaz, F., Iglesias, E. L., & Corchado, J. M. (2006). A comparative performance study of feature selection methods for the anti-spam filtering domain. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4065 LNAI, 106–120.
25. Corchado, J. M., Pavón, J., Corchado, E. S., & Castillo, L. F. (2004). Development of CBR-BDI agents: A tourist guide application. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 3155, pp. 547–559). https://doi.org/10.1007/978-3-540-28631-8
26. Laza, R., Pavn, R., & Corchado, J. M. (2004). A reasoning model for CBR_BDI agents using an adaptable fuzzy inference system. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 3040, pp. 96–106). Springer, Berlin, Heidelberg.
27. Corchado, J. A., Aiken, J., Corchado, E. S., Lefevre, N., & Smyth, T. (2004). Quantifying the Ocean's CO2 budget with a CoHeL-IBR system. In Advances in Case-Based Reasoning, Proceedings (Vol. 3155, pp. 533–546).
28. Corchado, J. M., Borrajo, M. L., Pellicer, M. A., & Yáñez, J. C. (2004). Neuro-symbolic System for Business Internal Control. In Industrial Conference on Data Mining (pp. 1–10). https://doi.org/10.1007/978-3-540-30185-1_1
29. Corchado, J. M., Corchado, E. S., Aiken, J., Fyfe, C., Fernandez, F., & Gonzalez, M. (2003). Maximum likelihood hebbian learning based retrieval method for CBR systems. In Lecture Notes in Computer Science (including subseries LNAI and Lecture Notes in Bioinformatics) (Vol. 2689, pp. 107–121). https://doi.org/10.1007/3-540-45006-8_11
30. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
31. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
32. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
33. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
34. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. Int. J. of Robust and Nonlinear Control, 28(16), 5087-5102.
35. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
36. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
37. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
38. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
39. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
40. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Creación de portales y Servicios Web para B2C

David Palomar Delgado [1]

[1] University Carlos III – Calle Madrid, 126, 28903 Getafe, Madrid, Spain
dpalomar@inf.uc3m.es

Resumen: La funcionalidad inicial de los servidores web era mostrar información, esto se realizaba mediante la publicación de documentos html que mostraban la información en formato de texto que interpretaba y mostraba el navegador del cliente que se conectaba. Hoy en día el comercio web ha experimentado una evolución asombrosa y aquí se analiza tanto la evolución tecnológica experimentada como los diferentes modelos de comercio electrónico. El cálculo empresarial describe los resultados de una empresa en términos de gastos e ingre-sos, siendo el objetivo de ese cálculo examinar si la organización ha obtenido beneficios, en-tendidos como la diferencia positiva entre los ingresos y los gastos. Este proceso esencial en el funcionamiento de las empresas requiere idear y diseñar mecanismos por los cuales la empresa obtiene ingresos o activos. Sobre esos mecanismos se estructuran las actividades de la empresa. Algunos modelos de negocio son relativamente simples. Por ejemplo, una empresa puede dedicarse simplemente a la venta de un nuevo producto. Pero otras empresas tienen modelos de negocio que no son tan sencillos.

Palabras clave: web; B2C; portales web

Abstract. The initial functionality of a web site was to display information, this was done through the publication of html documents that showed the information in text format that interpreted and showed the browser of the client that connected. Today, web commerce has undergone an astonishing evolution and here we analyze both the technological evolution experienced and the different models of electronic commerce. The business calculation describes the results of a company in terms of expenses and income. The objective of the calculation is to examine whether the organization has made profits, understood as the positive difference between income and expenses. This essential process in the functioning of companies requires devising and designing mechanisms by which the company obtains income or assets. It is on these mechanisms that the company's activities are structured. Some business models are relatively simple. For example, a company can simply sell a new product. But other companies have business models that are not so simple.

Keywords: web; B2C; web sites

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

Arquitecturas web.

El auge de la tecnología web tiene su cimiento en una evolución de las arquitecturas fundamentada principalmente en el abaratamiento de costes de infraestructura, desarrollo y mantenimiento. Por una parte, el ahorro de dinero en comunicaciones vino en primera instancia gracias a la familia de protocolos TCP/IP, que permitió comunicar plataformas de una manera sencilla, barata y homogénea y sentó las bases para las arquitecturas actuales, por otra parte las mejoras en prestaciones de componentes hardware y la reducción de precio de los mismos han permitido la creación de infraestructuras mucho más baratas, pero de igual modo fiables y robustas.

Tradicionalmente las arquitecturas de comunicación de ordenadores han venido derivando en un modelo cliente-servidor, esto no quiere decir que las arquitecturas web no lo sean, sólo que existen algunos matices a tener en cuenta. En una arquitectura cliente-servidor tradicional el procesamiento de la información, así como su almacenamiento es monolítico, de manera centralizada se disponía de enormes servidores de información y almacenamiento de datos, toda la lógica de la aplicación, así como el procesamiento y gestión de los datos residía en una sola máquina, los clientes sólo mostraban los datos y todos se conectaban al servidor donde se realizaban las consultas y procesamiento de la información. Era un modelo denominado "cliente delgado" (thin client) y por el contrario de servidores gruesos (fat servers), es de suponer que cuando un servidor pudiera sufrir un colapso de procesamiento de información en base a las peticiones de múltiples clientes, la manera de resolverlo era aumentar su capacidad de hardware, o bien modificando sus componentes, o bien añadiendo más servidores para aumentar la capacidad de procesamiento, pero en ambos casos se produce un efecto "cuello de botella" ya que la escalabilidad de estas arquitecturas no está garantizada al realizar todo el procesamiento de la aplicación el mismo servidor, un ejemplo típico de estas arquitecturas son las aplicaciones basadas en sistemas legados como aplicaciones Cobol tan frecuentes en sistemas financieros como los bancos o grandes empresas.

Por otro lado nos encontramos con las arquitecturas Fat client, o cliente grueso, en este caso se libera al servidor de toda la carga de procesamiento y lógica de negocio y simplemente queda como un repositorio de datos, pero por el contrario es en los clientes donde reside toda la lógica de negocio, es decir todas aquellas reglas que se deben ejecutar en las aplicaciones para que estas funcionen, también reside en la misma aplicación cliente toda la exposición de datos al usuario, es decir, el interfaz, y también estas aplicaciones mantienen su propio gestor de datos en local, con este modelo se libera la carga sobre el servidor, y la cantidad de solicitudes que debe procesar, limitándose a la gestión de los datos de la aplicación, pero los problemas no se resuelven tan fácilmente, el cuello de botella sigue existiendo y la escalabilidad no mejora, debido a que al residir tanto la lógica de negocio como el interfaz de usuario en la aplicación cliente, esta resulta extremadamente acoplada impidiendo cualquier actualización de las misma. Imagina una situación en la que una empresa con 500 comerciales con sus respectivos ordenadores portátiles y tengan instalada una aplicación de gestión comercial sobre la que trabajan de manera local y cuando terminan sus gestiones se conectan con el servidor central al que mandan los datos actualizados, ahora imagina que es necesario realizar una modificación en el interfaz de usuario, como por ejemplo un cambio de color corporativo, sería necesario actualizar uno por uno todos los portátiles de la fuerza comercial de esa empresa, incluso si el cambio no es tan trivial como ese, como puede ser un cambio en la funcionalidad de la aplicación al ser dependiente del mismo interfaz de usuario habría posiblemente que modificar toda la aplicación y luego actualizar todos los puestos [1-7].

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Estos son algunos ejemplos que ocurren en la vida real, pero si piensas que posiblemente en base a los anteriores ejemplos el modelo Fat server parece más adecuado, míralo de esta forma, al realizar el servidor todo el procesamiento de las funcionalidades de la aplicación, junto con toda la gestión de datos de la misma y además provocar la salida de la información para que los clientes la puedan ver de manera adecuada, significa que todo esta altamente acoplado, y cualquier cambio en alguna de esas áreas provocará un cambio en el resto, es por esto por lo que tampoco resulta escalable este modelo, si intentas añadir simplemente un nuevo componente al interfaz de usuario posiblemente tengas que modificar parte de la funcionalidad de la aplicación.

Este tipo de arquitecturas tradicionalmente se apoyaban en metodologías de programación estructurada, y lenguajes definidos para tal fin, algunos bastante crípticos y rígidos y otros más orientados a la programación sobre interfaces visuales, dentro de esta gama tenemos lenguajes conocidos como Cobol, RPG, clipper, C, delphi o visual basic.

La aparición de la orientación a objetos supuso un aire fresco a la programación de componentes, la posibilidad de crear componentes como módulos reutilizables, permitió minimizar los tiempos de desarrollo al permitir la reutilización de funcionalidades, se redujo el acoplamiento entre los componentes de las aplicaciones permitiendo añadir nuevas funcionalidades con mínimo esfuerzo, y por tanto se mejoró la escalabilidad de las aplicaciones [8-12].

La aparición del protocolo http también supuso una revolución sobretodo al permitir compartir documentos en formato de hipertexto entre distintos ordenadores. El lenguaje HTML permitía la creación de interfaces gráficas de manera simple, residiendo la información y lógica de navegación en los mismos documentos y por tanto en el ordenador que los servía, los clientes mediante los navegadores web se convertían así en clientes delgados y que no permanecían acoplados a una aplicación concreta, cualquier cambio en el interfaz de usuario y en la lógica de la aplicación o incluso en el modelo de datos no impactaba directamente en el cliente. Los tiempos de desarrollo se veían disminuidos, los gastos en hardware y de mantenimiento también.

La evolución de las arquitecturas cliente-servidor tradicionales a arquitecturas web han sufrido muchos cambios, incluso las mismas arquitecturas web han ido evolucionando hacia estructuras más complejas dependiendo de las funcionalidades y requisitos de las aplicaciones.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Evolución de las arquitecturas web.

La funcionalidad inicial de los servidores web era mostrar información, esto se realizaba mediante la publicación de documentos html que mostraban la información en formato de texto que interpretaba y mostraba el navegador del cliente que se conectaba. La información mostrada era puramente estática, lo que se mostraba es lo que había y no podía sufrir modificaciones a menos que se modificara el documento original, tampoco había interactividad por parte del usuario con el interfaz, ya que mayormente se limitaba al uso de etiquetas que proporcionaba el lenguaje html y al ser un lenguaje para formateo de la información y limitado no permitía la creación de interfaces de usuarios ricos en contenido y atractivos.

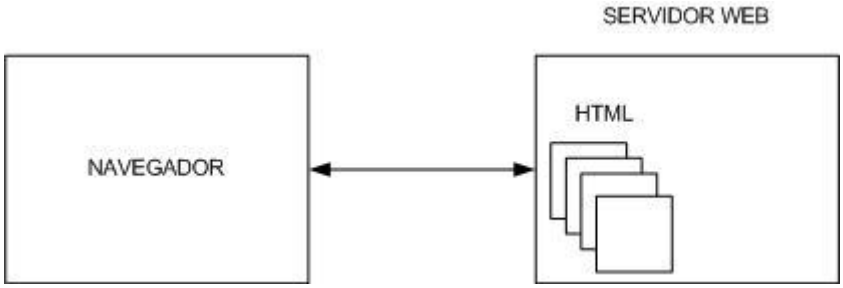


Figura 1. Arquitectura web simple

Una segunda evolución se realizó en torno al interfaz, aparecieron nuevas tecnologías aplicadas a mejorar el interfaz de usuario y la interactividad del usuario con respecto a la aplicación, esta interactividad se hacía a nivel de cliente, pero no de servidor, la aplicación seguía sin modificar su estado interno según las acciones del usuario, el único cambio que se realizaba era en apariencia.

Aparecieron tecnologías aplicadas a mejorar la exposición de los datos en el navegador, realizando presentaciones más elaboradas y vistosas, algunas de las tecnologías más conocidas son JavaScript, VBScript, ActiveX y Applets java, así como algunos componentes incrustados en el código que permitían mediante plugins instalados en el sistema cliente desplegar aplicaciones multimedia como video y sonidos, tal es el caso de shockwave o flash y quicktime por ejemplo.

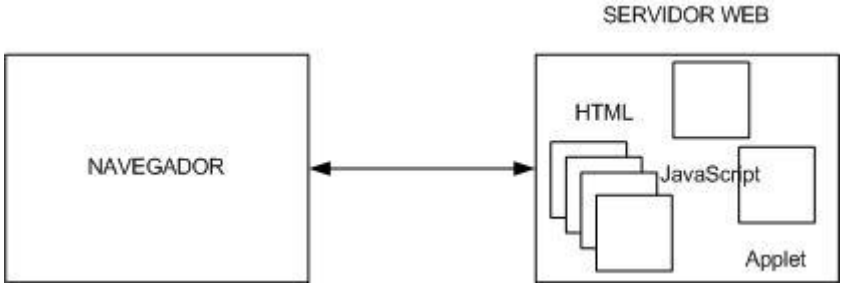


Figura 2. Mejora del interfaz.

Sin embargo la necesidad de recibir los datos por parte del usuario que ejecuta la aplicación cliente y en consecuencia mostrar información diferente dependiendo de las acciones que realice el usuario seguía estando presente. Surgen distintas tecnologías para recuperar esas acciones y permitir combinar el aspecto estático de la información proporcionada por los documentos html con información dinámica para ser presentada al usuario, de esta manera se consiguen aplicaciones más eficientes y atractivas sin desmejorar el aspecto visual de las mismas, pero que suscitan más interés a los usuarios

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

y proporcionan un nuevo sistema de crear aplicaciones que hasta la fecha sólo iban en un sentido. Surge la tecnología CGI (Common Gateway Interface) que usando las características y especificaciones del protocolo http permite establecer una comunicación e intercambio de datos entre distintas máquinas. Los lenguajes usados inicialmente para la creación de CGIs fueron C,C++ y Perl, posteriormente se han ido desarrollando más lenguajes específicamente orientados a la creación de aplicaciones web, cada uno usa una tecnología diferente, pero implícitamente la funcionalidad es la misma, hacer uso del protocolo http como sistema de comunicación y transporte.

El mundo Java en ese aspecto realizó un cambio tajante de dirección y orientó su tecnología a la parte servidora más que a la parte cliente, las propias arquitecturas web cada vez solicitaban más funcionalidades y prestaciones y era palpable que las características de orientación a objetos y multiplataforma del lenguaje Java podían realizar un mejor papel en los servidores que mediante la creación de pesados applets gráficos en los clientes. Se creó una infraestructura a través del jsdk 1.2 y posteriores, denominado a partir de aquel momento como java 2, que ampliaron en gran medida el API de desarrollo expuesto hasta entonces, se crearon multitud de librerías nuevas adaptadas a las necesidades del mercado que sirvieron para facilitar las tareas del desarrollador. La tarea de escribir CGIs fue enormemente facilitada y las aplicaciones se constituían de manera más rápida, sencilla de mantener y más robustas, pudiendo compartir datos de manera homogénea entre distintos repositorios de datos mediante el API JDBC, y posteriormente la posibilidad de gestionar las aplicaciones de manera distribuida mediante JNDI y tecnologías como RMI/IIOP y EJB.

La manera en que Sun Microsystems despliega su potencial en la creación de CGIs es mediante unos componentes denominados Servlets, los cuales mantienen algunas ventajas con respecto a sus antecesores y también heredan unos inconvenientes de aquel modelo de cgi. Una de las mayores ventajas era la seguridad de las aplicaciones y el rendimiento, ya que a diferencia de los CGIs tradicionales un servlet ejecuta un único proceso en el servidor y luego maneja el resto de peticiones mediante instancias del mismo servlet, pero por el contrario hereda los inconvenientes de los CGIs y es la mezcla de código java que gestiona la lógica de la aplicación con código java junto a código html, javascript, etc. que maneja la lógica de presentación, esto imposibilita en gran medida la escalabilidad de la aplicación, su mantenimiento y la hace menos reutilizable, para cualquier modificación sobre el interfaz gráfico (modificación de html) es necesario modificar los servlets y realizar una recompilación del código, en base a esto aparece la tecnología JSP (Java Server Pages), que permite establecer roles dentro de un proyecto, separando los componentes que se dedican a la lógica de negocio de la aplicación, de aquellos componentes que se dedican al formateo de esa información y posterior presentación al cliente. La tecnología JSP permite la inclusión de etiquetas específicas con código java dentro de documentos html, lo que posibilita separar funcionalidades dentro de una aplicación y reutilizar los componentes haciéndolos menos acoplados entre sí, esto favorece la escalabilidad, el mantenimiento y la reutilización. La tecnología JSP vino amparada por el éxito que Microsoft desarrolló con su tecnología ASP (Active Server Pages), con la salvedad de que JSP es más flexible y es multiplataforma [13-15].

Con la aparición de la tecnología JSP se proporciona una nueva infraestructura más flexible, en la cual se permite establecer la separación entre la lógica de la aplicación y el diseño del interfaz de usuario, de esta manera no es necesario recompilar los componentes de negocio por una modificación del interfaz. Pero siguiendo una infraestructura de tres capas y el modelo MVC, la siguiente evolución es proporcionar la separación al detalle. La lógica de negocio se apoya en una serie de componentes

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

reutilizables y lo más independiente del protocolo posible, estos componentes denominados JavaBeans, permiten reutilizar funcionalidades entre aplicaciones y a ser independientes del protocolo utilizado pueden ser adaptados a prácticamente cualquier arquitectura. En una arquitectura J2EE basada en web, tanto los JSP cómo los Servlets pueden hacer uso de JavaBeans. La aparición de los Enterprise JavaBeans, proporciona a la especificación J2EE la posibilidad de crear arquitecturas distribuidas, proporcionar servicios remotos con un sistema de seguridad y transacciones homogéneo, así como la posibilidad conectar con otros sistemas diferentes. Para proporcionar un uso común y estándar de este tipo de arquitecturas, aparece el denominado modelo de contenedor, este modelo dado por la especificación J2EE proporciona la infraestructura necesaria y reglas de comportamiento de los componentes dentro de este contenedor, de esta manera todas las aplicaciones que cumplan la especificación J2EE podrán desplegarse y ejecutarse de la misma manera y con el mismo comportamiento esperado en cualquier contenedor que cumpla dicha especificación [16-20].

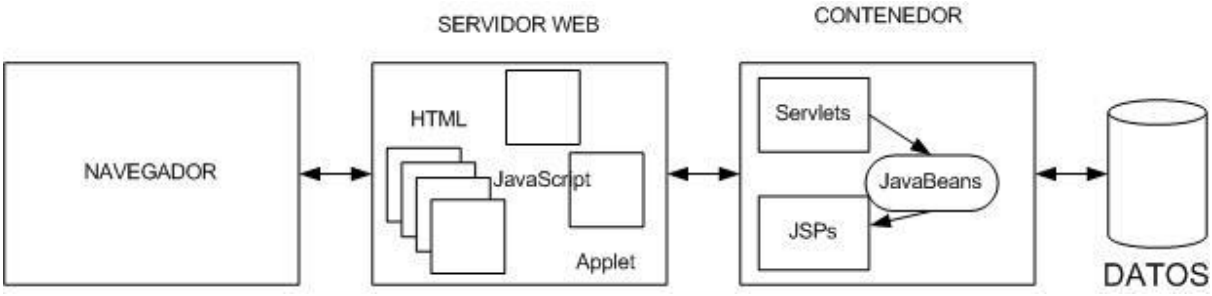


Figura 3. Separación al detalle.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Introducción al comercio electrónico

Se denomina comercio a la actividad socioeconómica consistente en el intercambio (la compra y venta) de bienes o servicios, sea para su uso, para su venta o para su transformación mediante otros procesos productivos. Esos intercambios tienen la característica de producirse libremente, debido a que cada una de las partes implicadas obtiene un valor subjetivo del intercambio, es decir, valora en más lo adquirido que lo entregado a cambio. Los intercambios comerciales implican la transferencia de bienes pero también la transferencia de documentos comerciales, tales como pedidos, albaranes o facturas. Desde esta perspectiva, el comercio electrónico (comercio-e o e-commerce) puede considerarse como el soporte de las transacciones comerciales mediante tecnologías de la información y las comunicaciones.

> El comercio electrónico es el intercambio comercial soportado por redes informáticas, incluyendo a las actividades asociadas de marketing llevadas a cabo mediante redes de ordenadores.

Categorías de comercio electrónico

Las transacciones del comercio-e pueden clasificarse de acuerdo a la naturaleza de los participantes en las mismas. Esto da lugar a tres categorías fundamentales:

- Business to Consumer (B2C), que implica la venta de bienes y servicios a usuarios finales. Todas las "tiendas virtuales" están incluidas en esta categoría. Un ejemplo típico de las mismas es la librería virtual Amazon (http://www.amazon.com/).

- Business to Business (B2B), que implica la venta de bienes y servicios entre empresas. Un ejemplo es Milacron (http://www.milacron.com/), proveedor a empresas de sistemas para el procesamiento de plástico.

- Consumer-to-Consumer (C2C), que implica la compra-venta entre usuarios finales. El ejemplo típico de este tipo de comercio electrónico son los sitios Web de subastas como eBay (http://www.ebay.es/).

Hay autores que citan otras categorías, por ejemplo, el B2E (Business to Employee) o el C2B (Consumer-to-Business).
El B2E incluye los intercambios comerciales de las empresas con sus propios empleados. Por ejemplo, una empresa turística puede ofrecer paquetes u ofertas especiales para empleados. Desde este punto de vista, el B2E es un tipo de B2C, pero también se utiliza el término B2E para hacer referencia al soporte mediante redes de otros procesos que no son estrictamente B2C, tales como el pago de gastos de desplazamiento, facturación de comisiones de ventas, etc.
Por otro lado, el C2B viene a dar nombre a situaciones concretas en las que un grupo de consumidores se reúne para hacer una petición a una o varias empresas de un producto o servicio determinado, habitualmente en condiciones de precio concretas. Si bien en este caso el inicio de la transacción está

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

en los consumidores, al final el intercambio es de tipo B2C, por lo que pueden considerarse una categoría especial dentro de B2C.
Otras categorías que se mencionan en ocasiones son las relaciones entre las administraciones del estado con las empresas, con los consumidores o con otras administraciones, que siguen las siglas A2B/C/A (Administration to Business/Consumer o Administration). Este tipo de relaciones podría incluir el cobro de impuestos o las gestiones de las solicitudes de servicios o subvenciones. No obstante, ese tipo de transacciones no son libres y no entran en la lógica del intercambio por el valor subjetivo de los bienes y servicios, por lo que no pueden llamarse propiamente "comercio".
Recientemente se han introducido otros criterios de categorización. Posiblemente el más conocido es el del tipo de tecnología utilizada, que ha surgido por la extensión del uso del teléfono móvil y otros dispositivos móviles para las transacciones comerciales (mobile commerce o m-commerce).

Comercio y negocio electrónico

El término "negocio electrónico" (negocio-e o e-business) tiene connotaciones muy similares al de comercio-e, lo cual requiere una clarificación de las diferencias.
El negocio (business) puede definirse como la gestión de los recursos humanos y materiales para organizar la productividad colectiva para mejorar la satisfacción de las necesidades de las personas. Este tipo de organización es típica de las economías libres, en las cuales las empresas (a las que también se hace referencia en ocasiones con el término "negocio") son el elemento organizador central. De esta definición se puede extraer que el comercio es una parte del negocio como organización general, si bien es la actividad imprescindible que permite dicha organización. Por tanto, el negocio-e incluye al comercio-e pero abarca otras áreas de la organización empresarial que no pueden clasificarse estrictamente como comercio.

> El negocio electrónico es cualquier actividad en el entorno empresarial que se realiza con el soporte de redes de ordenadores. Entre esas actividades se encuentran las actividades de comercio-e.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Breve reseña histórica

El soporte de transacciones comerciales mediante computadores y redes de ordenadores se introdujo a finales de los 70, con tecnologías como Electronic Data Interchange (EDI) and Electronic Funds Transfer (EFT). Estas tecnologías permitían ya el envío de documentos de carácter comercial como pedidos o facturas por medios electrónicos.

De EDI a XML

EDI son las siglas de "*Electronic Data Interchange*". EDIFACT (*Electronic Data Interchange for Administration, Transport and Commerce*) es un estándar de la ONU para el intercambio de documentos comerciales en el ámbito mundial.

Un mensaje típico EDIFACT tiene la forma siguiente (tomado del proyecto europeo ISIS):

```
UNH+1857+ORDERS:D:99A:UN:FI0084'
BGM+220+1999B2734:9'
DTM+137:19991105:102'
RFF+CT:652744'NAD+BY+5012345678900::9'
NAD+SU+6012345678900::9'
NAD+CA+7012345678900::9'
NAD+CZ+7012345678950::9'
NAD+CN+++THE VILLAGE STORE+2 THE REDDINGS:CHELTENHAM+GLOS++GL51 2UP'
LIN++1+37534656:EN'
IMD+F+8+:::SUPER PARTY POPPERS'
QTY+21:100'
DTM+2:1999121:102'
UNT+13+1857"
```

Cada segmento del mensaje comienza con una palabra de tres letras, y dentro del segmento hay componentes que comienzan por un signo de suma. El primer y último segmento (UNH, UNT) indican el comienzo y el final del mensaje. El segmento BGN indica el tipo y la función del mensaje, en este caso, 220 indica que el mensaje es un pedido. En el ejemplo, RFF indica una "referencia", IMD es una descripción de un artículo y QTY es la especificación de una cantidad. La interpretación completa de estos mensajes, que no trataremos aquí, está descrita en los documentos de Naciones Unidas que pueden consultarse en: http://www.unece.org/trade/untdid/directories.htm

Debido a la complejidad en la codificación EDIFACT, se crearon subconjuntos de reglas de codificación denominadas *Message Implementation Guideline* (MIG), que simplifican la estructura general para uso de cierto sector o ciertas aplicaciones concretas.

La popularización de Internet y la World-Wide-Web conllevó un auge de los denominados "lenguajes de marcado", de los cuales posiblemente el más conocido es el lenguaje HTML en el que están escritas la mayoría de las páginas Web. XML (*extensible markup language*) es un lenguaje de marcas pensado para transferir información de cualquier tipo. Obviamente, los documentos de las transacciones de comercio-e son un tipo concreto de información que puede estructurarse, como ya se hacía con EDIFACT. Esto ha llevado a que cada vez se use más XML como forma de codificación de los documentos de negocio. Existen diferentes formatos para la codificación de esos mensajes. Por ejemplo, el siguiente fragmento de XML es un ejemplo de codificación de un pedido.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<ProcessPurchaseOrder environment="Test" lang="en-US">
<ApplicationArea>...
<DataArea>
<Process acknowledge="Always"/>
<PurchaseOrder>
<Header>...</Header>
  <Line>...</Line>
  <Line>...</Line>
</PurchaseOrder>
</DataArea>
</ProcessPurchaseOrder>

Como puede verse en el ejemplo, cada parte del mensaje esta delimitados por marcas. Por ejemplo, cada línea del pedido estará entre las dos marcas <Line>...</Line>, y dentro de esas marcas aparecen otras con las cantidades, las referencias, etc.

```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

El comercio electrónico en España

Hábitos de consumo por Internet[1]

Se estima que el volumen del B2C en España en 2005 fue de más de 2100 millones de euros, lo que representa un incremento del 16,7% respecto al año anterior. El siguiente gráfico resume la evolución en los últimos años.

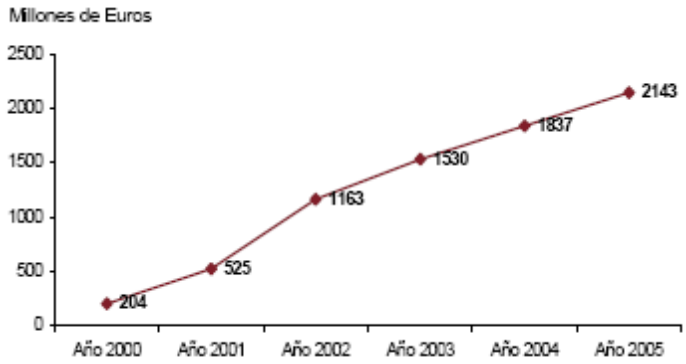


Figura 4. Hábitos de consumo

El perfil mayoritario actual es el de "hombres, de 25 años o más, con estudios medios o universitarios, de nivel socioeconómico medio alto y alto y con ingresos superiores a 1.200€ mensuales." A esto debe añadirse que la práctica totalidad de los compradores por Internet hace uso del correo electrónico.

Los billetes de avión/barco/tren/autobús (31.7%), seguidos de entradas para espectáculos (17.7%), libros (14.4%), electrodomésticos (12.2%), reservas de alojamientos (11.1%) y artículos de electrónica (10.6%) son los más comprados Internet durante 2005.

Respecto a Internet como canal diferenciado, las dos principales razones para iniciarse en el comercio electrónico son la comodidad y el precio o las promociones que pueden conseguirse. Secundariamente, los compradores valoran que en ocasiones Internet es la única vía para conseguir un determinado producto o servicio. También se valoran las facilidades que ofrece Internet para comparar ofertas y obtener información sobre productos o servicios [21-25].

Otras áreas del comercio-e

Un aspecto importante para concebir la estrategia digital de una empresa es entender los patrones de negocio electrónico que han emergido en el mercado y que caracterizan la economía digital en la actualidad. Esto permitirá tanto reconocer posibles esquemas de negocio de la competencia como identificar el patrón que mas se ajusta a las necesidades de la empresa. A continuación se resumen los aspectos particulares de cada patrón según se presenta en Kalakota (2002).

[1] *Fuente*: estudio compilatorio de los datos referentes al comercio electrónico en el sector minorista (B2C) de la Asociación Española de Comercio Electrónico y Marketing Relacional AECEM/FECEMD, junto a la Entidad Pública Empresarial Red.es, y de los hábitos de consumo a través de Internet en España en 2005.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- **e-Channel:** El patrón e-Channel, se refiere a las cadenas de relaciones entre compañías, clientes y socios de negocio, que participan cooperando con el fin de colocar un producto o servicio en el mercado. Este tipo de patrón puede emplearse para: -mejorar el aspecto transaccional del negocio mediante la presencia on-line (transacciones electrónicas); -innovar mediante la identificación de necesidades que puedan ser satisfechas con nuevos productos y servicios; -reducir la intermediación eliminando el numero de elementos en la cadena o pasos redundantes; o por el contrario, -alargar el numero de elementos de la cadena con miras a ofrecer productos o servicios complementarios.

- **Click-and-Brick:** El termino Click and Brick, (también Click and Mortar, o Brick and Click), se refiere tanto a compañías con presencia física (Brick and Mortar) que buscan transformar sus operaciones para soportar el modelo de negocios electrónico, como a compañías basadas en Internet (Pure-e) que reconocen la necesidad de contar con canales físicos además de los virtuales. Las razones para la presencia on-line han sido expuestas con anterioridad. Por otro lado, empresas que se iniciaron con presencia solamente en Internet, como Amazon, terminaron transformándose en modelos mixtos que incorporan tanto presencia física como on-line, al reconocer la necesidad de una presencia física (almacenes, en este caso) para servir mejor a sus mercados.

- **e-Portal:** Los portales electrónicos se comportan como intermediarios entre los clientes y proveedores, agregando algún tipo de valor a los servicios o productos. Son ejemplos de estos canales: -los superportales como Yahoo! que usan el atractivo de su contenido gratuito para dirigir el tráfico en Internet hacia otros sitios Web; -los portales de subastas, como eBay, que facilitan las transacciones entre compradores y vendedores para el intercambio de mercancías;- portales multi-transaccionales tales como Expedia, que mediante acuerdos con líneas aéreas y cadenas hoteleras, ofrecen paquetes de servicios que incluyen planes vacacionales, reservas de hotel, billetes de avión, alquiler de vehículo etc.

- **e-Market Maker:** Intermediario on-line que pone en contacto a compradores y vendedores dentro de una misma industria de mercado vertical. A través de este esquema, los compradores obtienen mejoras en los costos de compra a la vez que tienen más proveedores a su alcance. Los vendedores también bajan sus costos de venta y tienen más clientes a su alcance. Como ejemplo de este patrón se encuentran los sitios de intercambio, distribuidores virtuales (Chemdex), agregadores de catálogos, y Subastas.

- **Pure-e:** también denominado punto com o Dot com, se refiere a compañías que operan sin presencia física, solamente on-line. En determinados casos puede ser beneficioso operar como un negocio puramente electrónico, los costos de operación son generalmente más bajos (alquiler de local, mobiliario, servicios) y ofrecen la conveniencia al comprador de realizar

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

las transacciones desde la comodidad de su casa a un menor costo. Ejemplos de empresas Pure-e son los proveedores de servicio de Internet o Internet Service Providers (ISP), los motores de búsqueda como Google y Yahoo!, los operadores de bolsa on-line. Sin embargo este patrón puede ser no adecuado para todas los negocios, ya que cierto tipo de productos y servicios no pueden prestarse solamente on-line, tal es el caso de restaurantes, servicios de reparaciones del hogar, asistencia medica, aunque si pueden valerse de la presencia en este medio para promocionar sus servicios, proveer información adicional, y automatizar el manejo de citas [26-29].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Modelos de negocio en el comercio electrónico

El cálculo empresarial describe los resultados de una empresa en términos de gastos e ingresos, siendo el objetivo de ese cálculo examinar si la organización ha obtenido beneficios, entendidos como la diferencia positiva entre los ingresos y los gastos. Este proceso esencial en el funcionamiento de las empresas requiere idear y diseñar mecanismos por los cuales la empresa obtiene ingresos o activos. Sobre esos mecanismos se estructuran las actividades de la empresa. Algunos modelos de negocio son relativamente simples. Por ejemplo, una empresa puede dedicarse simplemente a la venta de un nuevo producto. Pero otras empresas tienen modelos de negocio que no son tan sencillos. En el sector audiovisual, diferentes medios, productores de contenidos y empresas de publicidad se relacionan, y de éstas relaciones aparecen modelos de negocio como el de la radio con emisión en abierto, que no obtiene sus ingresos directamente de los consumidores finales, entendidos estos como los radio-oyentes.

> Un modelo de negocio (también llamado diseño de negocio) es el mecanismo por el cual una empresa trata de generar ingresos y beneficios.

Una definición más completa, que enfatiza el carácter de diseño de los modelos de negocio es la siguiente, adaptada de (Osterwalder, Pigneur and Tucci, 2005):

> *[…] una herramienta conceptual que describe un conjunto de elementos y sus relaciones y que permite expresar la lógica del negocio de una empresa. Es una descripción del valor que la empresa ofrece a uno o varios segmentos de clientes, y la arquitectura de la empresa y su red de socios para la creación, marketing y entrega del valor y del capital de relación, de manera que se crean flujos de beneficios sostenibles y rentables.*

De la anterior definición se desprende que el modelo de negocio tiene que describir ambas partes del intercambio, el valor ofrecido, y cómo lo recibido por ese valor resulta en beneficios. Además, se incluye a los "socios" (p.ej. proveedores u otras empresas), por lo cual los modelos de negocio suelen hacer referencia a la posición de la empresa en una "cadena de valor" formada por varias empresas relacionadas. También se ha incluido en la definición el "capital de relación", es decir, la base de clientes de la empresa que han sido clientes y se espera vuelvan a serlo en el futuro, lo cual implica una relación sostenida en el tiempo.

38.Algunos ejemplos de cómo afrontar el modelo de negocio en la Web

39.

40.El artículo titulado "*Qué es Web 2.0. Patrones del diseño y modelos del negocio para la siguiente generación del software*" aparecido en el Boletín de la Sociedad de la Información de Telefónica el 23/02/06 proporciona comparaciones interesantes de la arquitectura general de modelos de negocio en la Web. A continuación describimos algunos de sus ejemplos.

41.

42.Definiendo el segmento de clientes: DoubleClick versus AdSense

43.

44.El mercado potencial de la Web no está limitado geográficamente, por lo que la segmentación de los clientes debe pensarse en cuanto a otras características. Una de las más importantes es el tamaño de la empresa – y sus

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

capacidades en cuanto a implantación de nuevas tecnologías. El siguiente fragmento de texto describe dos estrategias diferenciadas en este sentido.

45.

46. "*[…]finalmente DoubleClick se vio limitado por su modelo de negocio. Apoyó en los años 90 el concepto de que la web trataba de publicación, no participación; que los publicistas, no los consumidores, deben ser los que deciden; que el tamaño importaba, y que Internet cada vez estaba más dominada por los sitios web situados en la cima según las estadísticas de MediaMetrix y otras compañías que valoraban los anuncios de la web. Como consecuencia, DoubleClick cita orgulloso en su web 'más de 2000 implementaciones exitosas' de su software. ¡Yahoo! Search Marketing (antes Overture) y Google AdSense, por el contrario, ya dan cada uno servicio a centenares de millares de publicistas. El éxito de Overture y de Google fue fruto de la comprensión de lo que Chris Anderson cita como ' the long tail ' (literalmente 'la larga cola'), el poder colectivo de los sitios web pequeños que conforman la gran mayoría del contenido de la web. Las ofertas de DoubleClick requieren un contrato formal de venta, limitando su mercado a unos pocos miles de sitios web grandes. Overture y Google se las ingeniaron para permitir la colocación del anuncio prácticamente en cualquier página web. Lo que es más, evitaron los formatos de publicidad preferidos por los publicistas y las agencias de publicidad como banners y popups (ventanas emergentes), en favor de los anuncios de texto, mínimamente intrusivos, sensibles al contexto y amigables para el consumidor."*

47.

48. ***Definiendo el producto: Netscape versus Google***

49.

50. *"Netscape ideó el concepto de 'la web como plataforma' en términos del viejo paradigma del software: su buque insignia era el navegador web, una aplicación de escritorio, y su estrategia era utilizar su dominio en el mercado de los navegadores para crear un mercado de productos de servidor de gama alta. El control sobre los estándares para visualizar el contenido y las aplicaciones en el navegador, en teoría, dio a Netscape la clase de poder de mercado del que disfrutó Microsoft en el mercado de los PCs. […] Sin embargo, al final, los navegadores web y los servidores web resultaron ser commodities, y el valor se desplazó hacia los servicios ofrecidos sobre la plataforma web. Google, por el contrario, comenzó su vida como una aplicación web nativa, nunca vendida o empaquetada, sino siempre entregada como un servicio, con clientes pagando, directamente o indirectamente, por el uso de ese servicio. […] No hay programación de las actualizaciones de las versiones del software, sencillamente mejora continua. Ninguna licencia o venta, sencillamente uso. Ningún tipo de portabilidad a diferentes plataformas de forma que los clientes puedan ejecutar el software en su propio equipo, sencillamente, una colección masiva de PCs escalables en los que corren sistemas operativos de software abierto junto con aplicaciones y utilidades de su propia cosecha que nunca nadie de fuera de la compañía consigue ver.*

51. *En el fondo, Google requiere una capacidad que Netscape nunca necesitó: gestión de la base de datos. Google no es sencillamente una colección de herramientas software, es una base de datos especializada. […] El licenciamiento del software y el control sobre las APIs (la palanca de poder en la era anterior) es irrelevante porque el software no necesita ser distribuido sino ejecutado, y también porque sin la capacidad de recoger y de gestionar los datos, el software es de poca utilidad. […] El servicio de Google no es un servidor (aunque es ofrecido por una colección masiva de servidores de Internet) ni un navegador (aunque es experimentado por el usuario a través del navegador). Ni siquiera su servicio insignia, el de búsqueda, almacena el contenido que permite encontrar a los usuarios. Como una llamada telefónica, que no tiene lugar en los teléfonos de los extremos de la llamada sino en la red que hay entre medias, Google tiene lugar en el espacio que se encuentra entre el navegador y el motor de búsqueda y el servidor de contenido destino, como un habilitador o intermediario entre el usuario y su experiencia online. "*

52.

Muchas empresas tienen modelos de negocios similares o prácticamente iguales. Por eso puede hablarse de tipos de modelos de negocio, que permiten clasificar los mismos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Principales tipos de modelos de negocio en el comercio-e

Existen muchas clasificaciones de modelos de negocios para el comercio-e. Aquí se discute la clasificación del profesor Michael Rappa2, que tiene la ventaja de ser una abstracción más general de otras clasificaciones. Los modelos que se describen a continuación se hacen reales en negocios concretos de diversas formas, y es habitual que una misma empresa combine más de uno de esos modelos [30-32].

Modelos de intercambio (*brokerage models*)

Los brokers son negocios que crean mercados mediante el soporte a las transacciones y la interacción entre compradores y vendedores, normalmente en una determinada área. Estos brokers pueden servir en modalidades B2C, B2B o C2C. El modelo de negocio habitual es el cargar una comisión por cada transacción facilitada. Estos modelos incluyen a los siguientes.

Tipo	***Descripción***
Mercado de intercambio (*marketplace exchange*)	Ofrece un amplio rango de servicios, cubriendo la gestión de la transacción, proceso de negociación y compra. Operan de manera independiente o avalados por una industria en concreto. (aquanima, adquira).
Compra venta	Los clientes realizan pedidos para comprar o vender productos. (cashdirect, respond.com)
Sistemas de colección de demandas	Los compradores finales realizan pujas por un bien o servicio y el broker realiza el acuerdo.(Priceline.com)
Broker de subastas	El broker carga una tasa al vendedor que escala en función del precio de la transacción. Las subastas disponen de un modelo de reglas bastante amplio (eBay).
Broker de transacciones	Proporciona un mecanismo de pago a terceros para los compradores y vendedores en una transacción (PayPal, Escrow.com)
Distribuidores	Operaciones de catálogos que conectan con grandes cantidades de productos manufacturados tanto al por mayor como al detalle. Facilitan las transacciones entre distribuidores franquiciados y sus partners.
Agentes de búsqueda	Agentes de búsqueda o robots usados para buscar el mejor precio de un producto o servicio.
Mercado virtual	Servicio de hosting para comerciantes online que puede cargarles la configuración, el catálogo mensual o comisiones por transacción. Proporciona servicios de marketing (Amazon.com)

Modelo de publicidad (advertising model)

Es una extensión del modelo tradicional. El canal de anuncios o escaparate, en este caso un sitio web, proporciona contenido (normalmente gratis, aunque no necesariamente) y servicios (como email, mensajería, blogs) mezclado con mensajes publicitarios en forma de banners. El escaparate puede ser un creador de contenidos o

[2] http://www2.digitalenterprise.org/mrappa.html

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

un distribuidor de contenidos creado en cualquier sitio. El modelo de publicidad trabaja mejor cuando el tráfico de visitas es elevado.

Tipo	***Descripción***
Portal	Normalmente un motor de búsqueda que incluye varios contenidos o servicios (Yahoo!)
Clasificados	Listas de ítems para vender o comprar (Monster.com, Craigslist, Match.com)
Registro de usuarios	Contenido de sitios de acceso gratuito, pero requiere que los usuarios se registren y proporcionen información demográfica (NY times digital)
Posicionamiento basado en pago	Vende enlaces patrocinados o publicidad basada en términos clave de búsquedas (Google, Yahoo! marketing search)
Publicidad contextual	Desarrollos gratuitos que incluyen adaware wn sus productos. Puede usarse la publicidad basándose en la actividad de navegación del usuario (barras de navegadores, software de descarga. Claria)
Anuncios basados en el contenido	El pionero fue Google que identifica el contenidos de una página web y proporciona automáticamente anuncios cuando un usuario la visita.
Intromercials	Página a tamaño completo situada a la entrada de un sitio web.
Ultramercials	Anuncios interactivos online que requieren que el usuario responda intermitentemente a lo largo de un mensaje hasta alcanzar el contenido.

Modelo infomediario (Infomediary model)

Los datos de los clientes y sus hábitos de consumo con valorados especialmente cuando la información es cuidadosamente analizada y usada para realizar campañas de marketing. Independientemente de los datos recolectados sobre productores y sus productos, son útiles para ellos los consumidores que pueden ser considerados como objetivos de compra.

Tipo	***Descripción***
Redes de anuncios	Los banners se publicitan en una red de sitios. Los datos son recogidos sobre los usuarios web y pueden ser usados para comprobar la efectividad de una campaña de marketing (DoubleClick)
Servicios de medición de audiencia	Agencias de búsqueda de audiencias en mercados online (Netratings)
Marketing incentivo	Programas de lealtad de clientes que proporcionan incentivos a los mismos mediante puntos o cupones asociados a los comerciantes. Los datos recogidos de los clientes también son vendidos como objetivos de publicidad (Coolsaving)

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Metamediario (metamediary)	Facilita las transacciones entre comprador y vendedor facilitando información y servicios comprensibles sin estar involucrado en los procesos de adquisición de bienes o servicios entre ambas partes (Edmuns)

Modelo comerciante (Merchant model)

Comerciantes y minoristas de bienes y servicios. Las ventas pueden estar basadas en listas de precios o en subastas.

Tipo	***Descripción***
Mercader virtual	Comerciante minorista que opera solamente en la web (Amazon)
Comerciante de catálogo	Negocio web que combina, teléfono, correo electrónico y pedidos a través de la web (Land's End)
Click and mortar	Establecimiento de ventas tradicional con un interfaz web (Barnes & Noble)
Vendedor digital (Bit Vendor)	Comerciante que trata exclusivamente con productos y servicios digitales de la forma más pura y conduce las ventas y servicios a través de la web (Apple atunes music store)

Modelo directo

El modelo fabricante o directo se basa en el predicado de que la web permite a un fabricante alcanzar compradores directamente. Este modelo puede basarse en la eficiencia, mejorar el servicio con el cliente y entender mejor las necesidades del mismo.

Tipo	***Descripción***
Compra	la venta de un producto en el cual los derechos de la propiedad se transfieren al comprador
Leasing	A cambio de un alquiler, el comprador recibe el derecho de utilizar el producto según los términos de un acuerdo. El producto vuelve al vendedor sobre la expiración o el defecto del acuerdo de alquiler. El acuerdo puede incluir un derecho de compra sobre la expiración del contrato.
Licencia	La venta de un producto implica solamente la transferencia de los derechos de uso al comprador. De acuerdo con los "términos del acuerdo de uso". El fabricante sigue teniendo los derechos (e.g., con las licencias de software).
Contenido integrado en la marca	En contraste con los sponsors de marcas (modelo publicitario), el contenido integrado en la marca es creado por el propio fabricante con el único objetivo de situar un producto (bmwfilms)

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Modelo de afiliados

En contraste con el portal genérico, que intenta conducir un alto volumen de tráfico a un sitio, el modelo de afiliados proporciona oportunidades de compra dondequiera que la gente pueda navegar por la red. Esto se realiza ofreciendo incentivos financieros (bajo la forma de porcentaje de ventas) a los sitios afiliados del socio. Los afiliados proporcionan puntos de compra y tasas de clics al comerciante. Es un modelo de pago por rendimiento -- si un afiliado no genera ventas, no representa ningún coste al comerciante. El modelo del afiliado está intrínsecamente bien adaptado a la web. Las variaciones incluyen, intercambio de banners, pago por clic, y programas de compartir ventas.

Tipo	***Descripción***
Intercambio de banners	Comercio de banners a través de una red de sitios afiliados
Pago por clic	El sitio paga a sus afiliados en función del clic-through
Compartir ventas	Se ofrece un porcentaje de comisión por venta basado en el clic-through del usuario el cual debe comprar el producto

Modelo de comunidad

La viabilidad del modelo comunitario está basado en la lealtad del usuario. Los usuarios realizan una inversión alta en tiempo y emoción. Las ganacias pueden estar basadas en la venta de productos y servicios o contribuciones voluntarias o bien puede estar sustentada en publicidad contextual y subscripciones a servicios Premium. Internet está centrado completamente a los modelos de negocio comunitarios y actualmente es una de las áreas más fértiles en desarrollo, tal y como se puede ver en las redes sociales [33-35].

Tipo	***Descripción***
Open source (código abierto)	Software desarrollado de manera colaborativa por una comunidad de desarrolladores que comparten su código de manera abierta. En lugar de cobrar por el código, las ganancias se generan en los servicios prestados, como integración de sistemas, soporte, tutoriales, cursos y documentación. (Redhat)
Open content (contenido abierto)	Contenido desarrollado de manera colaborativa y accesible de forma abierta, desarrollado por una comunidad de voluntarios (Wikipedia)
Difusión pública	Difusión de radio y televisión en la web mantenidos mediante donaciones voluntarias (The classical station, wcpe.org)
Servicios de redes sociales	Sitios que proporcionan a unos individuos conectarse con otros con intereses comunes. Pueden proporcionar servicios de publicidad contextual y servicios premiun (flicker, orkut)

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Modelo de suscripción

A los usuarios se les carga una tasa periódica por suscribirse a un servicio. Es común encontrar sitios que combinan información gratuita con información sólo para miembros de pago. Los modelos de suscripción y publicidad son usados conjuntamente.

Tipo	***Descripción***
Servicios de contenido	Proporcionar vídeo, texto o audio a los usuarios que pagan la tasa para tener acceso a tal servicio (listen.com, netflix)
Servicios de redes persona-persona	Se basan en la distribución de información de usuarios, como por ejemplo la búsqueda de compañeros de colegio.
Servicios de confianza	Viene de la forma de asociaciones de miembros que adoptan un código de conducta y pagan una suscripción (www.truste.org)
Proveedores de servicios de Internet	Ofrecen conectividad y servicios a cambio de una suscripción mensual.

Modelo de utilidad

El modelo de utilidad o "bajo demanda" está basado en la aproximación "paga cuando quieras". A diferencia de los modelos de suscripción, estos modelos se basan en ratios de uso. Tradicionalmente se han usado en servicios domésticos (agua, luz, teléfono). Algunos ISPs, también usan este modelo cargando al usuario por tiempo de conexión.

Tipo	***Descripción***
Uso medido	Mide y factura al usuario basándose en el consumo actual del mismo.
Suscripciones medidas	Permite a los suscriptores comprar acceso al contenido en porciones, eg., número de páginas vistas (slashdot, baquia).

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Servicios Web

Se han dado muchas definiciones del concepto de Servicio Web, pero la definición "oficial" del consorcio W3C es la siguiente:

> "Un Servicio Web es un sistema software identificado por una URI, cuyas interfaces públicas y su forma se describen utilizando XML. Su definición puede ser descubierta por otros sistemas software, y esos sistemas pueden interactuar con el Servicio Web de la manera prescrita por su definición, utilizando mensajes basados en XML enviados mediante protocolos de Internet"

Por tanto, el concepto de Servicio Web es genérico, y las implementaciones comunes actualmente de los Servicios Web (sobre SOAP, WSDL y UDDI) son solamente una de las posibles implementaciones, pudiendo aparecer otras tecnologías de Servicios Web en el futuro. En cualquier caso, lo importante de los servicios Web es que proporcionan un mecanismo de **interoperabilidad** abierto, soportado por protocolos comunes de Internet, por lo cual, es más práctico que otras tecnologías precedentes de interoperabilidad, tales como CORBA, COM o las llamadas a procedimiento remoto (RPC) en general. Es importante tener en cuenta que además del W3C, existe otro consorcio denominado WS-I que trabaja sobre lo especificado por el W3C para hacer perfiles de compatibilidad de servicios Web más precisos. El *perfil básico* que ya proporciona WS-I define con un importante nivel de detalle las comunicaciones SOAP, sirviendo como base de verificación y prueba para las organizaciones interesadas en proporcionar Servicios Web con un alto grado de estandarización [36-40].

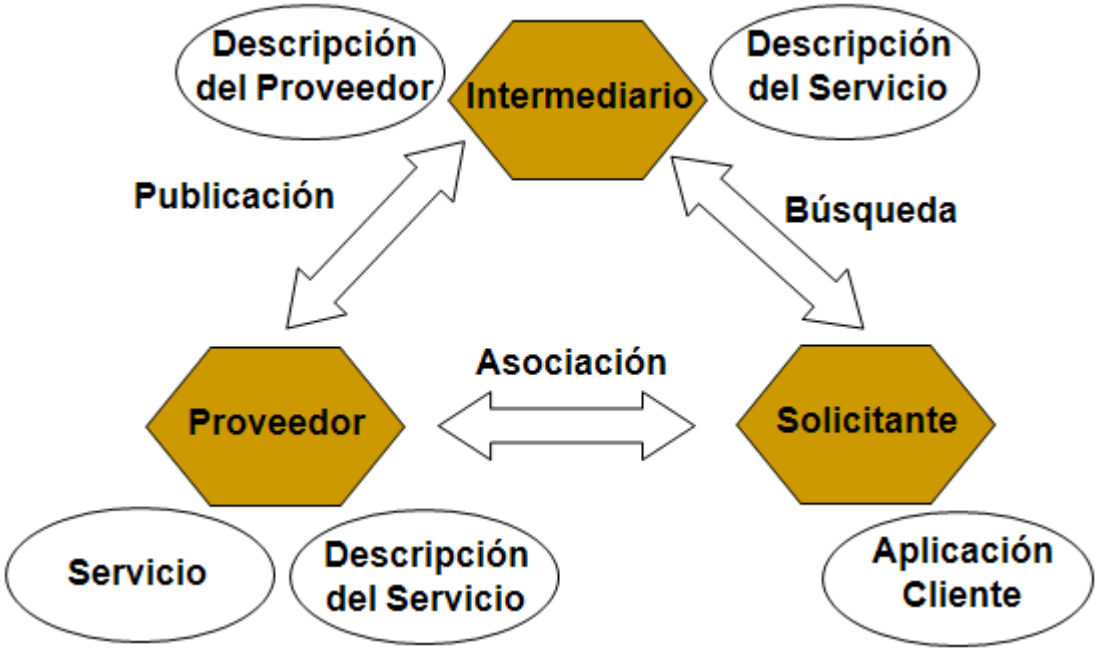


Figura 5Arquitectura global de los servicios web

La siguiente figura nos proporciona una visión general de la arquitectura de los Servicios Web. En ella vemos como sobre la base de los protocolos de Internet, y de XML como formato de representación de datos, se proporciona una capa de gestión de mensajes, que actualmente está ocupada por el protocolo SOAP y sus posibles extensiones. La capa de descripción de Servicios Web actualmente está soportada por el formato de especificación WSDL (también en XML), y sobre la mensajería y

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

descripción se pueden implementar procesos más complejos. Entre ellos, tenemos los sistemas de búsqueda y descubrimiento de Servicios Web, que pueden expresarse gracias a la arquitectura de repositorios UDDI. Por último, otros servicios transversales, como la seguridad o la gestión y auditoría de los mensajes pueden integrarse dentro de la arquitectura.

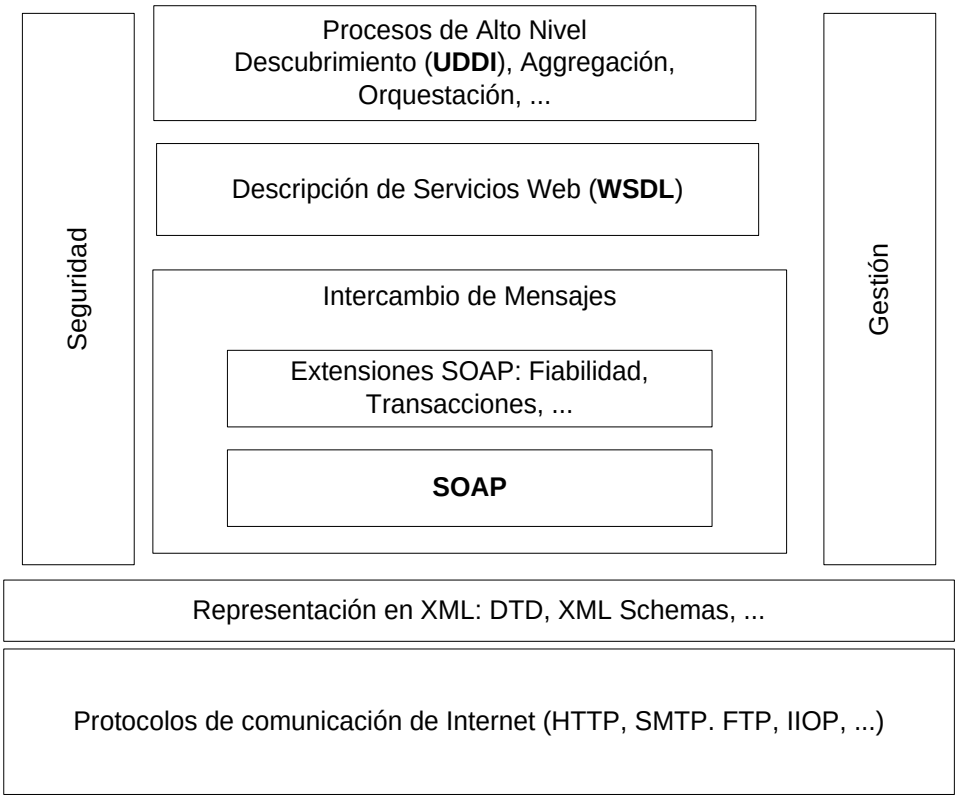


Figura 6. Arquitectura física de los servicios web

Por tanto, la definición de los mensajes y su intercambio actualmente se realizan mediante el protocolo SOAP, la descripción de los servicios mediante la especificación WSDL, y el descubrimiento de los servicios se puede llevar a cabo mediante UDDI. En el resto de esta sección describiremos los fundamentos básicos de estas tres tecnologías.
Por otro lado, tenemos a nuestra disposición bibliotecas para programar servicios Web en diferentes lenguajes, incluyendo la plataforma Java (Chappel and Jewell, 2002), y también el entorno de desarrollo .NET de Microsoft.

La arquitectura básica de SOAP

El protocolo *Simple Object Access Protocol* (SOAP) no es otra cosa que un protocolo basado en intercambio de mensajes XML. Concretamente, un mensaje SOAP es un documento XML normal, que incluye lo siguiente:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

1. Un elemento **Envelope** obligatorio que identifica el documento XML como mensaje SOAP.
2. Un elemento opcional **Header** con información sobre el mensaje.
3. Un elemento obligatorio **Body** que contiene la información de envío o respuesta.
4. Un elemento opcional **Fault** que proporciona información sobre los errores, en caso de que estos ocurran durante el procesamiento del mensaje.

Por tanto, la estructura general de un mensaje SOAP es la siguiente:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope" soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header> ... ...</soap:Header>
  <soap:Body> ...
    <soap:Fault> ... </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

La cabecera SOAP puede contener información específica de la aplicación, por ejemplo, sobre autentificación o pagos. Son atributos importantes los siguientes:

1. **Actor**: indica el receptor esperado del mensaje. Pueden utilizarse URIs estándar. Por ejemplo, podemos especificar http://schemas.xmlsoap.org/soap/actor/next para indicar que simplemente pase al siguiente nodo en un intercambio.
2. **mustUnderstand** indica que el receptor debe soportar el elemento.
3. Algunas especificaciones amplían este conjunto de atributos. Por ejemplo, ebXML define <from> y <to>.

Un ejemplo de cabecera con elementos propietarios de una organización determinada (en el espacio de nombres m) puede ser el siguiente:

```
<soap:Header>
 <m:MyInfo xmlns:m="http://www.catenon.com/transaction/"
   soap:actor="http://www.catenon.com/marketing/” soap:mustUnderstand="1">
    234
</m:MyInfo>
</soap:Header>
```

En el cuerpo de un mensaje SOAP se debe incluir la información que desea transmitirse. El siguiente es un ejemplo de cómo podría enviarse una solicitud de un precio en un cuerpo SOAP:

```
<soap:Body>
 <m:GetPrice xmlns:m="http://www.frutas.com/prices">
     <m:Item>Apples</m:Item>
 </m:GetPrice>
 </soap:Body>
```

Y el siguiente podría ser un ejemplo de la respuesta a la solicitud anterior:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<soap:Body>
<m:GetPriceResponse xmlns:m="http://www.frutas.com/prices">
   <m:Price>1.90</m:Price>
</m:GetPriceResponse>
</soap:Body>
```

Los mensajes SOAP como los que hemos visto deben enviarse a través de la red mediante un protocolo de transporte determinado, como puede ser HTTP o SMTP. El siguiente es un ejemplo de envío mediante HTTP:

```
POST /test/simple.asmx HTTP/1.1
Host: 131.107.72.13
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://soapinterop.org/echoString"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="http://soapinterop.org/"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <tns:echoString>
      <inputString>string</inputString>
    </tns:echoString>
  </soap:Body>
</soap:Envelope>
```

En el anterior ejemplo podemos ver la cabecera HTTP utilizando el método POST, y un campo de la cabecera específico para SOAP, que se denomina SOAPAction, y sirve para que el sistema receptor pueda identificar el Servicio Web al que va dirigida la solicitud.

Descripción de Servicios Web en WSDL

El lenguaje de descripción WSDL (*Web Service Description Language*) permite especificar el funcionamiento de un servicio Web en términos de una colección de puntos finales de acceso (*access endpoints*). WSDL describe cuatro tipos de datos esenciales sobre un Servicio Web:

1. Información sobre las interfaces, describiendo todas las funciones que están disponibles públicamente.
2. Declaraciones de tipos de datos necesarias para los parámetros de solicitud y respuesta.
3. Información sobre dónde los puntos de acceso al servicio según el protocolo de transporte.
4. Información sobre la dirección dónde puede encontrarse el servicio.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El siguiente esquema describe las informaciones fundamentales en WSDL, junto con su análogo en la tecnología de componentes orientados a objetos:

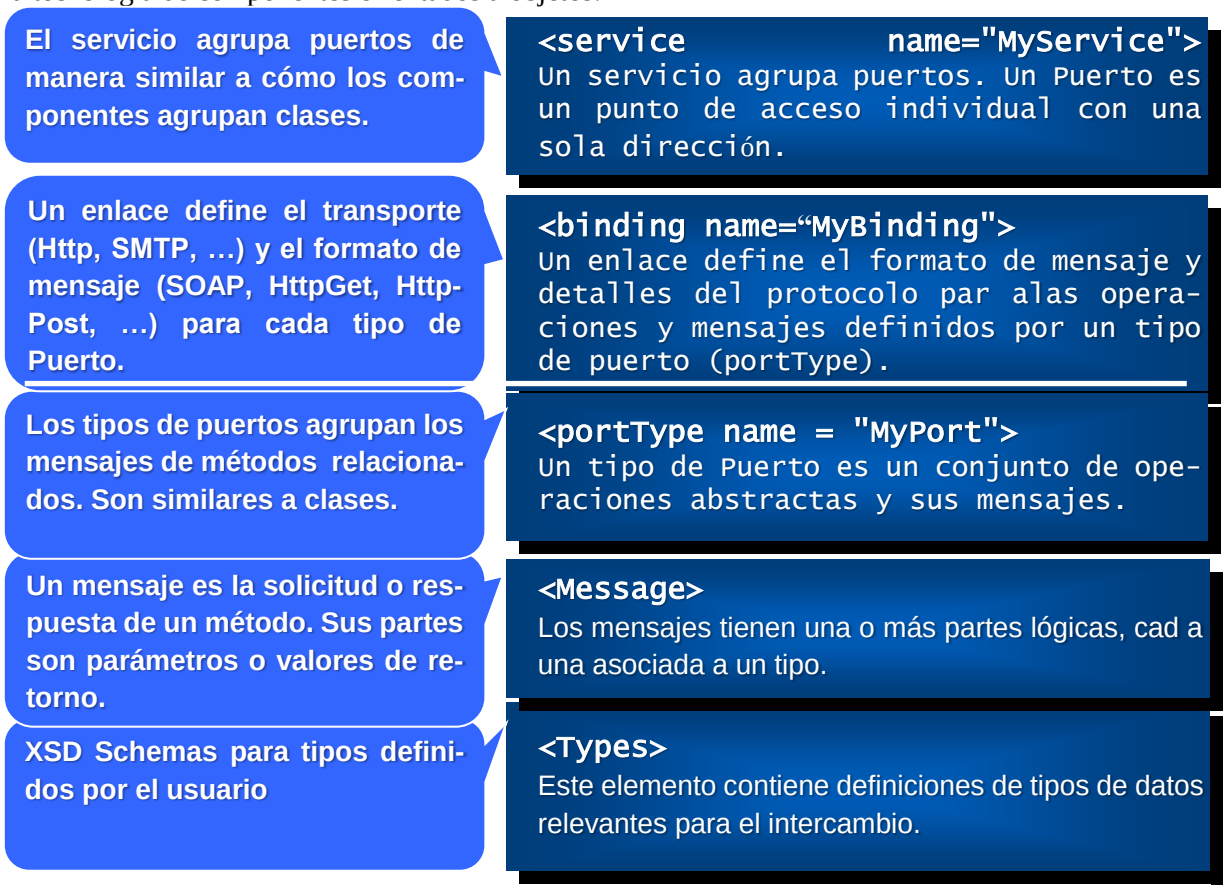


Figura 7. Descripción de WSDL

El siguiente es un ejemplo de descripción WSDL con definiciones para cada una de las partes indicadas:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="WeatherService“
             targetNamespace="http://www.ecerami.com/wsdl/WeatherService.wsdl“
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://www.ecerami.com/wsdl/WeatherService.wsdl"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <message name="getWeatherRequest">
     <part name="zipcode" type="xsd:string"/>
 </message>
 <message name="getWeatherResponse">
     <part name="temperature" type="xsd:int"/>
 </message>
 <portType name="Weather_PortType">
     <operation name="getWeather">
       <input message="tns:getWeatherRequest"/>
       <output message="tns:getWeatherResponse"/>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
    </operation>
 </portType>
<binding name="Weather_Binding" type="tns:Weather_PortType">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getWeather">
      <soap:operation soapAction=""/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:weatherservice"
          use="encoded"/>
      </input>
      <output>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:weatherservice"
          use="encoded"/>
      </output>
    </operation>
 </binding>
<service name="Weather_Service">
    <documentation>WSDL File for Weather Service</documentation>
    <port binding="tns:Weather_Binding" name="Weather_Port">
      <soap:address
        location="http://localhost:8080/soap/servlet/rpcrouter"/>
    </port>
  </service>

</definitions>
```

Los servicios Web descritos por WSDL pueden invocarse gracias a esa descripción. En Internet podemos encontrar herramientas que permiten generar un formulario Web para invocar un determinado servicio que esté descrito con WSDL. Una de estas herramientas es la siguiente:
http://www.soapclient.com/soaptest.html

Registros UDDI de Servicios Web

El proyecto *Universal Description, Discovery & Integration* (UDDI) es fruto de la iniciativa de un consorcio industrial que cuenta con empresas importantes en el área de Internet, como Ariba, IBM o Microsoft, entre otras, y que trabaja en conexión con el consorcio W3C y el IETF. Su dirección Web es: **http://www.uddi.org/**
UDDI es esencialmente un protocolo para la descripción y búsqueda de Servicios Web, junto con las especificaciones para crear y operar un registro compartido común de Servicios en la Web.
El siguiente esquema da una idea general de los objetivos de los registros UDDI, cuya operación puede sintetizarse en cinco pasos:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

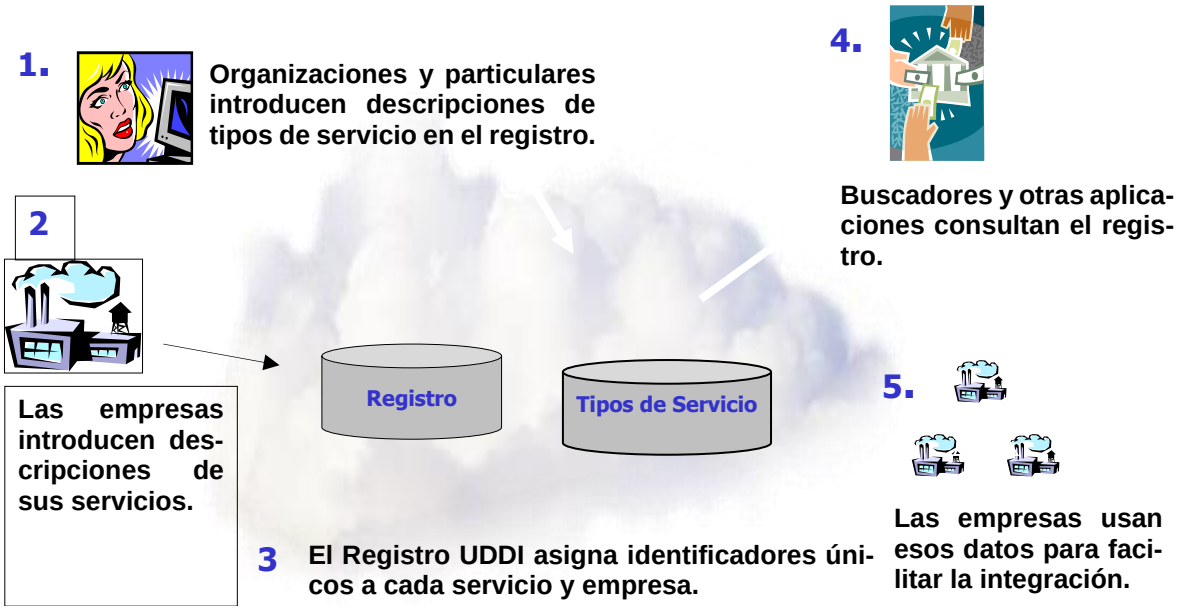


Figura 8. Funcionamiento UDDI

Los tipos de registro que pueden hacerse en los registros UDDI pueden clasificarse en las siguientes categorías:

- Descripciones técnicas de "tipos de registro" típicamente realizadas por organismos de estandarización o programadores.
- Descripciones de negocio, no técnicas, basadas en los tipos anteriores. En este caso son descripciones de empresas determinadas y los servicios que ofrecen, y pueden implementarse páginas blancas, amarillas o verdes.

La arquitectura del registro público de UDDI es una arquitectura distribuida, que se basa en la replicación de la información de sus nodos asociados de manera diaria. Así, una empresa puede registrar su información en cualquier nodo del registro UDDI, y esa información se copiará al resto de los nodos, estando accesible públicamente. Para el registro de información se han especificado un conjunto de operaciones que pueden invocarse a través de mensajes SOAP, con un modelo de datos también estandarizado, que se basa en descripciones de empresas denominadas *BusinessEntity*.
Existe también la posibilidad de crear registros UDDI que no sean de libre acceso, es decir, que estén restringidos o requieran de un pago por los servicios.
La siguiente imagen muestra un ejemplo, con el prototipo de acceso al registro UDDI que proporciona Microsoft.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

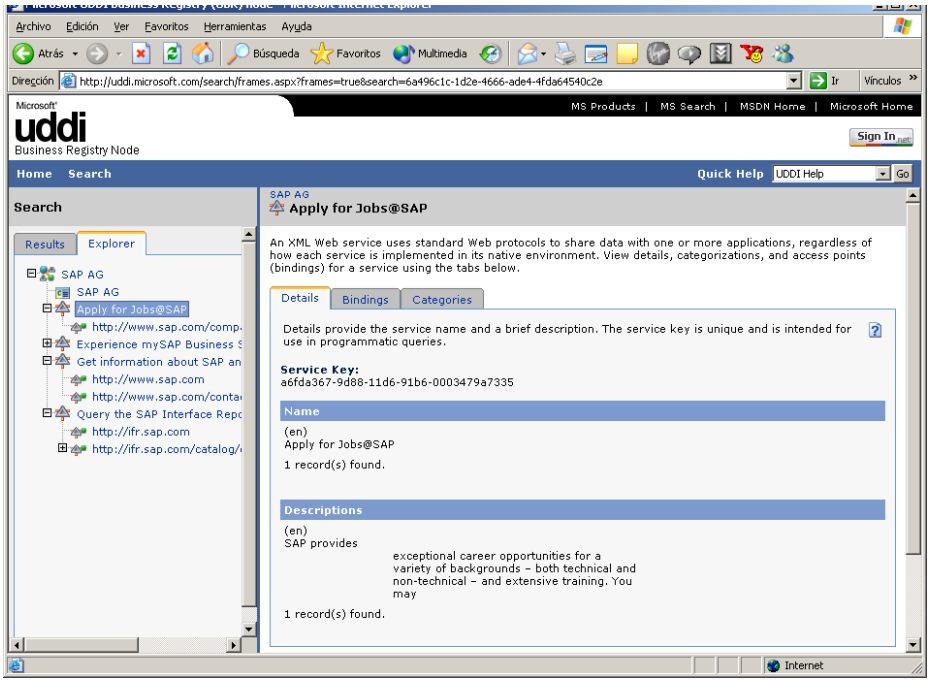


Figura 9: Ejemplo de información UDDI.

En la captura de pantalla se puede apreciar la información de la empresa SAP AG, que proporciona un Servicio Web para solicitar trabajo en la empresa, del cual se muestra su identificador único, y los datos de conexión por Internet (*bindings*).

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Portales

El interés en la expansión de las conexiones creció desde las primeras redes ARPANET, y aparecieron nuevas aplicaciones para ello, las tecnologías de Internet se esparcieron por el resto del mundo.

Varios sitios que no podían conectarse directamente a Internet empezaron a hacerlo por medio de simples portales para permitir la transferencia de correo electrónico, siendo esta última por entonces la aplicación más importante. Esos sitios con sólo conexiones intermitentes usarían UUCP o Fidonet, y confiarían en los portales entre esas redes e Internet. Algunos servicios de portales fueron más allá del simple peering de e-mail, ofreciendo servicios como el acceso a sitios FTP a través de UUCP o e-mail.

Otro servicio que tuvo un enorme auge en los primeros años de Internet fue un servicio de foros y grupos de discusión denominado News. Las News eran un servicio global donde cualquiera podía participar, que funcionó muy bien mientras los grupos de usuarios eran pequeños y homogéneos. Cuando Internet creció el servicio se hizo inmanejable y cayó en desuso. En su lugar aparecieron servicios de creación de comunidades más cerrados que permiten comunicarse a grupos de personas con un interés común. Hoy en día se han incluido funciones similares en muchos portales de Internet, pero donde los gestores de los grupos pueden decidir quién puede acceder y quién no. Por ejemplo, está el servicio de grupos de YAHOO o los servicios que ofrece el Messenger de Microsoft asociados a MSN.

El último Consumer Electronics Show, que se celebró a principios de 2006 confirmó (como era de esperar) la extraodinaria abundancia de innovaciones que se dan en el sector de la electrónica de consumo, pero lo más destacable que se ha podido ver han sido las pretensiones de los grandes motores y portales de búsqueda en esta materia: Google, Yahoo y Microsoft han rivalizado con sus anuncios para restar protagonismo a los actores más tradicionales de la electrónica de consumo como Sony.

El trabajo y la colaboración a través de la red han creado un nuevo tipo de tecnologías destinadas a soportar la colaboración que reciben el nombre de groupware. Estas tecnologías automatizan de forma eficaz actividades tales como: uso de repositorios de documentos compartidos, gestión de flujos de trabajo e información, notificaciones al grupo, foros de discusión diferidos e interactivos, votaciones electrónicas, reuniones a través de la red con videoconferencia, gestión de agendas compartidas, etc. Hoy en día la mayoría de los portales, tanto corporativos como públicos, soportan la creación de servicios para grupos y comunidades [41-45].

Igual que en el mundo de los seres vivos la agrupación en manadas, enjambres o bandadas crea conjuntos muy eficaces, la red ofrece la posibilidad de crear nuevos tipos de grupos donde usuarios, programas o agentes colaboren de forma eficaz. Es en este tipo de aplicaciones donde la red desarrolla el objetivo para la que fue creada y las aplicaciones de comunidades y grupos han florecido y florecerán siempre en Internet. De hecho, la mayoría de las aplicaciones actuales pueden ser consideradas dentro de este apartado desde una cierta perspectiva. Por ejemplo, el Web es una aplicación para crear comunidades que intercambian información y conocimiento o los servicios de portales tales como Yahoo, Terra-Lycos o MSN de Microsoft, dan soporte a grupos de amigos, familias o incluso grupos de actividades y trabajos.

Pero Los portales no solamente sirven para integrar comunidades heterogéneas, si no que sirven para integrar servicios heterogéneos a una comunidad homogénea, como por ejemplo los trabajadores de una empresa, donde a través de un portal, los distintos usuarios de la empresa pueden acceder a distintos servicios tales como: nóminas, vacaciones, boletines informativos y noticias de la empresa,

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

sindicatos, productos, novedades, planes de pensiones, seguros sociales, plantillas de trabajo, listines telefónicos, sincronización de agendas, conocer las tareas adjudicadas por sus jefes, planes de proyectos, presupuestos, fichas y contactos de los clientes, herramientas colaborativas inter e intradepartamentales, gestión documental, workflows, emails, chats, fax, etc. Y todos estos servicios pueden separarse por departamentos, así por ejemplo el departamento de ingeniería no tiene porque disponer de la información de presupuestos de clientes, o el departamento financiero de las tareas asignadas y tiempos de ejecución de un plan de proyecto del departamento de ingeniería, pero todos pueden disponer de una agenda común, minutas, actas de reuniones, y diversos medios de comunicación, teléfono, mensajería, email, sms, etc.

Introducción a los estándares de portales

Con respecto al punto anterior, podemos entonces concluir que el rol de un portal es la integración de aplicaciones a nivel de interfaz de usuario, que disponga de un acceso personalizado a tales aplicaciones o servicios y que permita su uso tanto en entornos de internet como de intranet.
Podemos clasificar los portales como:

- Portales de primera generación
- Portales de segunda generación.

Los portales de primera generación cumplen todo lo especificado anteriormente pero resultan **monolíticos** y son difíciles de desarrollar y mantener.
Por el contrario, los portales de segunda generación permiten que los usuarios dispongan de una o varias páginas compuestas por portlets.
Un **portlet** es una min-iaplicación web interactiva que devuelve fragmentos de markup (HTML, WML, etc.). El portal construye cada página agregando las respuestas de los portlets que contiene y decora las respuestas de los portlets con una barra de título y un conjunto de botones (minimizar/maximizar, editar, etc.). Cada portlet es un componente que se puede instalar fácilmente en el portal e implementa un conjunto de interfaces.
Los portlets surgieron hace tiempo en el contexto de Internet (my.yahoo.com, my.excite.com, my.lycos.com, etc.), y ahora se hacen populares en las intranets. Ofrecen una vista personalizada y restringida de la información interna de la empresa y en los últimos años han surgido un gran número de servidores de portales ("portal servers") Ejemplos: BEA WebLogic Portal, IBM WebSphere Portal, Sun Java System Portal Server, Oracle Portal, Vignette Portal, Microsoft SharePoint Portal Server, Liferay Portal, Jboss Portal, Jakarta Jetspeed, eXo Platform, etc.
Estas plataformas suelen proporcionar un portal pre-construido en el que es posible instalar ("deploy") portlets.

¿Por qué son necesarios los estándares?

Las primeras versiones de los servidores de portales presentan problemas de incompatibilidad, tales como que un portal no puede consumir remotamente los portlets instalados en otro portal, o que los portlets desarrollados con un determinado portal no se pueden instalar en otro portal.
Debido a esta problemática recientemente han surgido estándares que hacen frente a estos problemas. La mayor parte de los fabricantes han adaptado (o están adaptando) sus servidores de portales para soportar estos estándares.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Por ejemplo, en un entorno como el siguiente:

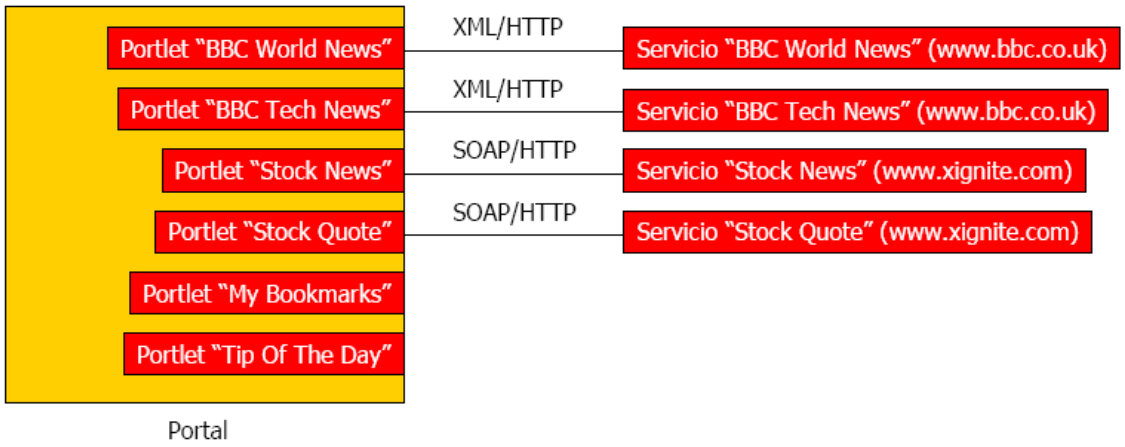


Figura 9. Portal accediendo a contenidos externos

Cada Portlet es local al portal y proporciona una interfaz gráfica en respuesta a las acciones del usuario, asistente de personalización, información de ayuda, etc. También puede proporcionar datos o es un proxy de un servicio externo al portal (que sólo ofrece datos).

Las consecuencias en este sentido son que cualquier portal que integre los servicios de la BBC o de Xignite debe re-implementar la interfaz gráfica de los portlets, dado que los servicios sólo ofrecen datos. Si se desea construir otro portal que use los portlets "My Bookmarks" y "Tip Of The Day" es preciso volver a instalarlos en el nuevo portal (si está construido con el mismo servidor de portales) o desarrollarlos completamente (en otro caso).

Este caso sería probable dentro de una gran corporación con distintos portales (e.g. uno para cada división) que desean compartir algunos portlets.

La solución pasa por utilizar portlets remotos. Cada productor de portlets implementa un servicio Web estándar (definido en WSDL) que permite que los consumidores (típicamente portales) interactúen con sus portlets. Entre otras, el interfaz ofrece una operación que recibe como parámetro los datos correspondientes a una interacción que realiza el usuario sobre un portlet (e.g. clic en un enlace) y devuelve el nuevo markup del portlet. Si un portal desea exportar algunos de sus portlets locales, sólo necesita proporcionar un productor.

El administrador del portal puede añadir portlets remotos sin necesidad de programación. Todos los productores implementan el mismo interfaz [46-48].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

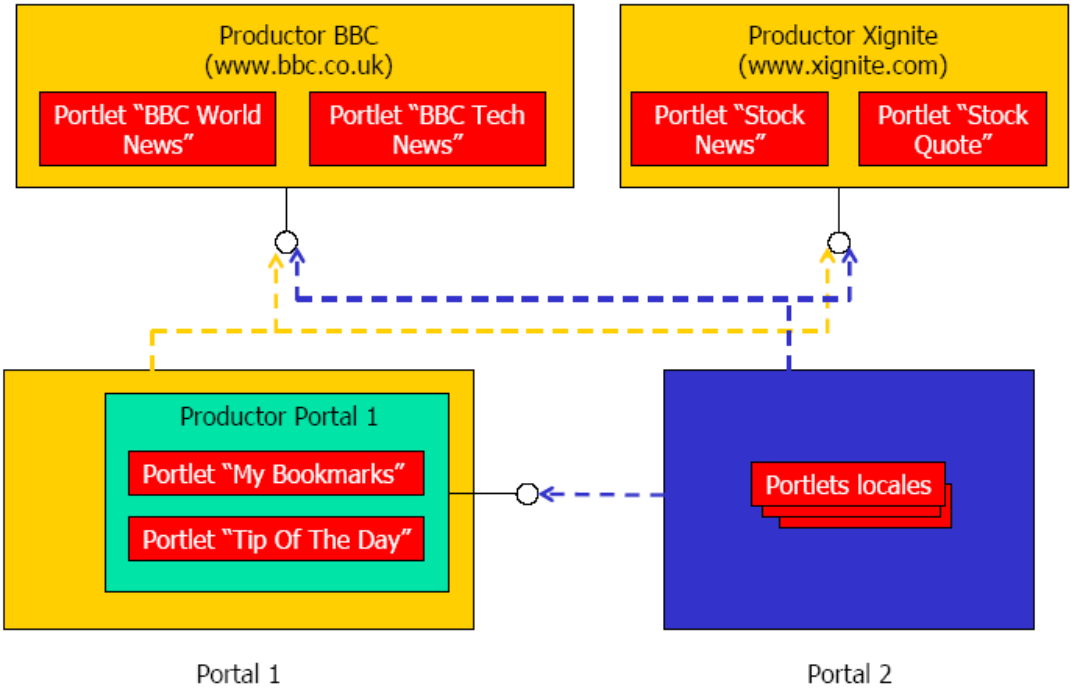


Figura 10. Portlets remotos

En Septiembre del 2003, OASIS (Organization of the Advancement of Structured Information Standards) publicó la primera versión del estándar WSRP (Web Services for Remote Portlets)
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp
Especifica el conjunto de interfaces (definidos en WSDL) que debe implementar un productor de portlets. Tanto productor y consumidor pueden usar distintas tecnologías (J2EE, .NET, etc.)
Debería ser posible instalar los portlets desarrollados en un servidor de portales en otro diferente, siempre que usen la misma tecnología (e.g. J2EE) yevitar dependencias con respecto al fabricante, por lo que se precisa un API estándar para cada lenguaje de programación. El estándar WSRP no define este API, sólo define un API para exportar los portlets de un productor a aplicaciones remotas, y no para desarrollar portlets locales.
La mayor parte de los portlets de un portal serán locales (por eficiencia y porque muchos serán específicos al portal) y los interfaces de WSRP son de muy bajo nivel. Un portlet es un componente que genera markup, como una aplicación Web convencional.
Por lo que se precisa un API que permita desarrollar portlets locales con las mismas tecnologías que se usan para desarrollar aplicaciones Web (e.g. JSP o ASP.NET).
En Octubre del 2003, se publicó la primera versión de la especificación de portlets Java (JSR 168)
http://jcp.org/aboutJava/communityprocess/final/jsr168

- Un portlet Java se puede instalar en cualquier contenedor de portlets estándar
- Es compatible con WSRP

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Microsoft SharePoint incluso soporte para la programación de “Web Parts” que es equivalente al concepto de portlets Java

La Especificación Portlet

Como hemos definido anteriormente, los portlets son componentes web desarrollados en Java, gestionados por un contenedor de portlets, que procesa las peticiones y genera contenido dinámico. Los portales usan portlets como componentes de interfaz de usuarios que proporcionan una capa de presentación a los sistemas de información.

El objetivo del **JSR 168(Java Specification Request),** es permitir la interoperabilidad entre portlets y portales. Está especificación define el contrato entre portlets y el contenedor de portlets, y un conjunto de APIs para manejar la personalización, presentación y seguridad.

Con el aumento de un gran número de portales empresariales, varios vendedores han creado APIs diferentes para los componentes de los portales, llamados portlets. Esta variedad de interfaces incompatibles genera problemas para los proveedores de aplicaciones.

JSR 168 define los portlets como **componentes web Java, gestionados por un contenedor de portlets,** que procesa las peticiones y genera contenido dinámico. Los objetivos del JSR 168 son los siguientes:

1. Definir el entorno de ejecución, o el contenedor, para los portlets.
2. Definir el API entre el contenedor y los portlets.
3. Proporcionar los mecanismos para almacenar datos transitorios y existentes para los portlets.
4. Proporcionar un mecanismo que permita a los portlets incluir Servlets y JSP.
5. Definir un empaquetamiento de portlets para permitir un despliegue sencillo.
6. Permitir la portabilidad binaria de los portlets entre portales que cumplan el JSR 168.
7. Ejecutar los portlets JSR 168 usando el protocolo para portlets remotos con servicios web (WSRP).

JSR 168 es un grupo formado por distintas empresas como son: Apache, ATG, BEA, Boeing, Borland, Broadvision, Citrix, EDS, Fujitsu, Hitachi, IBM, Novell, Oracle, SAP, SAS, Sun Microsystems, Sybase, TIBCO y Vignette.

Si hemos definido un **portal** como una aplicación web que proporciona personalización, single singon, y agregación de contenido desde diferentes fuentes, y gestiona la capa de presentación de los sistemas de información. Agregación es el proceso de integrar contenido desde diferentes fuentes dentro de una página web, una **ventana de portlets** consiste en una barra de título con el título del portlet, adornos, y el contenido producido por el portlet. Los adornos pueden incluir botones para cambiar el estado y el modo de los portlets [49].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

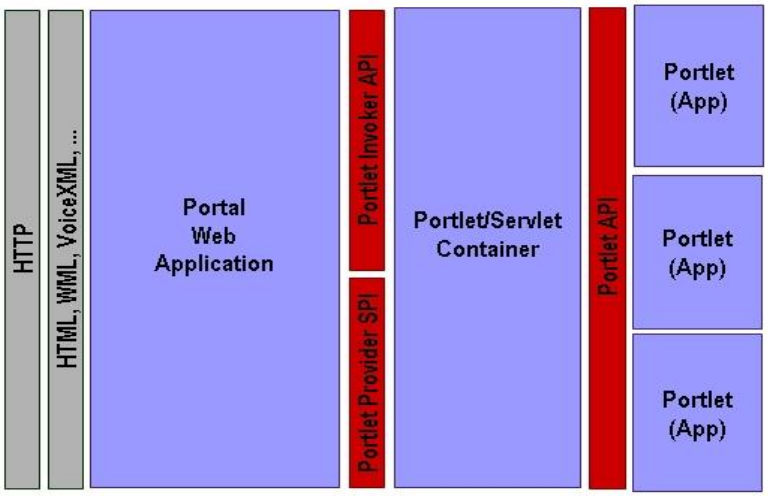


Figura 11. Arquitectura Portlet

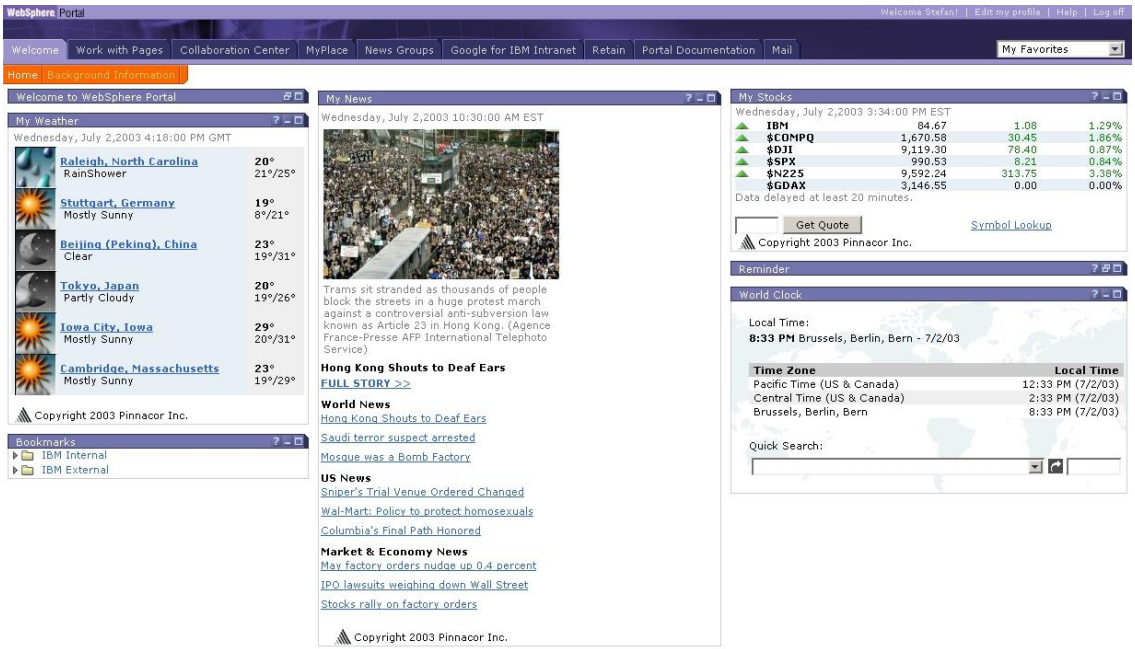


Figura 12. Ejemplo portal con portlets

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Como se ha mencionado anteriormente, un portlet es un componente web Java que procesa las peticiones y genera contenido dinámico. El contenido generado por un portlet es llamado un fragmento, un trozo de lenguaje de marcas (HTML, XML, XHTML, WML) junto con algunas reglas. Un fragmento puede ser agregado con otros fragmentos para formar un documento completo. El contenedor de portlets gestiona el ciclo de vida del portlet.

Los clientes web interactúan con los portlets mediante el paradigma petición-respuesta implementado por el portal. Normalmente, el usuario interactúa con el contenido producido por los portlets, por ejemplo, siguiendo los enlaces o enviando formularios, resultando como acciones de los portlets al ser recibidas por el portal, el cual a continuación reenviará a los portlets de interés las interacciones del usuario. El contenido generado por un portlet puede variar de un usuario a otro dependiendo de la configuración de usuario del portlet [50].

Contenedor de portlets.

Un contenedor de portlets ejecuta los portlets y les proporciona un entorno de ejecución, y gestiona su ciclo de vida. También proporcionan mecanismos de almacenamiento persistente a las preferencias del portlet. Un contenedor de portlets recibe las peticiones desde el portal. No es el responsable de agregar el contenido producido por los portlets; es el portal en sí mismo el que maneja la agregación.

Ciclo de vida del portlet.

El ciclo de vida propuesto en el JSR 168 es:

1. **Inicio**: inicialista del portlet y lo pone dentro del servicio.
2. **Manejar peticiones**: procesa diferentes clases de acciones y maneja las peticiones.
3. **Destrucción**: saca al portlet fuera de servicio.

El contenedor de portlets gestiona el ciclo de vida del portlet y llama a los correspondientes métodos en el interfaz del portlet.

Interfaz del portlet.

Cada portlet debe implementar el interfaz portlet, o extender de una clase que implemente dicho interfaz. El interfaz consiste en los siguientes métodos:

1. **init (PortletConfig config):** para inicializar el portlet. Este método es llamado sólo una vez después de instanciar el portlet. Este método puede ser usado para crear objetos complejos o recursos para ser usados por el portlet.
2. **processAction(ActionRequest request, ActionResponse response)**: notifica al portlet que el usuario ha lanzado una acción a este portlet. Sólo se puede lanzar las solicitudes de una única acción por cliente. En una acción, un portlet puede manejar una dirección, un cambio en el modo o en el estado de la ventana, modificar su estado persistente, o realizar operaciones con los parámetros.
3. **render(RenderRequest request, RenderResponse response):** genera el lenguaje de marcas. Por cada portlet en la página actual el método render es llamado, y el portlet puede producir el lenguaje de marcas, el cual dependerá del modo o el estado de la ventana, los parámetros, atributos, estado persistente, datos de la sesión, o datos del backend.
4. **destroy():** indica al portlet su ciclo de vida ha terminado. Este método permite al portlet liberar recursos y actualizar cualquier dato persistente que pertenezca a este portlet.

Modos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Un modo indica la función que debe llevar a cabo el portlet. Un modo avisa al portlet que tarea debería llevar a cabo y que contenido debería ser generado. Cuando se invoca un portlet, el contenedor de portlets proporciona el modo actual al portlet. Los portlets pueden programáticamente cambiar su modo mediante el cual se procesa una solicitud de acción. JSR 168 establece los modos en tres categorías:

1. **Modos obligatorios**: cada portal debe soportar los modos edición, ayuda, y vista. Un portlet debe por lo menos soportar el modo vista, usado para mostrar el lenguaje de marcas de una página. El modo edición para cambiar las propiedades de usuario y personalizar el lenguaje de marcas final, y el modo que ayuda es usado para mostrar una pantalla de ayuda.

2. **Modos opcionales**: Hay modos que un portal puede soportar. Estos incluyen el modo "acerca de", el modo de configuración, para permitir a los administradores configurar el portlet; el modo Edit_defaults permite a un administrador editar los valores por defecto del portlet; el modo previsualización muestra una previsualización del portlet; y el modo imprimir muestra una vista que es fácil de imprimir.

3. **Modos específicos del vendedor**: estos modos no están definidos en la especificación y son propietarios del vendedor.

Estados.
Indican la cantidad de espacio de la página del portal que será asignada al contenido generado por el portlet. JSR 168 define los siguientes estados de ventana:

1. **Normal**: indica que un portlet puede compartir la página con otros portlets. Este es el estado por defecto.
2. **Maximizado**: indica que un portlet puede ser el único portlet de la página del portal, o que el portlet tienen más espacio comparado con el resto de portlets.
3. **Minimizado**: indica que el portlet debería mostrar una salida mínima o incluso sin salida.

Almacenamiento persistente.
El portlet puede almacenar datos existentes para un usuario específico usando el objeto *PorletPreferences*. Las preferencias pueden ser leídas y escritas en la fase de acción, y sólo leídas en la fase de presentación. El modo preferido para escribir las preferencias es el modo de edición, el cual proporciona al usuario una pantalla de personalización. Las preferencias pueden ser preconfiguradas con valores por defecto en el descriptor de despliegue.

Sesiones.
JSR 168 incorpora el concepto de sesión basado en el objeto *HttpSession* definido para las aplicaciones web. Se usa la misma sesión que los Servlets. El alcance de la sesión predeterminada es el alcance del portlet. JSR 168 además soporta el alcance de sesión de aplicación web, en el cual la información puede ser usada para compartir un estado transitorio entre diferentes componentes de la misma aplicación web (por ejemplo: entre portlets, o entre un portlet y un Servlet).

Introduciendo Servlets y JSP.
Para soportar el patrón modelo-vista-controlador, el portlet debe ser capaz de incluir contenido generado desde Servlets y JSP. De esta manera el portlet puede actuar como controlador, completar un

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

JavaBean con datos, e incluir una página JSP en para mostrar la salida. El mecanismo de inclusión es el mismo que para los Servlets sólo que vía contexto del portlet:

```
PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(editJSP);
  rd.include(portletRequest, portletResponse);
```

Alineamiento con WSRP

WSRP (Web Services for Remote Portlets) es un intento de crear un protocolo de agregación de portlets remotos, definido por el OASIS. Agrega contenido producido por portlets que se ejecutan en máquinas remotas que usan diferentes entornos de programación, como J2EE y .NET. Los servicios de WSRP están orientados a la presentación, a usar un sistema plug and play con portales u otras aplicaciones. No es necesaria una adaptación específica para consumir portales; los portales pueden ser fácilmente agregados sin esfuerzo de programación. En este sentido cabe destacar un esfuerzo que se está haciendo por medio de varias empresas como Plumtree o BEA en el desarrollo de un proyecto opensource denominado POST (Portal Open Source Trading) y que puede encontrarse en el sitio *http://portlet-opensrc.sourceforge.net.*

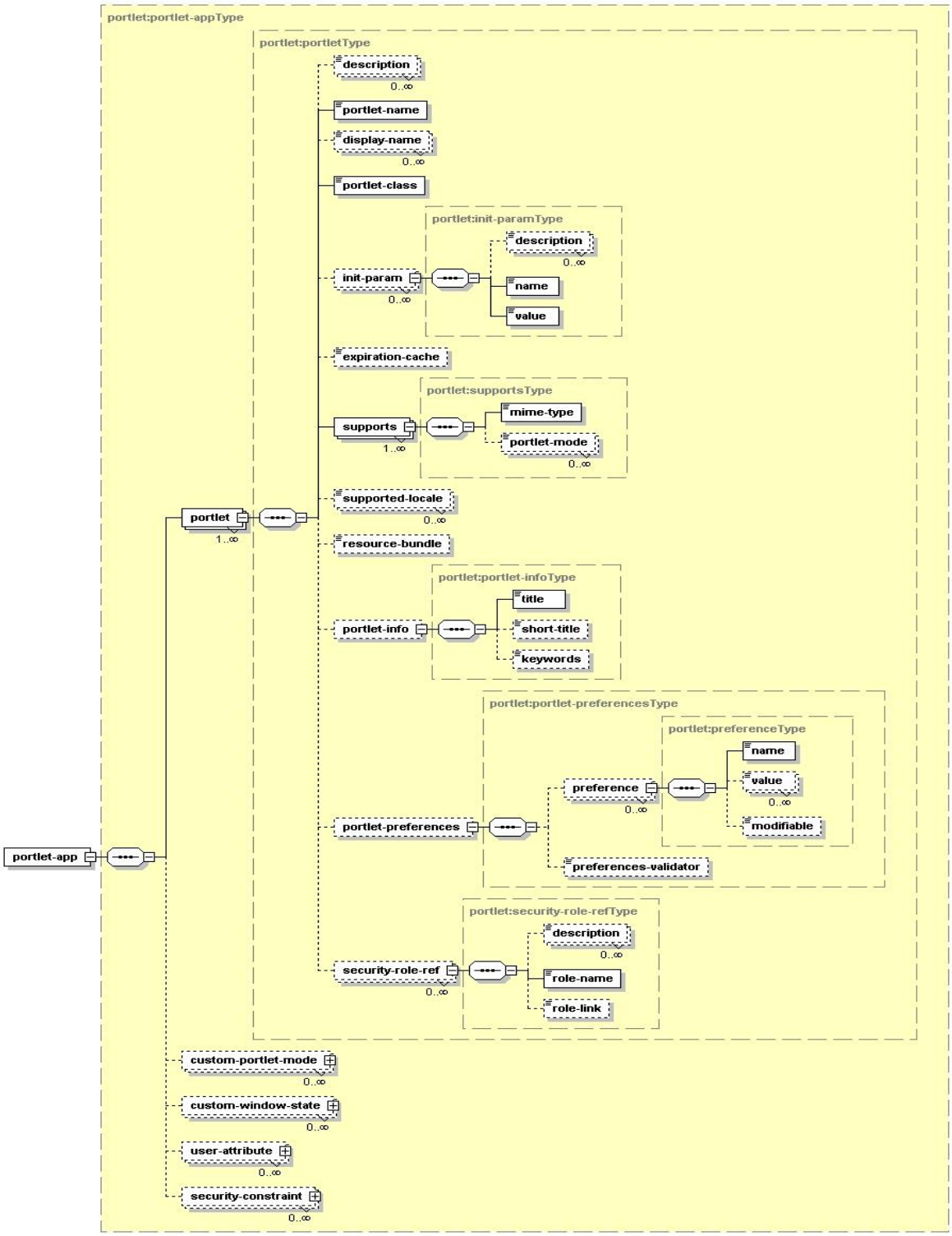
Empaquetamiento y despliegue.

Los recursos, portlets, y descriptores de despliegue son empaquetados juntos en un archivo de aplicación web (archivo war). En contraste con las aplicaciones orientadas sólo a Servlets, en las aplicaciones de portlets consisten en dos descriptores de despliegue: uno para especificar los recursos del aplicación web (web.xml) y otro para especificar recursos del portlet (portlet.xml). Todo lo recursos web que no son portlets deben ser especificados en el descriptor web.xml. Todos los portlets y configuraciones relacionadas con un portlet deben ser especificados en un archivo adicional llamado portlet.xml. Sin embargo hay tres excepciones a esta regla, y son todas definidas en el archivo web.xml, y son válidas para toda la aplicación web:

1. La descripción de la aplicación portlet.
2. El nombre de la aplicación portlet.
3. La seguridad y el role mapping de la aplicación portlet.

Una descripción del archivo descriptor de despliegue portlet.xml se muestra a continuación:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc



ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Figura 13. Descriptor de despliegue portlet

El despliegue de una aplicación de portlets es un despliegue en dos pasos, por un lado se produce el despliegue de la aplicación web en el servidor de aplicaciones y por otro los portlets en el servidor del portal. Para lograr esta tarea, el contenedor de portlets debe inyectar artefactos Servlet dentro de cada archivo war que conforma la aplicación portlet. Como se muestra en la figura siguiente, el componente portlet, coge el archivo war original a continuación inyecta un nuevo web.xml modificado y un Servlet para envolver cada portlet, y es usado como punto de inicio punto durante la invocación del portlet. El contenedor de portlets llama al Servlet inyectado como punto de entrada en el archivo war desplegado.

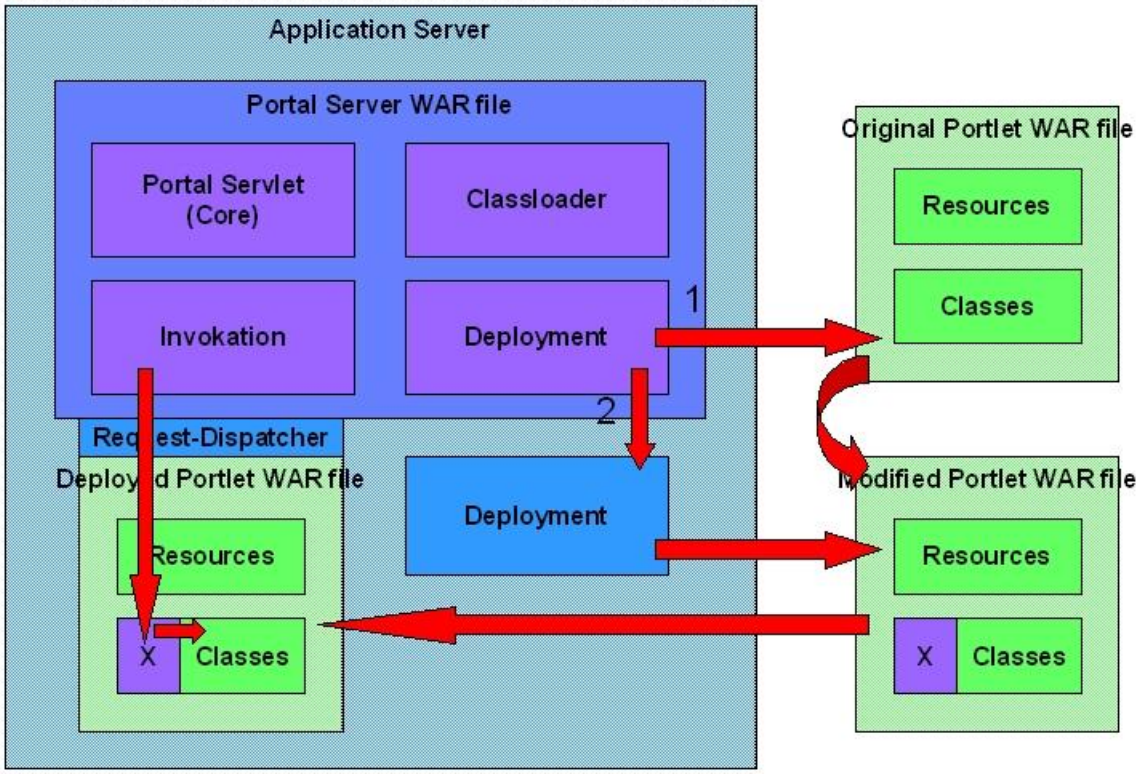


Figura 14. Deployment Portlet

4.6 Proyecto Pluto

El grupo Apache (http://jakarta.apache.org) mantiene actualmente la elaboración de un API fundamentado en el trabajo desarrollado por el JSR 168. Su labor es crear un framework de desarrollo que sea estándar para todas las implementaciones de portales que estén basadas en el JSR 168 y dentro del proyecto incluyen un contenedor de portlets para realizar las pruebas.

La distribución se baja en formato comprimido y la instalación pasa por proceso simple de la descompresión; viene con la configuración necesaria para ser compilado y distribuido usando la herramienta Ant, aunque si bien esto es posible, en la documentación recomiendan el uso de la herramienta

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Maven de gestión de proyectos también desarrollada por el grupo apache, que permite la administración, configuración, compilación y despliegue de proyectos.
La implementación de un portlet simple mediante el API del proyecto Pluto podría ser el siguiente:

```
import javax.portlet.*;
import java.io.IOException;
public class BookmarkPortlet extends GenericPortlet
{
   public void doView (RenderRequest request, RenderResponse response) throws PortletException, IOException
   {
      response.setContentType("text/html");
      String jspName = getPortletConfig().getInitParameter("jspView");
    PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(jspName);
      rd.include(request,response);
   }
   public void doEdit (RenderRequest request,
                  RenderResponse response) throws PortletException, IOException
   {
      response.setContentType("text/html");
      String jspName = getPortletConfig().getInitParameter("jspEdit");
      PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(jspName);
      rd.include(request,response);
   }
}
```

Y a continuación una de las páginas JSP que permite el modo edición (edit.jsp).

```
<%@ page session="false" %>
<%@ page import="javax.portlet.*"%>
<%@ page import="java.util.*"%>
<%@ taglib uri='/WEB-INF/tld/portlet.tld' prefix='portlet'%>
<portlet:defineObjects/>
Hello,<br>I am the bookmark portlet.<br><br>
Current Portlet Mode: <%=renderRequest.getPortletMode()%><br>
Current Window State: <%=renderRequest.getWindowState()%><br>
<br>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Páginas web de consulta.

- Área de descargas de BEA
- [http://commerce.bea.com/index.jsp]
- Área de descargas de ATG
- [http://www.atg.com/en/myatg/mydownloads.jhtml]
- The Server Side: Web de recursos sobre servidores de aplicaciones basados en J2EE.
- [http://theserverside.com]
- Información del proyecto Pluto
- [http://jakarta.apache.org/pluto/]
- Información acerca del WSRP
- [http://www.oasis-open.org/committees/wsrp/]
- Página del W3C sobre la iniciativa de los Servicios Web, incluye el documento de arquitectura de los Servicios Web.
- [http://www.w3.org/2002/ws/]
- Página del consorcio de interoperabilidad de Servicios Web WS-I:
- [http://www.ws-i.org/]
- Página de la comunidad SOAPBuilders, dedicada a la interoperabilidad de los servicios Web.
- [http://www.soapbuilders.org/]
- Página del servidor de aplicaciones J2EE de fuente abierto *jBoss*.
- [http://www.jboss.org/index.html]

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

References

1. Roman, E., Ambler, S.W., Jewell, T. (2001) *Mastering Enterprise JavaBeans* (2nd Edition), John Wiley & Sons.
2. Allaramaju, S. et al. (2002) *Professional Java Server Programming, J2EE 1.3*. Sams.
3. Chappell, D.A., Jewell, T. (2002). *Java Web Services.* O'Reilly & Associates.
4. Gamma, E. *et al*. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
5. Alexandre Silvestre Ferreira, Aurora Pozo, Richard Aderbal Gonçalves (2015) An Ant Colony based Hyper-Heuristic Approach for the Set Covering Problem. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
6. Casado-Vara, R., & Corchado, J. (2019). Distributed e-health wide-world accounting ledger via blockchain. Journal of Intelligent & Fuzzy Systems, 36(3), 2381-2386.
7. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
8. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
9. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
10. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
11. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
12. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
13. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
14. Chamoso, P., Rivas, A., Martín-Limorti, J. J., & Rodríguez, S. (2018). A Hash Based Image Matching Algorithm for Social Networks. In Advances in Intelligent Systems and Computing (Vol. 619, pp. 183–190). https://doi.org/10.1007/978-3-319-61578-3_18
15. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
16. Corchado, J. A., Aiken, J., Corchado, E. S., Lefevre, N., & Smyth, T. (2004). Quantifying the Ocean's CO2 budget with a CoHeL-IBR system. In Advances in Case-Based Reasoning, Proceedings (Vol. 3155, pp. 533–546).
17. Corchado, J. M., & Aiken, J. (2002). Hybrid artificial intelligence methods in oceanographic forecast models. Ieee Transactions on Systems Man and Cybernetics Part C-Applications and Reviews, 32(4), 307–313. https://doi.org/10.1109/tsmcc.2002.806072
18. Corchado, J. M., & Fyfe, C. (1999). Unsupervised neural method for temperature forecasting. Artificial Intelligence in Engineering, 13(4), 351–357. https://doi.org/10.1016/S0954-1810(99)00007-2
19. Corchado, J. M., Borrajo, M. L., Pellicer, M. A., & Yáñez, J. C. (2004). Neuro-symbolic System for Business Internal Control. In Industrial Conference on Data Mining (pp. 1–10). https://doi.org/10.1007/978-3-540-30185-1_1
20. Corchado, J. M., Corchado, E. S., Aiken, J., Fyfe, C., Fernandez, F., & Gonzalez, M. (2003). Maximum likelihood hebbian learning based retrieval method for CBR systems. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 2689, pp. 107–121). https://doi.org/10.1007/3-540-45006-8_11

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

21. Corchado, J. M., Pavón, J., Corchado, E. S., & Castillo, L. F. (2004). Development of CBR-BDI agents: A tourist guide application. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 3155, pp. 547–559). https://doi.org/10.1007/978-3-540-28631-8
22. Corchado, J., Fyfe, C., & Lees, B. (1998). Unsupervised learning for financial forecasting. In Proceedings of the IEEE/IAFE/INFORMS 1998 Conference on Computational Intelligence for Financial Engineering (CIFEr) (Cat. No.98TH8367) (pp. 259–263). https://doi.org/10.1109/CIFER.1998.690316
23. Cristian Peñaranda, Jorge Agüero, Carlos Carrascosa, Miguel Rebollo, Vicente Julián (2016). An Agent-Based Approach for a Smart Transport System. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
24. Daniel Ayala, Juan C. Roldán, David Ruiz, Fernando O. Gallego (2015). An approach for discovering keywords from Spanish tweets using Wikipedia. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 2
25. David Griol, José Molina (2015). Measuring the differences between human-human and human-machine dialogs. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 2
26. Di Mascio, T., Vittorini, P., Gennari, R., Melonio, A., De La Prieta, F., & Alrifai, M. (2012, July). The Learners' User Classes in the TERENCE Adaptive Learning System. In 2012 IEEE 12th International Conference on Advanced Learning Technologies (pp. 572-576). IEEE.
27. Fábio Silva, Cesar Analide (2015). Tracking Context-Aware Well-Being through Intelligent Environments. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 2
28. Fdez-Riverola, F., & Corchado, J. M. (2003). CBR based system for forecasting red tides. Knowledge-Based Systems, 16(5–6 SPEC.), 321–328. https://doi.org/10.1016/S0950-7051(03)00034-0
29. Fyfe, C., & Corchado, J. (2002). A comparison of Kernel methods for instantiating case based reasoning systems. Advanced Engineering Informatics, 16(3), 165–178. https://doi.org/10.1016/S1474-0346(02)00008-3
30. Fyfe, C., & Corchado, J. M. (2001). Automating the construction of CBR systems using kernel methods. International Journal of Intelligent Systems, 16(4), 571–586. https://doi.org/10.1002/int.1024
31. Gabriel Santos, Tiago Pinto, Zita Vale, Isabel Praça, Hugo Morais (2016). Enabling Communications in Heterogeneous Multi-Agent Systems: Electricity Markets Ontology. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
32. Gabriele Di Giammarco, Tania Di Mascio, Michele Di Mauro, Antonietta Tarquinio, Pierpaolo Vittorini (2015). SmartHeart CABG Edu. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
33. García, O., Chamoso, P., Prieto, J., Rodríguez, S., & De La Prieta, F. (2017). A serious game to reduce consumption in smart buildings. In Communications in Computer and Information Science (Vol. 722, pp. 481–493). https://doi.org/10.1007/978-3-319-60285-1_41
34. Glez-Bedia, M., Corchado, J. M., Corchado, E. S., & Fyfe, C. (2002). Analytical model for constructing deliberative agents. International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications, 10(3).
35. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
36. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
37. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

38. Jesús Ángel Román Gallego, Sara Rodríguez González (2015). Improvement in the distribution of services in multi-agent systems with SCODA. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 3
39. José Alemany, Stella Heras, Javier Palanca, Vicente Julián (2016). Bargaining agents based system for automatic classification of potential allergens in recipes. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
40. Juan Carlos Alvarado-Pérez, Diego H. Peluffo-Ordóñez, Roberto Therón (2015). Bridging the gap between human knowledge and machine learning. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
41. Laza, R., Pavn, R., & Corchado, J. M. (2004). A reasoning model for CBR_BDI agents using an adaptable fuzzy inference system. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 3040, pp. 96–106). Springer, Berlin, Heidelberg.
42. Manuel Gómez Zotano, Jorge Gómez-Sanz, Juan Pavón (2015). User Behavior in Mass Media Websites. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 3
43. Méndez, J. R., Fdez-Riverola, F., Díaz, F., Iglesias, E. L., & Corchado, J. M. (2006). A comparative performance study of feature selection methods for the anti-spam filtering domain. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4065 LNAI, 106–120. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-33746435792&partnerID=40&md5=25345ac884f61c182680241828d448c5
44. Mohamed Frikha, Mohamed Mhiri, Faiez Gargouri (2015). A Semantic Social Recommender System Using Ontologies Based Approach For Tunisian Tourism. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
45. Pablo Chamoso, Fernando De La Prieta (2015). Simulation environment for algorithms and agents evaluation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 3
46. Palomino, C. G., Nunes, C. S., Silveira, R. A., González, S. R., & Nakayama, M. K. (2017). Adaptive agent-based environment model to enable the teacher to create an adaptive class. Advances in Intelligent Systems and Computing (Vol. 617). https://doi.org/10.1007/978-3-319-60819-8_3
47. Ricardo Azambuja Silveira, Rafaela Lunardi Comarella, Ronaldo Lima Rocha Campos, Jonas Vian, Fernando De La Prieta (2015). Learning Objects Recommendation System: Issues and Approaches for Retrieving, Indexing and Recomend Learning Objects. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 4
48. Ricardo Silveira, Guilherme Klein Da Silva Bitencourt, Thiago Ângelo Gelaim, Jerusa Marchi, Fernando De La Prieta (2015). Towards a Model of Open and Reliable Cognitive Multiagent Systems: Dealing with Trust and Emotions. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 3
49. Rodriguez-Fernandez J., Pinto T., Silva F., Praça I., Vale Z., Corchado J.M. (2018) Reputation Computational Model to Support Electricity Market Players Energy Contracts Negotiation. In: Bajo J. et al. (eds) Highlights of Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection. PAAMS 2018. Communications in Computer and Information Science, vol 887. Springer, Cham.
50. Silvia Rossi, Francesco Barile, Antonio Caso (2015). Dominance Weighted Social Choice Functions for Group Recommendations. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Creación de portales para B2C:

David Palomar Delgado [1]

[1] University Carlos III – Calle Madrid, 126, 28903 Getafe, Madrid, Spain
dpalomar@inf.uc3m.es

Resumen: En este capítulo ofrecemos una panorámica general de dos productos comerciales muy utilizados actualmente en el desarrollo profesional de portales y sitios Web de comercio electrónico. Aunque estos productos actualmente también proporcionan soporte para desarrollar procesos de B2B, aquí nos centraremos fundamentalmente en el desarrollo de portales mediante com-ponentes preparados para su reutilización (a veces llamados portlets). La visión sucinta pro-porcionada en este documento puede complementarse con el uso de licencias de evaluación de los productos que se tratan, que cuentan con tutoriales que dan una idea más práctica de las funcionalidades proporcionadas. En el apartado de "Recursos en la Web" pueden encontrarse las direcciones Web de estos productos. Dentro del mundo Java existen otras alternativas opensource para la creación de portales que contienen todos los servicios anteriormente mencionados por las otras plataformas y que cumplen los estándares de industria, y que son alternativas perfectamente válidas para un pro-yecto de grandes prestaciones. En este capítulo se revisarán las alternativas existentes.

Palabras clave: B2B; portales web

Abstract. In this chapter we offer an overview of two commercial products currently widely used in the professional development of e-commerce portals and websites. Although these products currently also provide support for developing B2B processes, here we will focus primarily on the development of portals using reusable speakers (sometimes called portlets). The succinct vision provided in this document can be complemented with the use of evaluation licenses for the products in question, which include tutorials that give a more practical idea of the functionalities provided. The Web addresses of these products can be found in the "Web Resources" section. Within the Java world there are other opensource alternatives for the creation of portals that contain all the services previously mentioned by the other platforms and that meet industry standards, and that are perfectly valid alternatives for a project of great features. In this chapter we will review the existing alternatives.

Keywords: B2B; web sites

Introducción

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Los grandes sitios de comercio electrónico y los portales Web gestionan una enorme cantidad de páginas dinámicas, y dan soporte a procesos de negocio B2C que requieren varios pasos para su realización, y que deben estar adecuadamente soportados por procesos automatizados para ser eficientes. La complejidad y dificultad de mantenimiento de este tipo de aplicaciones requiere de software especializado, ya que "partir de cero" en estos casos supone un esfuerzo de desarrollo tan grande que difícilmente encaja en los calendarios de desarrollo. Por otro lado, las funcionalidades y servicios proporcionados por estas aplicaciones son **muy similares** de un sistema a otro, lo cual facilita la reutilización de componentes y el uso de herramientas uniformes. Por todo ello, es importante conocer las posibilidades que nos ofrecen los paquetes comerciales a la hora de decidir sobre su uso y de estimar el esfuerzo necesario para desarrollar sistemas de comercio electrónico utilizándolos.
En este tema ofrecemos una panorámica general de dos productos comerciales muy utilizados actualmente en el desarrollo profesional de portales y sitios Web de comercio electrónico. Aunque estos productos actualmente también proporcionan soporte para desarrollar procesos de B2B, aquí nos centraremos fundamentalmente en el desarrollo de portales mediante componentes preparados para su reutilización (a veces llamados *portlets*). La visión sucinta proporcionada en este documento puede complementarse con el uso de licencias de evaluación de los productos que se tratan, que cuentan con tutoriales que dan una idea más práctica de las funcionalidades proporcionadas. En el apartado de "Recursos en la Web" pueden encontrarse las direcciones Web de estos productos.
Ambas plataformas están basadas en el estándar J2EE de Java, y por tanto, descansan en el soporte de los *Enterprise Java Beans* (EJB) – véase (Roman, Ambler & Jewell, 2001) – aunque los servicios que proporcionan y su arquitectura varían de manera significativa, como se verá a lo largo de este tema [1-7].

La plataforma j2ee.

J2EE comprende un conjunto de APIs que amplían y mejoran el modelo J2SE (Java 2 Standar Edition), orientado su funcionalidad fundamentalmente hacia aplicaciones distribuidas. Pero además J2EE proporciona una infraestructura de entorno de ejecución para albergar y administrar aplicaciones, típicamente hablamos del servidor donde se ejecutan las aplicaciones, la especificación no dice como debe estar construida la infraestructura de ejecución, en su lugar define unos roles e interfaces que separan claramente la infraestructura y las aplicaciones que se ejecutan en ella. Define el concepto de contenedor y especifica lo que llama un contrato entre el contenedor y las aplicaciones. En la especificación entra a jugar parte de la gran mayoría de las empresas relevantes en las tecnologías de la información, la especificación ha pasado de ser propiedad de uno solo a ser un conjunto de normas y "Best Practices" proporcionadas por todas las industrias del sector, aportando su conocimiento y manera de trabajo. Esto ha permitido que distintas empresas jueguen con un conjunto de reglas comunes, ya que la especificación no dicta como debe construirse la infraestructura, los distintos fabricantes personalizan con su mejor conocimiento las herramientas que dan soporte a la especificación.

A continuación, se muestran las APIs que entran a formar parte de la especificación:

1. JDBC 3.0: API que permite de manera estándar la conexión con distintas fuentes de datos. Proporcionando un pool de conexiones.
2. EJB 3.0: Componentes distribuidos y robustos, de lógica de negocio.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

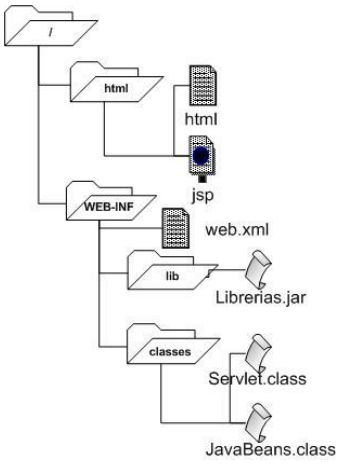
3. Java Servlets 2.4: Abstracción OO para construir aplicaciones Web.
4. JSP 2.0: Extensión de la anterior basada en plantillas.
5. JMS 1.0.1: Mensajes asíncronos. (Message Oriented Middleware).
6. JTA 1.0: Transacciones distribuidas.
7. JavaMail 1.2: Proporciona toda la funcionalidad para emisión y recepción de datos mediante correo electrónico. Soporte a IMAP, SMTP.
8. Java XML Pack: Conjunto de librerías para el tratamiento de la información en formato XML, este pack contiene:
    a. Java API for XML Processing (JAXP): procesa documentos usando varios parsers.
    b. Java Architecture for XML Binding (JAXB): Procesa documentos XML usando un Schema como resultado de componentes JavaBeans.
    c. Java API for XML-based RPC (JAX-RPC) – Permite enviar llamadas a métodos de componentes remotos usando SOAP sobre Internet y recibir los resultados.
    d. Java API for XML Messaging (JAXM) – Permite enviar mensajes SOAP sobre Internet de una manera estándar.
    e. Java API for XML Registries (JAXR) – Proporciona una manera estándar de acceder a registros de negocio para compartir información.
9. Java IDL: Proporciona la infraestructura para la integración con servicios CORBA.
10. RMI-IIOP: Permite la comunicación remota de objetos, y es un API base para otras tecnologías.
11. JNDI: Permite la conexión a servicios de nombres de una forma estándar.
12. JCA 1.0: Conectores para la integración con Sistemas de Información (EIS).

El contenedor.

La especificación J2EE propone un entorno bajo el cual se asegura un comportamiento de los componentes de aplicación igual para distintos entornos, el modelo del contenedor contiene recursos necesarios para establecer mecanismos de seguridad, transacciones, gestión de memoria, concurrencia, sincronización de tareas y conexiones con otros sistemas de manera homogénea para todas los aplicaciones, independientemente de la plataforma donde sean ejecutadas [8-11].
El lenguaje Java, proporciona los medios necesarios para que una misma aplicación pueda ejecutarse en distintas plataformas, pero una aplicación tiene una serie de requisitos que no inciden directamente en la funcionalidad pero que deben ser contemplados, como por ejemplo la seguridad, la gestión de memoria, y la configuración del acceso a los datos. Además una aplicación contiene una serie de recursos que pueden ser exclusivos de la aplicación en si, como conexiones con base de datos, servi-

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

cios de nombres, archivos de configuración, etcétera. Resulta difícil ejecutar una aplicación en distintas plataformas si tiene integrado en su código la gestión de todos estos recursos, sin tener que adaptar esa aplicación a la nueva plataforma y sin recompilar la aplicación de nuevo.
Por ello, la especificación J2EE proporciona el modelo del contenedor, de manera que cualquier aplicación que siga las normas que dicta la especificación pueda ser desplegada en cualquier contenedor que cumpla también la especificación, independientemente de la plataforma. La especificación expone las normas que deben cumplir los proyectos J2EE y el contenedor, esto es necesario porque el contenedor debe tener acceso a todos los recursos del proyecto para poder gestionarlos. Este modelo permite que el desarrollo se centre en el aspecto funcional, dejando las tareas de bajo nivel gestionadas por el contenedor, se produce un modelo de programación declarativa. La unión entre el contenedor y la aplicación se realiza mediante el llamado contrato, que es un fichero de configuración dependiente del proyecto, denominado Deployment Descriptor y cuyo formato es XML [12-15].
Aún queda un asunto por resolver, ya que para que los proyectos puedan ser distribuidos entre distintos contenedores deben crearse manteniendo una estructura común de directorios y directivas de configuración, para que de esta manera independientemente de la forma de trabajo del equipo de desarrollo, cualquier contenedor pueda acceder y gestionar los recursos. La estructura de un proyecto web en la especificación J2EE se expone a continuación:



53.Figura 15. Estructura de proyecto web.

Como se aprecia en la figura todo proyecto web contiene un directorio raíz del cual dependen una serie de directorios, inicialmente el contenido estático como páginas html, imágenes o multimedia dependen del directorio raíz o en su defecto de uno creado a tal fin, los documentos jsp también se incorporan aquí. A continuación y de manera obligatoria debe existir un directorio llamado WEB-INF donde se gestionarán todos los recursos dinámicos del proyecto, aquí se insertará también el descriptor de despliegue (Deployment Descriptor) que configura todos los componentes del proyecto, debajo contendrá obligatoriamente dos directorios llamados lib y classes el primero almacenará las librerías externas que usemos en el proyecto, y en el segundo se encontrarán los componentes desarrollados como servlets, javabeans y librerías de etiquetas. Con esta estructura homogénea la especificación se asegura de que un proyecto pueda ser desplegado en distintos contenedores y estos sepan tratar la misma información por igual.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Para poder distribuir de manera homogénea una aplicación, la especificación J2EE propone tres tipos de empaquetamiento de las clases y recursos que forman parte de la aplicación.

1. Ficheros JAR (Java Archive)
2. Ficheros WAR (Web Archive)
3. Ficheros EAR (Enterprise Archive)

El primero de ellos se usa para almacenar componentes que se reutilizarán en las aplicaciones, típicamente clases, ficheros de configuración y otros recursos, en una aplicación J2EE este tipo de ficheros se usa para la generación de Enterprise JavaBeans (EJB). Los archivos WAR, permiten empaquetar todos los recursos y estructura de directorios visto anteriormente, se usan para la creación y distribución de proyectos web, por último el tipo EAR permite empaquetar una aplicación mucho más robusta que haga uso de componentes web y EJBs juntos, de esta manera un EAR permite juntar en un único archivo ficheros WAR y ficheros JAR [16-23].

El descriptor de despliegue.

La especificación J2EE proporciona un modelo de programación declarativa, permitiendo al programador centrarse en desarrollar las funcionalidades de la aplicación y dejando tareas de bajo nivel como puede ser la gestión de la memoria, conexión con repositorios de datos, sincronización, concurrencia, transacciones, multi-threading, seguridad, en manos del contenedor.

La especificación nombra como contrato a la relación que existe entre los distintos componentes que forman parte de una aplicación J2EE y el contenedor que la soporta. Esta relación puede venir en base a alguna de las tareas de bajo nivel expuestas anteriormente, o bien, como la especificación dicta que un contenedor web debe dar soporte completo al protocolo http, el contenedor web puede por lo tanto funcionar como un servidor web aunque no sea este su cometido, por lo que esta relación puede venir en forma de alias, mapping de directorios virtuales, paso de parámetros de configuración, etc.

Este contrato viene definido mediante un documento en formato XML, denominado Deployment Descriptor o descriptor de despliegue, básicamente la tarea de configurar el proyecto mediante el descriptor será realizada o bien por el programador encargado del proyecto o por un perfil administrador, también denominado “deployer” que normalmente mediante una consola de administración del servidor de aplicaciones podrá configurar las directivas necesarias para el proyecto, aunque también es posible generar estas directivas de manera manual editando el propio descriptor. Ya que el descriptor es un documento XML debe cumplir con las normas de este lenguaje de marcado y además contiene un DTD que especifica la gramática que soporta. Todo descriptor de despliegue de un proyecto web comienza con un nodo raíz denominado <web-app>, bajo el cual se irán indicando las distintas etiquetas necesarias para configurar el proyecto, por ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
        <servlet>
                <servlet-name>Manager</servlet-name>
                <servlet-class>org.apache.catalina.servlets.ManagerServlet</servlet-class>
                <init-param>
                        <param-name>debug</param-name>
                        <param-value>2</param-value>
                </init-param>
        </servlet>
        <servlet-mapping>
                <servlet-name>Manager</servlet-name>
                <url-pattern>/*</url-pattern>
        </servlet-mapping>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        <security-constraint>
                <web-resource-collection>
                        <web-resource-name>Entire Application</web-resource-name>
                        <url-pattern>/*</url-pattern>
                </web-resource-collection>
                <auth-constraint>
                        <role-name>manager</role-name>
                </auth-constraint>
        </security-constraint>
        <login-config>
                <auth-method>BASIC</auth-method>
                <realm-name>Tomcat Manager Application</realm-name>
        </login-config>
</web-app>
```

En este ejemplo de un descriptor web.xml, se aprecia como en esta aplicación web exite un servlet especificado por su paquete de clases dentro de la etiqueta <servlet-class> al cual se establece un alias denominándolo Manager, además se establecen unos parámetros de inicio con las estiquetas <init-param> y <param-value>. Con la etiqueta <servlet-mapping> se establece una unión en las que todas las llamadas que se realicen a la URL raíz dentro de ese servidor (indicado por /*), se pasen automáticamente al servlet Manager. A continuación se configura la seguridad en el proyecto, estableciendo un tipo de autenticación básica con un reino de seguridad denominado Tomcat Manager Application y que sólo estará disponible para el rol de manager, esta seguridad se aplica a toda la aplicación nuevamente determinando a todas las peticiones que se realicen a la URL raíz [24-27].

La especificación servlet.

En las arquitecturas web tradicionales la necesidad de recibir y manipular información proveniente del usuario fomentó la aparición de el modelo CGI, el cual permitía usar las características de comunicación del protocolo http para recibir la información del usuario mediante peticiones y parámetros, realizar la ejecución de un programa con su lógica de negocio en el servidor, y después hacer las operaciones para provocar una respuesta enviada de vuelta al cliente. Inicialmente los lenguajes soportados para la creación de programas que realizaran estas tareas y se ejecutarán en el servidor fueron lenguajes como Perl, C o C++. La necesidad de manipular la información del usuario y disponer de contenido dinámico hizo que la plataforma Java pasara de ejecutarse en las plataformas de los clientes mediante los denominados applets a copiar el modelo CGI y ejecutarse en el servidor mediante nuevo modelo de componentes denominados Servlets.

Los Servlets copian el modelo CGI proporcionando una infraestructura para el trabajo mediante el protocolo http, proporcionando un API completo de desarrollo, y la especificación J2EE proporciona el modelo de contenedor, mediante el cual asegura la portabilidad de los proyectos web. Los Servlets proporcionan mejoras sobre el modelo de CGI clásico, pero también heredan sus carencias, como por ejemplo: la unión en un mismo programa de lógica de negocio, y lógica de presentación haciendo las aplicaciones menos escalables y mantenibles. En cuanto a las mejoras, proporcionan un sistema de seguridad diferente al de CGI clásico, el cual estaba basado en permisos de ejecución sobre directorios, algo muy susceptible a la incorporación de código malicioso, los Servlets no disponen su seguridad en base a permisos de ejecución, sino que se ejecutarán bajo una infraestructura de seguridad de la máquina virtual. Otra ventaja es que en el modelo CGI clásico se generaba un nuevo proceso por cada petición de un cliente, mientras que en el modelo Servlets se genera un único proceso y se sirven múltiples instancias, una por cada cliente [28-30].

Un servlet puede considerarse como una especie de mini-servidor especializado, escrito en Java, que se encarga de recibir la petición, generar contenido dinámicamente y, finalmente, devolver la respuesta. Los servlets se ejecutan dentro de un contenedor web (típicamente conocido como motor de servlets) que se encarga de traducir las peticiones nativas del protocolo a objetos Java y las respuestas en forma de objetos Java a respuestas nativas del protocolo.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

La especificación Servlet define un ciclo de vida de los componentes, así de esta manera se asegura de que todos los componentes se escriban y se ejecuten de la misma forma, independientemente del contenedor que les dé soporte. El contenedor se encarga de gestionar el ciclo de vida del servlet, así como de proporcionarle los servicios que necesite. El servidor redirige las peticiones al contenedor para que éste, a su vez, las delegue en el servlet correspondiente. En el caso del servidor web, este comportamiento se configura nativamente y, en el caso del contenedor, mediante un mecanismo estándar definido en la especificación Servlet.

Según la especificación Servlet el contenedor web debe implementar las funciones del servidor web, sin embargo, este efecto se consigue habitualmente conectando un servidor web con el contenedor, por ejemplo Apache + JBoss. Por tanto puede considerarse al contenedor web como un servidor web que debe llevar a cabo una serie de tareas tales como: proporcionar servicios de red para el establecimiento de peticiones y respuestas mediante http, codificar y decodificar peticiones y respuestas en formato MIME, configurar los servlets en función de los descriptores de despliegue y gestionar el ciclo de vida de los servlets. Además hay otras características de los contenedores que la especificación no exige pero que recomienda, en la mayoría de los casos estas características se pueden dar por presentes, como dar soporte al protocolo HTTP 1.1, soporte a HTTPS, y r estricciones de seguridad para la ejecución de los servlets, por ejemplo acceso al sistema de ficheros, o limitaciones en la creación de hilos.

El mecanismo de gestión del ciclo de vida de los servlets se basa en un contrato definido entre el contenedor y el servlet, es decir mediante una interfaz. Los servlets deben implementar la interfaz javax.servlet.Servlet para que el contenedor pueda comunicarse con ellos, el contenedor instancia e invoca el servlet por medio del API de reflexión. De esta manera el contenedor conoce y se asegura de que el componente que está instanciando es un Servlet y no otro. Aunque, cuando se habla de Servlets, se supone siempre una relación muy estrecha con la web, en realidad, el API servlet es independiente del protocolo, aunque la mayoría de implementaciones son para el protocolo HTTP, sería fácil implementar un Servlet FTP. El que un Servlet pueda ser independiente del protocolo es gracias a que la interfaz Servlet es muy genérica, de hecho el API proporciona una clase GenericServlet que dota de toda la funcionalidad para establecer peticiones y respuestas sobre cualquier protocolo, así mismo existe una clase HttpServlet que proporciona las mismas funcionalidades pero sobre protocolo http [31-25].

El ciclo de vida de un servlet viene definido en base a los métodos que proporciona el interfaz javax.servlet.Servlet, estos métodos son:

1 public void init(ServletConfig config) throws ServletException.

2 public void service(ServletRequest req, ServletResponse res) throws IOException, ServletException

3 public void destroy()

Con el método init(), la especificación se asegura de que el contenedor no llamará a ningún otro método antes de que este haya terminado, de esta manera sólo existirá un único proceso. Este método sólo es llamado una vez, normalmente bajo la primera petición del primer cliente, o bien precargando el servlet desde la consola de administración del servidor, o bien configurando este comportamiento en el descriptor de despliegue. El método init() típicamente es usado para la configuración de la aplicación, abrir recursos que debe de estar disponibles para el resto de la aplicación, como conexiones con bases de datos, acceso a un repositorio o sistema de ficheros, leer un archivo de configuración,

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

etc. El método service() lo invoca el contenedor de manera concurrente cada vez que recibe una petición para ese servlet. Antes de invocar a service, el contenedor construye dos objetos: Uno de tipo ServletRequest, que encapsula la petición, especialmente los parámetros. Otro de tipo ServletResponse, que encapsula la respuesta. Estos dos objetos son de tipo genérico y pueden establecer las peticiones y respuestas de cualquier protocolo, en el caso de http el API proporciona los métodos necesarios para establecer las cabeceras, códigos de estado y métodos de conexión adecuados. Por último el método destroy() es el último método que llamará el contenedor antes de destruir el servlet, las tareas que se realizan en este método son de liberación de recursos que ha usado el servlet, normalmente cerrará y limpiará aquellos recursos que se han abierto o usado en el init, aunque también en el service (conexiones con bases de datos, ficheros abierto), además de esta manera el contenedor se asegura de que no intentará destruir el servlet antes de que el método desrtoy haya terminado [36-39].

Conceptos básicos: Peticiones, Respuestas y Contexto.
La especificación Servlet define cuatro objetos básicos que son manejados por cualquier servlet:

Concepto	**Clase General**	**Clase HTTP**
Petición	javax.servlet.ServletRequest	javax.servlet.http.HttpServletRequest
Respuesta	javax.servlet.ServletResponse	javax.servlet.http.HttpServletResponse
Configuración	javax.servlet.ServletConfig	javax.servlet.ServletConfig
Contexto	javax.servlet.ServletContext	javax.servlet.ServletContext

Tanto el contexto, como la configuración del servlet son objetos independientes del protocolo, de ahí que no exista una implementación específica para http. El contexto define toda la configuración para una aplicación entera con todos sus componentes, mientras que la configuración define los parámetros de configuración para cada componente de manera individual. El contexto es el punto de contacto de los servlets con el sistema de archivos, ya que es el contenedor el que conoce el mapeo de directorios virtuales a directorios físicos. Así, el contexto proporciona métodos para recuperar recursos como para recuperar su tipo MIME:

1. public String getRealPath(String path)
2. public InputStream getResourceAsStream(String path)
3. public URL getResource(String path) throws MalformedURLException
4. public Set getResourcePaths(String path)
5. public String getMimeType(String file)

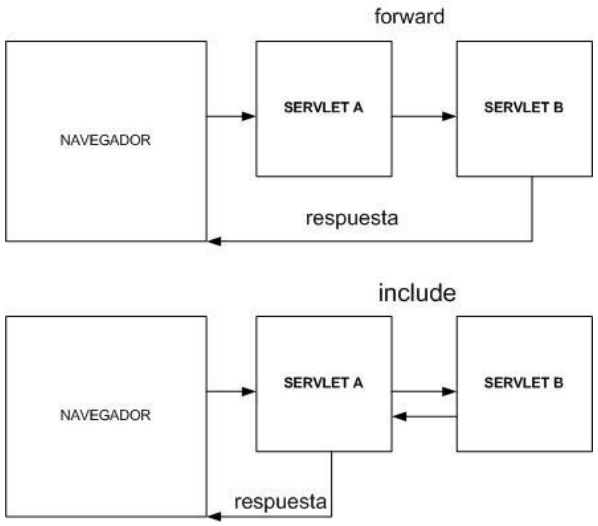
ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Las peticiones se encapsulan en el interfaz javax.servlet.ServletRequest, y ya que un servlet no tiene que estar ligado al protocolo http, y es independiente del protocolo usado, se pueden crear subinterfaces para protocolos específicos, siendo esto trabajo del proveedor del contenedor. Las peticiones encapsulan información sobre:

6. Parámetros de la petición.
7. Atributos.
8. Internacionalización.
9. El cuerpo de la petición.
10. El protocolo de la petición.
11. El servidor.
12. El cliente.
13. Redirección de la petición (Dispatchers).

La redirección de la petición puede venir por dos mecanismos, en ambos la petición original es pasada a otro componente y modificada para ser complementada y de esta manera producir la respuesta al cliente. Los dos mecanismos son forward o include. La diferencia entre uno u otro estriba en quién puede hacer la modificación de la petición y sobretodo quién envía la respuesta al cliente, tal y como se muestra en la siguiente figura:



54.Figura 16. Redirección de la petición.

Para dar soporte HTTP al API Servlet, éste incluye una serie de interfaces y clases, así, las peticiones HTTP están encapsuladas por la interfaz javax.servlet.http.HttpServletRequest que es subinterfaz de javax.servlet.ServletRequest. Este interfaz define métodos clasificables en los siguientes grupos:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

14. Seguridad
15. Cabeceras HTTP
16. Información sobre URL's
17. Métodos HTTP
18. Gestión de la sesión

Los parámetros de la petición se reciben desde el cliente en forma de cadenas de texto y son los que, habitualmente, dan información para la generación del contenido de la respuesta.
Los métodos de gestión de parámetros de la clase javax.servlet.ServletRequest son:

19. public String getParameter(String name) Devuelve el valor de un parámetro en función de su clave.
20. public Enumeration getParameterNames() Devuelve todos los nombres de los parámetros.
21. public String[] getParameterValues(String name) Devuelve todos los valores de un parámetro compuesto.
22. public Map getParameterMap()Devuelve un mapa con los pares clave/valor.

En el caso de las redirecciones, la misma petición puede pasar por varios servlets o JSPs, y el API nos permite modificar la petición enviada por el cliente añadiéndole una serie de atributos. Debe quedar claro que, mientras los parámetros llegan desde el cliente, los atributos se añaden durante la gestión de la petición, es decir, dentro del contenedor. Los métodos proporcionados son:

23. public Object getAttribute(String name) Devuelve el valor de un atributo en función de su clave.
24. public Enumeration getAttributeNames()Devuelve todos los nombres de los atributos.
25. public void setAttribute(String name, Object o) Añade un atributo a la petición.
26. public void removeAttribute(String name) Elimina un atributo de la petición.

Un mensaje de petición http, contiene un método de conexión y una serie de cabeceras que complementan al cuerpo de la petición. En algunos casos las peticiones pueden incluir más información de las que puden incluir las cabeceras, así algunas peticiones incluyen un cuerpo, por ejemplo haciendo uso del método de conexión PUT o POST. El API proporciona una serie de métodos para gestionar la información del cuerpo de la petición:

27. public String getCharacterEncoding()Devuelve el juego de caracteres del cuerpo de la petición.
28. public void setCharacterEncoding(String env) throws UnsupportedEncodingException Sobreescribe el juego de caracteres del cuerpo de la petición.
29. public int getContentLength()Devuelve la longitud total del cuerpo de la petición.
30. public String getContentType()Devuelve el tipo MIME del cuerpo de la petición.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

31. public ServletInputStream getInputStream() throws IOException Devuelve un InputStream para leer el cuerpo de la petición (binario).
32. public BufferedReader getReader() throws IOException Devuelve un Reader para leer el cuerpo de la petición (texto).

Para realizar la redirección de la petición se hace uso de los métodos forward(Request, Response) o include(Request, Response) (delegación/composición) que el API servlet proporciona un mecanismo basado en la interfaz RequestDispatcher, y desde la interfaz ServletRequest se puede recuperar un RequestDispatcher usando el método:
public RequestDispatcher getRequestDispatcher(String path).
Por ejemplo:

33. getServletContext().getRequestDispatcher("/pages/showBalance.jsp").forward(req,res).
34. getServletContext().getRequestDispatcher("/pages/navigation_bar.html").include(req,res);

Para dar soporte HTTP al API Servlet, éste incluye una serie de interfaces y clases, así, las peticiones HTTP están encapsuladas por la interfaz javax.servlet.http.HttpServletRequest (subinterfaz de javax.servlet.ServletRequest).Este interfaz define métodos clasificables en los siguientes grupos:

35. Seguridad
36. Cabeceras HTTP
37. Información sobre URL's
38. Métodos HTTP
39. Gestión de la sesión

El contenedor web proporciona mecanismos de gestión de la seguridad (autenticación y autorización) declarativos (mediante los descriptores de despliegue), pero es posible utilizar seguridad programática, para ello la interfaz HttpServletRequest declara los siguientes métodos:

40. public String getAuthType()
41. public String getRemoteUser()
42. public boolean isUserInRole(String role)
43. public Principal getUserPrincipal()

Así mismo se definen métodos para recuperar la metainformación de las cabeceras, como por ejemplo:

44. public long getDateHeader(String name)
45. public String getHeader(String name)
46. public Enumeration getHeaders(String name)
47. public Enumeration getHeaderNames()
48. public int getIntHeader(String name)

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Los servlets generan la respuesta a partir de un objeto que implementa la interfaz javax.servlet.ServletResponse que, básicamente, encapsula un OutputStream, sobre el que escribe el servlet. Los objetos ServletResponse son instanciados por el contenedor y pasados al servlet. El OutputStream del que depende esta interfaz se implementa usando un mecanismo de buffer intermedio, así, la interfaz proporciona métodos para controlar este buffer.
El cuerpo de la respuesta se escribe por medio de un stream que se apoya en un buffer. En el caso de ir a generar contenido dinámico se usará un javax.servlet.ServletOutputStream y en el caso de texto plano se usará un PrintWriter. Los métodos para recuperar, manejar y recuperar información del flujo de escritura son:

49. public ServletOutputStream getOutputStream() throws IOException

50. public PrintWriter getWriter() throws IOException

Estos dos métodos son mutuamente excluyentes, sí se ha invocado alguno de ellos una invocación sobre el otro lanzará IllegalStateException.

51. public void reset(). Vacía el stream, tanto el cuerpo como las cabeceras.

52. public boolean isCommitted(). Devuelve true si el buffer ha sido volcado (lo que implica que se han escrito las cabeceras y el cuerpo).

53. public void setBufferSize(int size) Establece el tamaño del buffer (el tamaño por defecto depende de la implementación).

54. public int getBufferSize()Devuelve el tamaño del buffer (o cero si no se está usando buffer).

55. public void flushBuffer() throws IOException Vuelca el buffer, es decir, escribe la respuesta (cuerpo y cabeceras), cualquier llamada a isCommited a partir de la invocación de este método devolverá true.

56. public void resetBuffer()Vacía el buffer, es decir, vacía el cuerpo, pero no las cabeceras.

El API sobrescribe el método service de la clase genérica, adaptándolo a los diferentes métodos de conexión en el protocolo http, así en lugar de disponer un único método service, existe un método por cada método de conexión siendo doGet, doPost, doPut, doDelete, doTrace, doOptions y doHead.

Listeners

La especificación Servlet añade una serie de escuchadores de eventos (siguiendo el modelo de eventos de Java) para dar la posibilidad al programador de reaccionar ante cambios de estado de la aplicación. Todos estos listeners siguen el mismo esquema (a excepción de HttpSessionBindingListener que fue incluido en la especificación 2.2 y tiene un comportamiento diferente).
Estos eventos pueden ser usados para conseguir persistencia (serializando objetos) y validación de estados.
Tales eventos de la aplicación necesitan ser declarados en el Deployment Descriptor, a continuación se presenta un resumen de los posibles escuchadores de eventos sobre los eventos desencadenados:
Eventos del contexto:

- ServletContextListener - ServletContextEvent

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- ServletContextAttributeListener – ServletContextAttributeEvent

Eventos de la Sesión:

- HttpSessionListener - HttpSessionEvent
- HttpSessionActivationListener - HttpSessionEvent
- HttpSessionAttributeListener - HttpSessionBindingEvent
- HttpSessionBindingListener – HttpSessionBindingEvent

Por ejemplo, Todos los escuchadores se declaran en el Deployment Descriptor mediante la siguiente estructura:

<listener><listener-class></listener-class></listener>

Es el contenedor, por medio de reflexión, el encargado de distinguir la clase del escuchador, es decir, no se indica en ningún momento qué tipo de escuchador se está registrando.
Una vez declaradas, podríamos utilizar por ejemplo la interfaz HttpSessionListener para trabajar con sesiones, Las implementaciones de esta interfaz reciben notificación de cambios en la lista de sesiones activas de la aplicación.
Los métodos por los que recibe notificación son:

- public void sessionCreated(HttpSessionEvent e) -> Se ha creado una nueva sesión.

- public void sessionDestroyed(HttpSessionEvent e) -> Se ha eliminado una sesión.

Y así podríamos utilizar esta información para cerrar recursos abiertos (por ejemplo conexiones con una base de datos, o ficheros para que no queden en un estado corrupto) cuando un usuario ha salido del sistema y ha cerrado su sesión.

Filtros

En versiones anteriores de la especificación se utilizaban métodos (ya obsoletos), para comunicar servlets entre sí, a partir de la especificación Servlet 2.3 se añadió a la especificación el mecanismo de filtros que complementa el mecanismo de dispatchers.
Permite el encadenamiento de filtros para la distribución de las tareas, así, un filtros validará, otro cifrará, y al final de la cadena, un servlet procesará, en lugar de tener un servlet monolítico que realice todas estas tareas.
Con los filtros el programador puede, por primera vez, interceptar el flujo de las peticiones en el contenedor de una manera portátil y estándar.
Los filtros capturan la petición y pueden cambiar el flujo de ésta (decidir a qué servlet dirigirla, rechazarla, encadenar una serie de servlets para procesar la petición secuencialmente, etc…).
Entre sus aplicaciones se incluyen:

- Autenticación y autorización, tanto en el contenedor como en recursos externos (sistemas legados).
- Sistemas de registro de eventos (log) y auditoria.
- Implementación de cachés de contenido generado.
- Control parametrizado de acceso (p.e. dependiendo de la hora/fecha).

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Transformaciones de la respuesta:
- Cifrado.
- Compresión.
- Transformaciones XSL/T.

Un filtro es una clase que implementa la interfaz javax.servlet.Filter que define los siguientes métodos:

- public void init(FilterConfig config) throws ServletException

Llamado cuando el contenedor inicializa el filtro.

- public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException

Llamado cuando el contenedor necesita que se ejecute el filtro.

- public void destroy()

Llamado cuando el contenedor va a descargar el filtro.
Un ejemplo:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Auditor implements Filter {
        private ServletContext ctx;
        public void init(FilterConfig fg) {ctx = fg.getServletContext();}
        public void doFilter(ServletRequest rq, ServletResponse rs, FilterChain
fc) throws ServletException, IOException {
                ctx.log("Conexión desde " + rq.getRemoteAddr());
                fc.doFilter(rq, rs);
        }
        public void destroy() {}
}
```

Como se puede observar, el filtro realizará una redirección mediante un dispatcher interno, pero en ningún momento se dice a qué clase hay que enviar la información en esta redirección. Esto se hace mediante el descriptor de despliegue, lo que facilita interconectar filtros sin modificar el código y por tanto recompilar como si lo hiciéramos con el RequestDispatcher tradicional.
La información en el descriptor de despliegue sería como la siguente:

```
<filter>
        <filter-name>Nombre del Filtro</filter-name>
        <filter-class>nombre.de.la.Clase</filter-class>
        <init-param> <!-- Opcional -->
          <param-name>nombre del parámetro</param-name>
                <param-value>valor del parámetro</param-value>
        </init-param>
</filter>

<filter-mapping>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        <filter-name>Nombre del Filtro</filter-name>
        <url-mapping>/*</url-mapping>
</filter-mapping>
```

La especificación JSP.

Uno de los problemas heredados por los Servlets del modelo CGI clásico es que no existe separación de responsabilidades, un mismo componente contiene lógica de presentación y lógica de negocio, mezclando distintos lenguajes y tecnologías. Este hecho no permite la escalabilidad de las aplicaciones, ni la reutilización del código, obligando a recompilar los componentes de negocio por tareas, a veces sencillas, de modificación del interfaz, el mantenimiento de la aplicación puede llegar a ser un verdadero caos, siendo difícil aumentar las funcionalidades.
Bajo estas premisas, surge la especificación JSP, amparada por el éxito del modelo ASP de Microsoft viene a resolver estas carencias. Un JSP es un componente Java específicamente diseñado para la generación de la información al cliente de manera dinámica [40-42].

Un JSP es un documento de texto, normalmente escrito en código a html o XML con una serie de etiquetas dinámicas que permiten la ejecución de código Java, como documento de texto es fácilmente editable con cualquier editor de textos simple, esto permite una mayor sencillez a la hora de modificar el interfaz de usuario. El fichero debe tener la extensión .jsp, y dentro de un proyecto se instalarán como si fueran documentos html normales. Los JSP nos permiten eliminar el código html de los Servlets, separando la lógica de control de la vista, por tanto los Servlets quedarán como componentes de control. La parte estática de un JSP vendrá definida por las etiquetas propias del lenguaje (ya sea html o XML), y la parte dinámica puede venir definida por dos tipos de etiquetas, por un lado el formato clásico y por otra mediante un formato XML.

1. <% %>: formato clásico
2. <jsp: > </jsp>: formato XML

```
<HTML>
        <HEAD>
                <TITLE>Ejemplo Hola mundo  en una JSP</TITLE>
        </HEAD>
          <BODY>
            <% out.println("Hola mundo desde una JSP"); %>
          </BODY>
</HTML>
```

55.Figura 17. JSP de Ejemplo.

Por tanto, los JSP son vistos como plantillas que necesitan de una interpretación en el servidor para producir la respuesta dinámica. Los JSP sufren una compilación en el servidor, el contenedor detectará que es un tipo de componente, y necesita ser compilado antes de devolver la respuesta al cliente. La compilación que sufre un JSP, internamente es a un Servlet, por lo tanto comparten el mismo API, los mismos objetos petición y respuesta, también dispondrá de un objeto contexto y un objeto de configuración, y también tienen un ciclo de vida al igual que los Servlets manejando un método inició, un método de servicio, y un método de destrucción. Todo esto permite al contenedor configurar los proyectos web por igual, y los componentes pueden trabajar juntos compartiendo datos mediante

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

sesiones, o redireccionando la petición. El JSP contiene una serie de elementos para su trabajo, como son:

1. Expresiones de la forma **<%= expresión %>** que son evaluadas e insertadas en la salida (equivalentes a un out.println).
2. Scriptlets de la forma **<% codido %>** que se insertan dentro del método service() del servlet, usadas para la creación de algoritmos.
3. Declaraciones de la forma **<%! codigo %>** que se insertan en el cuerpo de la clase del servlet, y permiten la declaración de variables y métodos.
4. Directivas de la forma **<%@ directive atributo1="valor1"... atributoN="valorN" %>** que afectan a la estructura general del servlet generado, como por ejemplo importar librerías, establecer el Content-Type, establecer el tamaño y comportamiento del buffer de salida, etc.

En el siguiente ejemplo se muestra el uso de las etiquetas en un JSP, en concreto se usan scriptles y expresiones y se muestra como es posible usar los mismos objetos proporcionados por el API java como es el objeto Date, y recuperar parámetros proporcionados por el cliente mediante el objeto HttpServletRequest, además se muestra como es posible escribir en la salida estándar mediante el método println de igual forma que hace una expresión.

```
<HTML>
<HEAD>
<TITLE>Ejemplo de JSP</TITLE>
</HEAD>
<% String bgColor = request.getParameter("bgColor");
            boolean hasExplicitColor;
            if (bgColor != null) {
              hasExplicitColor = true;
            } else {
              hasExplicitColor = false;
              bgColor = "WHITE";
} %>
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">El  color de Fondo es: <%= bgcolor %> </H2>
<H2>Ejemplos de expresiones JSP</H2>
<UL>
  <LI>Hora actual en el servidor: <%= new java.util.Date() %>
  <LI>Nombre del host: <%= request.getRemoteHost() %>
   <LI>Valor del parametro testParam:   <% out.println( request.getParameter("testParam") );%>
</UL>
</BODY>
</HTML>
```

56.Figura 18. Ejemplo de uso de etiquetas JSP.

Sin embargo en este ejemplo se hace uso del objeto request y del objeto out sin ningún tipo de declaración previa, esto es debido a que la especificación JSP adopta una serie de objetos implícitos que permiten el trabajo de la misma forma que se hacia con los servlets, como son:

1. request.
2. response.
3. out.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

4. session.
5. application.
6. config.
7. pageContext.
8. page.
9. exception

El uso de JSP es útil para no mezclar lógica de negocio y lógica de presentación, sin embargo es muy fácil ligar de manera dura a un JSP con la parte de negocio, debido a que comparte el mismo API que un Servlet, el hecho de que se compile como tal y funcione de la misma forma, permite a un JSP hacer exactamente las mismas tareas que un Servlet. Sin embargo, no hay que caer en esa tentación, el JSP debería estar destinado al interfaz de usuario, no a la lógica de negocio, ni enlazarse directamente con los datos. Para desacoplar a los JSP del resto de capas de la aplicación, existen una serie de componentes intermedios que fueron mencionados al principio del documento, cuando se hacía una evolución de las arquitecturas web, estos componentes albergan en su interior lógica de negocio y acceso a datos, lo que les permite ser reutilizables, de esta manera la aplicación es más mantenible y escalable. Típicamente estos componentes son JavaBeans, Taglisbs y EJBs [43-45].

JavaBeans y JSP

La acción <jsp:useBean> nos permite localizar o instanciar un JavaBean en la página JSP. La sintaxis más simple para especificar que se debería usar un Bean es:

<jsp:useBean id="nombre" class="paquete.clase" />

Con esto conseguimos localizar o instanciar un objeto, de la clase especificada por "class", y enlazarlo con una variable, con el nombre especificado en "id".

También se puede especificar un atributo "scope" que hace que el bean se asocie con más de una página. En este caso, es útil obtener referencias a los beans existentes, y la acción jsp:useBean especifica que se instanciará un nuevo objeto si no existe uno con el mismo nombre y ámbito.

Como un JavaBean es una clase cuyos métodos son codificados como propiedades (get, set), esta forma de trabajo nos permite acceder a dichas propiedades mediante setProperty y getProperty.

Usamos **jsp:setProperty** para establecer los valores de las propiedades de los beans que se han referenciado anteriormente con la acción jsp:useBean. Además valor de id en jsp:useBean debe coincidir con el valor de name en jsp:setProperty.

La acción **jsp:getProperty** obtiene el valor de una propiedad de un bean, usando el método getter del bean, e inserta su valor en la respuesta.

Antes de usar esta acción debe aparecer una acción jsp:useBean para instanciar o localizar el bean.

Librerías de etiquetas: Taglibs

Gracias a la directiva taglib podemos definir nuestros propios tags JSP.

Para definir un tag necesitamos definir 4 componentes:

- Una clase java que defina el comportamiento del tag.
- Un fichero TLD (Tag library descriptor) para hacer visible la clase en el servidor.
- Un fichero web.xml para hacer visible el tag en el servidor.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Un fichero JSP que use el tag

La sintaxis completa de la directiva taglib es:

<%@ taglib uri="URIForLibrary" prefix="tagPrefix" %>

El atributo uri define donde se localiza el fichero TLD, y puede ser un URL, un URN o un PATH absoluto o relativo. Si la URI es una URL, entonces el TLD es localizado por medio del mapping definido dentro del fichero web.xml. Si la URI es un path, entonces es interpretado como un acceso relativo a la raíz de la aplicación y debería resolverse en un fichero TLD directamente, o un fichero .jar que contiene el fichero TLD.

De esta forma podemos tener un conjunto de librerías de etiquetas muy útiles para diseñadores de interfaz de usuario que necesiten realizar operaciones complejas (como extraer datos de una base de datos) sin conocimientos en java, o bien para personal externo a nuestras aplicaciones que necesiten llevar a cabo dichas operaciones, pero no queremos que conozcan nada de nuestros sistemas. Así nos aseguramos de que todo el mundo utilizará los accesos a las bases de datos sin incorporar código propietario, por ejemplo.

Como conclusión las librerías de etiquetas facilitan el desarrollo de páginas jsp reduciendo su complejidad y número de líneas de código y facilitan la seguridad, la reutilización y el mantenimiento.

A continuación, se muestra un ejemplo de uso de una taglib:

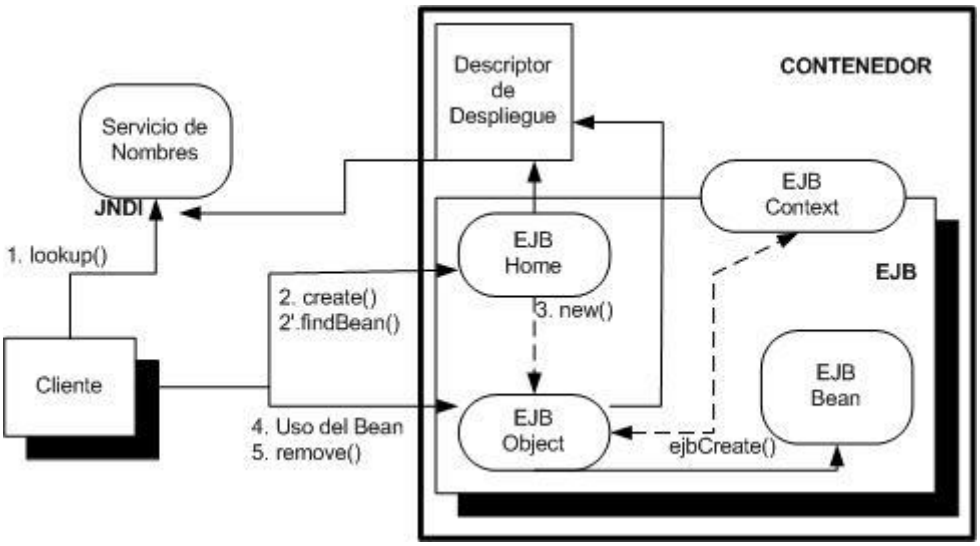
Programación distribuida: el modelo EJB.

La evolución de las arquitecturas cliente-servidor tradicionales, siempre han buscado el mismo fundamento, escalabilidad de las aplicaciones, reutilización de las aplicaciones a modo de servicios y minimizar en lo posible el impacto de un cambio de plataforma, nuevas funcionalidades, y cambios en la plataforma cliente. Hasta ahora la invocación remota de procedimientos (RPC), o el estándar

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

CORBA han proporcionado el sustento para la computación distribuida y dentro de la especificación J2EE aparece el modelo de componentes distribuidos denominados Enterprise JavaBeans (EJB).
La especificación J2EE también proporciona un modelo de contenedor para los EJBs con el fin de proporcionar una arquitectura estándar para la construcción de aplicaciones empresariales distribuidas basadas en componentes, guiar todo el ciclo de vida de la aplicación: desarrollo, despliegue y gestión en tiempo de ejecución, proporcionar portabilidad entre plataformas.(Write Once, Run Anywere) y ocultar al desarrollador de los detalles de bajo nivel relacionados con la seguridad, transacciones, distribución de objetos, concurrencia, conexiones con repositorios, etc.
En un modelo distribuido, los servicios no son accesibles de manera directa, son independientes del cliente que los usa y por tanto no tienen por qué estar en la misma máquina, ni el mismo segmento de red en el que se encuentra al cliente. De esta manera, es necesario proporcionar un sistema mediante el cual el servidor publique sus servicios y el cliente pueda buscarlos de manera remota antes de enlazarse con el servidor. El servidor publica en un servidor de nombres los detalles del servicio, siendo este una estructura jerárquica en forma de árbol, y el cliente navega a través del servidor de nombres buscando el servicio que necesita, una vez encontrado obtendrá toda la información necesaria para hacer uso de ese servicio [46].
En el modelo EJB el cliente nunca accederá directamente al componente que implementa el servicio, la localización, instanciación y uso del servicio se realiza por medio de una serie de interfaces, que rutarán entre el cliente y el servidor final, tal y como muestra la siguiente imagen:



El servicio implementado con la lógica de negocio viene representado por el EJB Bean, para acceder a él, el cliente tendrá que localizar su referencia en un servicio de nombres, al que accederá mediante JNDI, donde localizará el interfaz Home, que es el que incorpora los métodos necesarios para la búsqueda e instanciación del servicio. Una vez creado, el interfaz Home, devolverá un interfaz Object al cliente, este interfaz es el que tiene una descripción de los métodos de negocio que el cliente puede usar, toda llamada a estos métodos producirá una delegación a los métodos reales proporcionados por el EJB Bean. De esta manera el cliente nunca accede directamente a los métodos proporcionados por

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

el servicio. La configuración del comportamiento del componente, ciclo de vida, transacciones, seguridad, etc, viene definida de manera declarativa en el descriptor de despliegue que es gestionado por el contenedor.

Modelo de componentes.

Existen diversos componentes dependiendo del tipo de servicio que queramos proporcionar, se establecen tres tipos:

1. Componentes de Entidad (Entity Beans).
2. Componentes de Sesión (Session Beans)
3. Componentes de Mensajería (Message Drive Beans).

Componentes de Entidad.

Este tipo de componentes representan datos de la base de datos vistos como objetos, es decir son persistentes. Disponen de un acceso compartido ya que varios clientes pueden acceder a los mismos datos, y por lo tanto su ciclo de vida es largo, durarán tanto como duren los datos en la base de datos y como consecuencia sobreviven a las caídas de sistema. Dentro de este tipo de componentes hay dos variantes:

1. Container Manager Persistence (CMP).

2. Bean Manager Persistence (BMP).

El tipo CMP es gestionado íntegramente por el contenedor, realizando las operaciones necesarias para acceder a los datos de la base de datos, mientras que el tipo BMP es gestionado prácticamente por el desarrollador del componente. Todos los EJBs disponen de un ciclo de vida gestionado por el contenedor, la diferencia entre el tipo CMP y el BMP reside en que en el primero es el contenedor el que realiza todo el trabajo de acceso a datos, bloqueo, transaccionalidad, mapeo objeto-.relacional, etc. Mientras que en el segundo caso es cuestión del desarrollador proporcionar los mecanismos necesarios para realizar esas tareas. El tipo BMP es bastante más trabajoso que el CMP, pero proporciona más control y se suele utilizar cuando el modelo CMP no proporciona todo el control que requiere la aplicación.

Componentes de Sesión.

Estos componentes se ejecutan ligados a un cliente, por tanto, no son compartidos entre clientes, pueden ser transaccionales y aunque no representan datos en la base de datos pueden acceder y actualizar dichos datos, pero no es su cometido. Al no representar datos y estar ligados a un cliente, tienen una vida relativamente corta, tanto como dure la conversación con el cliente. No sobreviven a las caídas del sistema. Son la representación lógica del cliente en el servidor, contienen información específica del cliente y representan los procesos de negocio de la aplicación. Su uso suele ser para

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

realizar las operaciones de negocio, establecer flujos de peticiones y ocultar los detalles de implementación de los servicios proporcionados por los EJB de entidad a los clientes, accediendo por tanto los EJBs de sesión a los de entidad y haciendo de cliente de estos.
Dentro de este tipo de componentes hay dos variaciones:

1. Sesión sin estado (Session Beans Stateless).
2. Sesión con estado (Session Beans Stateful).

Los EJB de sesión sin estado no mantienen el estado conversacional con un cliente dado, son ligeros y no guardan ningún dato que pueda identificar al cliente, al revés que los EJB de sesión con estado cuyo cometido es proporcionar los mecanismos necesarios para mantener la conexión con el cliente hasta que o bien este o el servidor decidan cancelar la comunicación, este tipo de componentes suele llevar a cabo una serie de funcionalidades añadidas, de gestión y mantenimiento que hace que sean más pesados que sus compañeros, normalmente con operaciones de escritura en disco (serialización de objetos), que en algunos sistemas pueden producir efectos adversos de ralentización o en clustering y balanceo de carga.

Componentes de mensajería.

Los MDB aparecieron a partir de la especificación EJB 2.0, debido a la carencia de un modelo asíncrono. Utilizan JMS para encaminar los mensajes del cliente, pueden ser transaccionales, y aunque su cometido no es representar datos, pueden acceder y actualizar dichos datos. Su finalidad es prácticamente la misma que la de los EJB de Sesión, modelar la lógica de negocio de la aplicación, pero desde un punto de vista asíncrono, por lo tanto su vida es relativamente corta y tampoco sobreviven a las caídas del sistema.

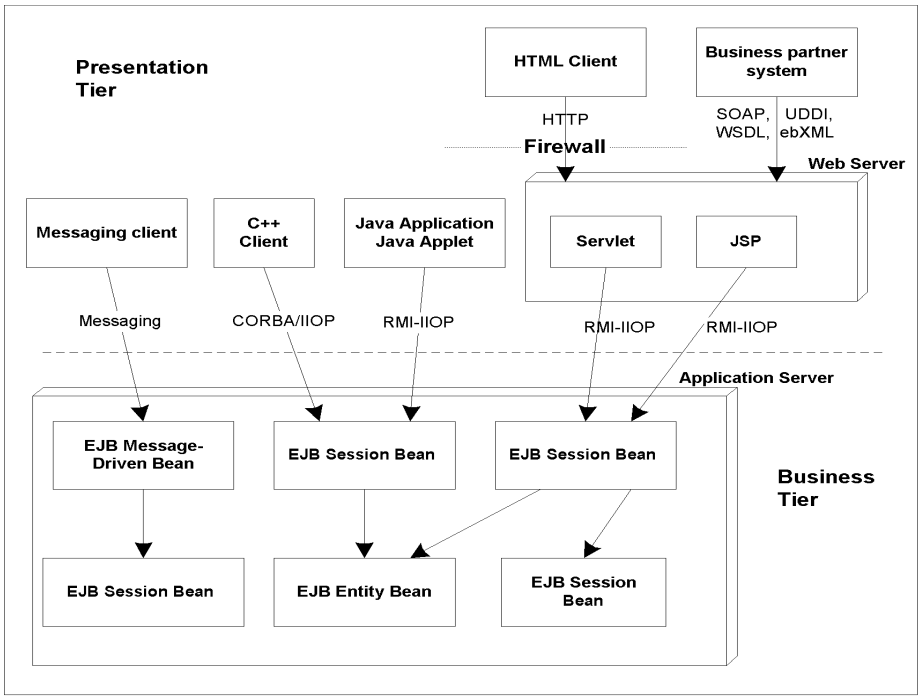
Novedades

Las novedades más importantes surgen a partir de la versión EJB 2.0, donde como ya se ha comentado se proporciona el soporte a la comunicación asíncrona mediante los MDBs, pero también se proporciona la posibilidad de publicar de manera local los interfaces de conexión de los componentes en lugar de manera remota como era antes obligado, lo que afectaba al rendimiento en algunas aplicaciones, sobretodo cuando toda la infraestructura se gestionaba en la misma máquina. La tercera novedad es la posibilidad de publicar los servicios usando la infraestructura de servicios web mediante XML, así usar el descubrimiento mediante UDDI y uso de los interfaces mediante WSDL usando SOAP como protocolo de transporte [47].

Uniéndolo todo. El servidor de aplicaciones

La especificación J2EE define un modelo común de desarrollo e implementación de aplicaciones de manera que estas sean portables y ejecutables entre distintas herramientas que cumplan la especificación con mínimo impacto, para ello define su modelo de contendor y el contrato con los componentes desplegados en el. Una arquitectura basada en J2EE da soporte para múltiples tipos de aplicaciones tal y como se muestra en la figura siguiente:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241



57.

58.Figura 19. Tipos de aplicaciones J2EE.

En la figura se aprecia los dos tipos de contenedores que forma parte de un servidor de aplicaciones en contenedor web y el contenedor EJB, dentro de la parte EJB aparecen los componentes comentados anteriormente y como los EJB de Sesión y MDBs hacen de clientes de los de Entidad ocultando la implementación hacia el cliente final, el contenedor también da soporte a clientes asíncronos mediante los MDBs y a aplicaciones que no sean Java, mediante el uso de los APIs proporcionados para comunicaciones con aplicaciones CORBA. Asimismo aparece una caja que simula aplicaciones Java que no son web, estas aplicaciones pueden acceder a los servicios implementados con EJBs haciendo uso de RMI, con lo que las aplicaciones en tecnología J2EE no tienen que ir orientadas exclusivamente hacia el mundo web, para el cual también hay soporte mediante el contenedor web, a través de los componentes ya mencionados (Servlets y JSPS). Si se usa una arquitectura web con un cliente ligero basado en html puede acceder a la aplicación por este camino, internamente los servlets harán de clientes de los EJBs usando RMI. La incorporación de los Servicios Web a este tipo de arquitecturas proporciona un marco idóneo, ya que la integración de aplicaciones no-Java pueden ser hechas usando el protocolo de transporte http y la infraestructura web de manera más sencilla que implementar la comunicación a través de un ORB de CORBA, o usando Sockets o RMI.

ISBN: 978-84-9012-861-9

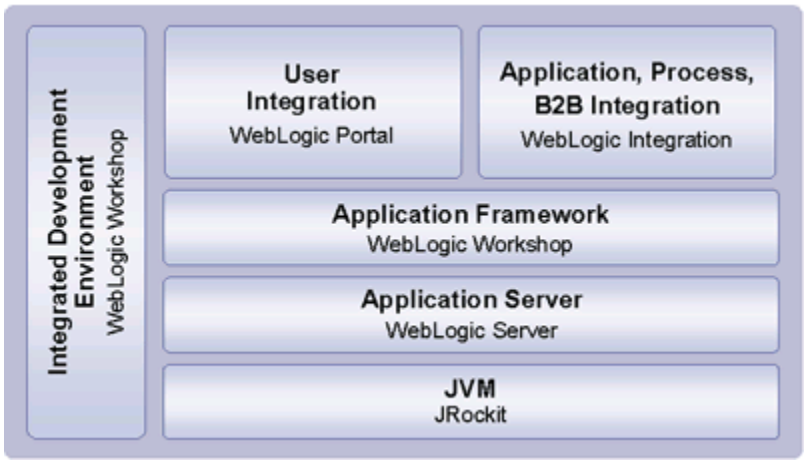
Colección Aquilafuente: AQ-241

La plataforma BEA WebLogic

La plataforma WebLogic constituye un conjunto de productos relacionados orientados a diferentes necesidades de desarrollo o producción. En su actual versión, la plataforma incluye los siguientes elementos:

1. *WebLogic Server*: Un servidor de aplicaciones basado en J2EE, junto a sus herramientas de desarrollo y administración.
2. *WebLogic Workshop*: Un entorno de desarrollo visual para el desarrollo de aplicaciones con la plataforma.
3. *WebLogic Integration*: Una solución de integración de aplicaciones especialmente indicada para el B2B.
4. *WebLogic Portal*: Un marco (*framework*) para el desarrollo de portales sobre la plataforma, incluyendo administración, personalización y gestión de contenidos.
5. *WebLogic JRockit*: Una máquina virtual Java optimizada para un mayor rendimiento y escalabilidad en el servidor.

El siguiente esquema muestra la relación entre los componentes mencionados.



59.Figura 20: Componentes BEA Weblogic

De estos componentes, aquí nos centramos en WebLogic Portal, que proporciona el soporte fundamental para el desarrollo de aplicaciones B2C.

Arquitectura de la plataforma WebLogic

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

La herramienta de configuración de WebLogic nos permite crear configuraciones para el desarrollo de diferentes tipos, teniendo habilitados unos servicios u otros. La Figura 1 nos muestra las configuraciones predefinidas que podemos seleccionar.

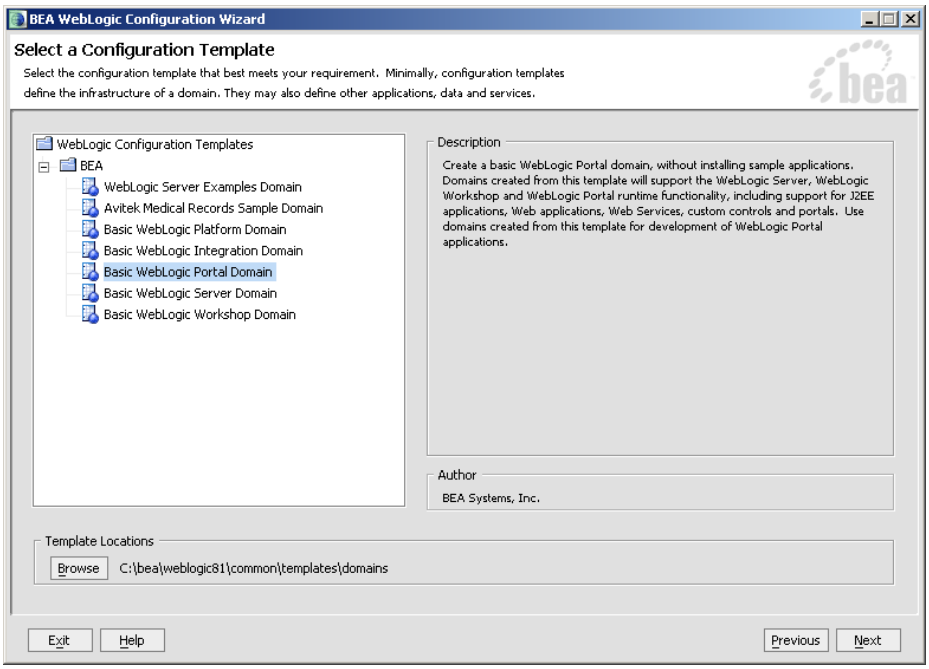


Figura 2: Configuración inicial de WebLogic platform

Al crear la configuración, se nos preguntará por una cuenta (usuario y password) para comenzar la administración posteriormente, y se generarán los ficheros iniciales de la aplicación en el directorio indicado, por ejemplo, en C:\bea\user_projects\domains\portalDomain. En ese directorio se encuentra un *script* de inicialización startWebLogic.cmd que utilizaremos para arrancar el servidor de administración.

Las instalaciones de WebLogic tienen como unidad administrativa el **dominio**, que representa un conjunto de recursos. Estos recursos pueden incluir más de un servidor WebLogic, y permite configuraciones en clúster de servidores, pero siempre hay uno de los servidores que funciona como Servidor Administrativo, siendo los demás "servidores gestionados" (*managed*). De esta manera se permite la *partición* de las aplicaciones, de manera que diferentes funcionalidades se asignen a diferentes servidores, y también la provisión de mecanismos de redundancia para mejorar la tolerancia a fallos de la instalación. De hecho, es recomendable que las aplicaciones en funcionamiento se asignen a servidores gestionados, de modo que la máquina de administración se utilice solamente para soportar tareas de administración.

La configuración del dominio se guarda en un fichero config.xml en la máquina que alberga al servidor de administración. La *Consola de Administración del Sistema* (CAS) puede utilizarse para crear

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

y modificar configuraciones mediante una interfaz gráfica de usuario. La Figura 2 muestra la apariencia de la consola de administración (accesible por HTTP a través del puerto de administración, por defecto el 7001), concretamente la página de administración de servidores dentro del dominio.

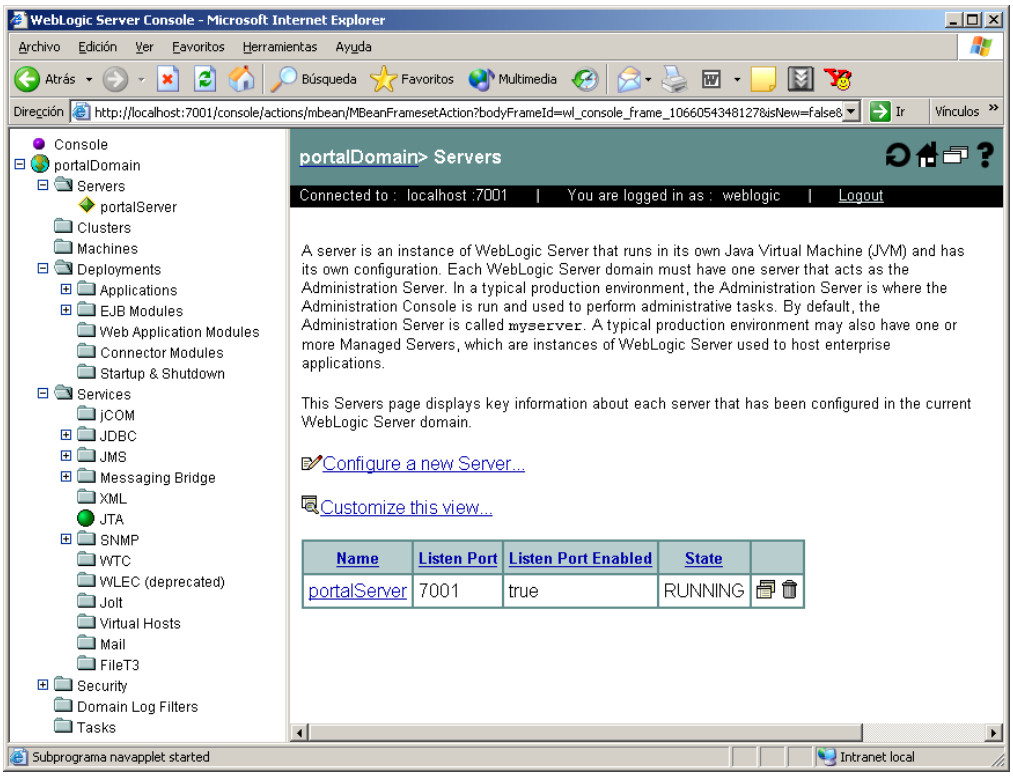


Figura 3: Consola de Administración de WebLogic platform

En la Figura 2 se pueden apreciar las diferentes opciones de administración, incluyendo clusters, instalación de aplicaciones o componentes (*deployments*), y otros servicios como controladores JDBC para acceso a bases de datos, o acceso a colas de mensajes JMS. También se puede configurar todo lo relativo a la seguridad, y permite el seguimiento de tareas (*tasks*) de administración que tardan un cierto tiempo en ejecutarse.

Desarrollo de portales con la plataforma WebLogic

Una vez creado un dominio, se pueden crear aplicaciones dentro de él. Para ello, se puede utilizar el editor integrado WebLogic Workshop. Cuando se utiliza *File|New|Application* tenemos que decidir el tipo de aplicación que queremos crear, como se muestra en la Figura 3.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

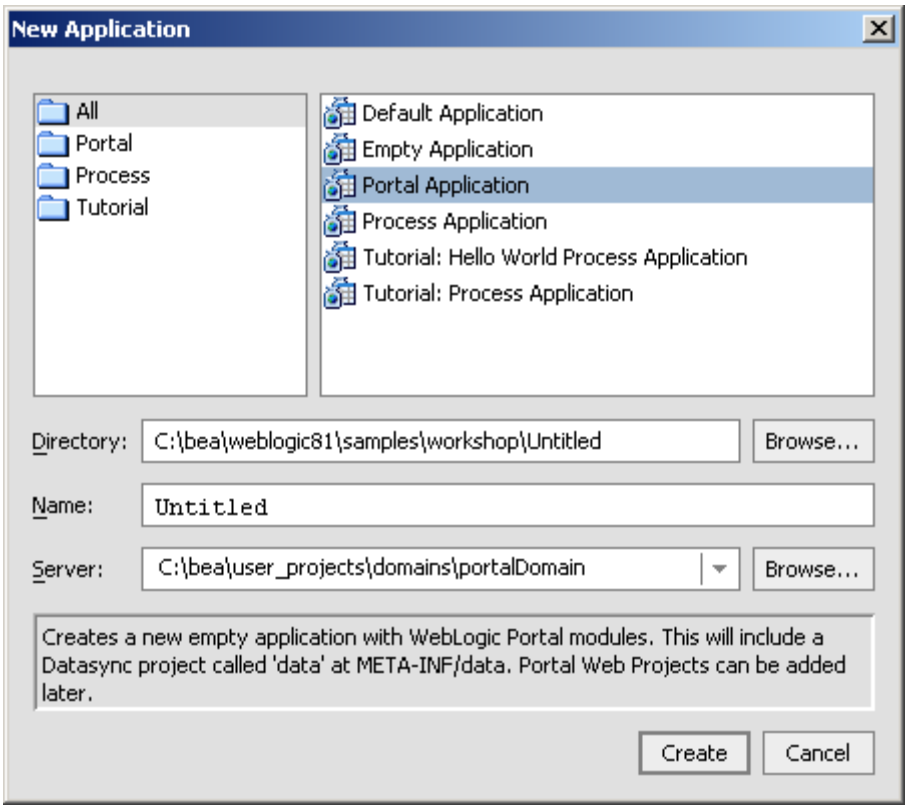


Figura 4: Creación de aplicaciones con WebLogic Workshop

La administración de portales en WebLogic se realiza con otra herramienta administrativa diferente a la Consola de Administración. Esta herramienta está accesible a través de la URL http://localhost:7001/Aplicacion1Admin, donde Aplicacion1 debe cambiarse por el nombre de la aplicación concreta que queramos administrar.

Examinemos a continuación un *portlet* muy sencillo, que implementa un diccionario (puede encontrarse en los tutoriales del producto):

```
<%@ taglib uri="render.tld" prefix="render" %>
<%-- ------------------------------------------------------------ --%>
<%-- Local variables                                              --%>
<%-- ------------------------------------------------------------ --%>
<%
   // url prefix of site
   String urlValue = "http://www.m-w.com/cgi-bin/dictionary?";

   // name of query parameter for the site
   String symbolParameterName = "va";

   // attributes of pop up windows
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
   String windowAttributes = "toolbar=no,alwaysRaise=yes,resizable=yes,scroll-
bars=yes,width=600,height=300";
%>
<%-- ------------------------------------------------------------ --%>
<%-- Portlet HTML content                                         --%>
<%-- ------------------------------------------------------------ --%>
<form method="post" action="<%=urlValue%>" onsubmit=" return <render:encodeName
name="DictionaryResultWindow"/>(this)">
   <table border="0" align="center">
      <tr>
         <td align="center">Enter the word you want to look up:</td>
      </tr>
      <tr>
         <td align="center">
            <input type="text" name="<%=symbolParameterName%>" size="15" maxlength="50" >
            <input type="hidden" name="lang" value="en">
            <input type="submit" name="wordSearchButton" value="Look up">
         </td>
      </tr>
   </table>
</form>

<script language="javascript">

/*
 * this is the local form handling.
 * since the actual search engine URL is not known till submit,
 * this intercepts the submit data and sends the full search URL
 * to the pop up window
 *
 * the function name must be unique to avoid conflict with other portlets
 */
function <render:encodeName name="DictionaryResultWindow"/>( form )
{
   // if browser is Netscape 4.x, let the from open the new
   // window instead of the script
   var appName = navigator.userAgent.toLowerCase();
   var isNetscape = ((appName.indexOf("mozilla") != -1) &&
                     (appName.indexOf("compat") == -1));
   var majorVersion = parseInt(navigator.appVersion);
   if (isNetscape && (majorVersion == 4)) return true;

   var urlPrefix = "<%=urlValue%>";
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
   var searchValue = form.elements["<%=symbolParameterName%>"].value;
   var fullUrl = urlPrefix + "<%=symbolParameterName%>=" +searchValue;

   // create initial window if it doesn't already exists
   window.open(fullUrl, "<render:encodeName name="result_window"/>", "<%= windowAttributes
%>");

   // never submit the form
   return false;
}

</script>
```

En el código anterior podemos apreciar que el *portlet* no es otra cosa que un fichero JSP con sus partes de código HTML, de gestión en cliente (javascript) y de código específico del *portlet*. Por tanto, la idea de portlet no es más que la de reutilizar fragmentos típicos de código Web, de manera que al registrarlos y definir su configuración y parámetros dentro de la plataforma WebLogic, se podrán reutilizar en diferentes portales.

La plataforma ATG

La plataforma ATG proporciona dos productos por separado que pueden utilizarse como infraestructura y soporte al desarrollo de aplicaciones B2C:

1. *Dynamo Application Server* (DAS) es un servidor de aplicaciones basado en J2EE.
2. *ATG* es un conjunto de productos que proporcionan soporte a la construcción de portales.

La versión actual de ATG es la 6.1 y es independiente del servidor de aplicaciones sobre la que se instala. Actualmente, soporta tanto DAS que es del mismo fabricante, como otros productos de fabricantes diferentes, concretamente, soporta los servidores J2EE de *WebLogic* y el *WebSphere* de IBM. De esta manera, lo propio de la gestión y desarrollo de portales en ATG permite utilizar un servidor de aplicaciones de otro fabricante.

Arquitectura de la plataforma ATG

Los componentes básicos de la plataforma ATG se muestran en el siguiente diagrama.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

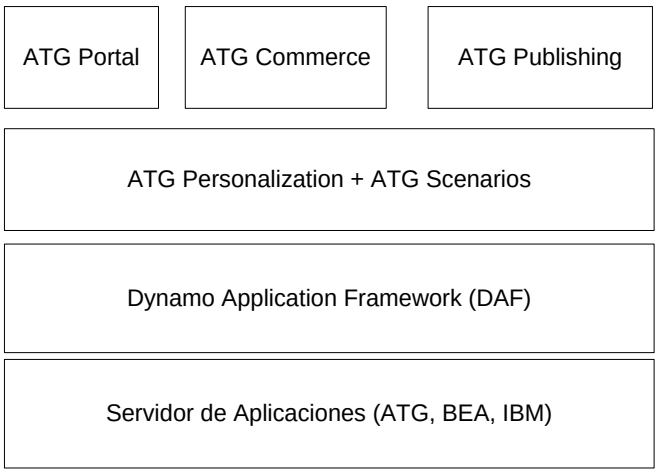


Figura 5: Componentes básicos de ATG.

Los componentes de ATG son, por tanto:

1. El servidor de aplicaciones, de entre los soportados por ATG.
2. Un marco de desarrollo básico, denominado DAF, que proporciona facilidades para el desarrollo e instalación de aplicaciones.
3. Dos módulos de personalización integrados. ATG Personalization proporciona un sistema de personalización basado en reglas que permite definir categorías de usuarios y de contenidos, y utilizar reglas para asociar contenidos personalizados a los usuarios. ATG Scenarios permite programar escenarios de interacción con los clientes, de modo que el sistema puede hacer campañas promocionales automatizadas y responder automáticamente a los usuarios de acuerdo al comportamiento de estos.
4. Tres productos basados en las capas anteriores. ATG Portal está orientado a construir portales para audiencias concretas mediante la reutilización de componentes. ATG Commerce proporciona servicios básicos de gestión de transacciones de compra y gestión de catálogo. Finalmente, ATG Publishing proporciona servicios de gestión de contenidos, incluyendo la posibilidad de definir y mantener flujos de trabajo (*workflows*).

La aplicación *ATG Control Center* (ACC) es el entorno de administración y desarrollo integrado para los elementos que acabamos de describir.

. Desarrollo de portales con la plataforma ATG

Los portales en ATG son colecciones de contenidos, servicios y usuarios. Los usuarios se pueden organizar en **comunidades.** ATG utiliza también el concepto de *Gear*, que viene a ser una utilidad o un área de contenidos de una página dentro de un portal (por ejemplo, un área de noticias, un foro, etc.). Son el equivalente al concepto de *Portlet* configurable y reutilizable de WebLogic Portal. Desde el punto de vista técnico, no son otra cosa más que un conjunto de ficheros JSP, clases Java y acceso

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

a servicio y datos que se pueden incluir y configurar en cualquier página del portal. Este es el concepto más relevante del desarrollo de portales en ATG.

La siguiente figura nos muestra un ejemplo ilustrativo de Gear soportando un foro de discusión, que se proporciona en la documentación de ATG, mostrándose en modo compartido, es decir, con otras instancias en la página.

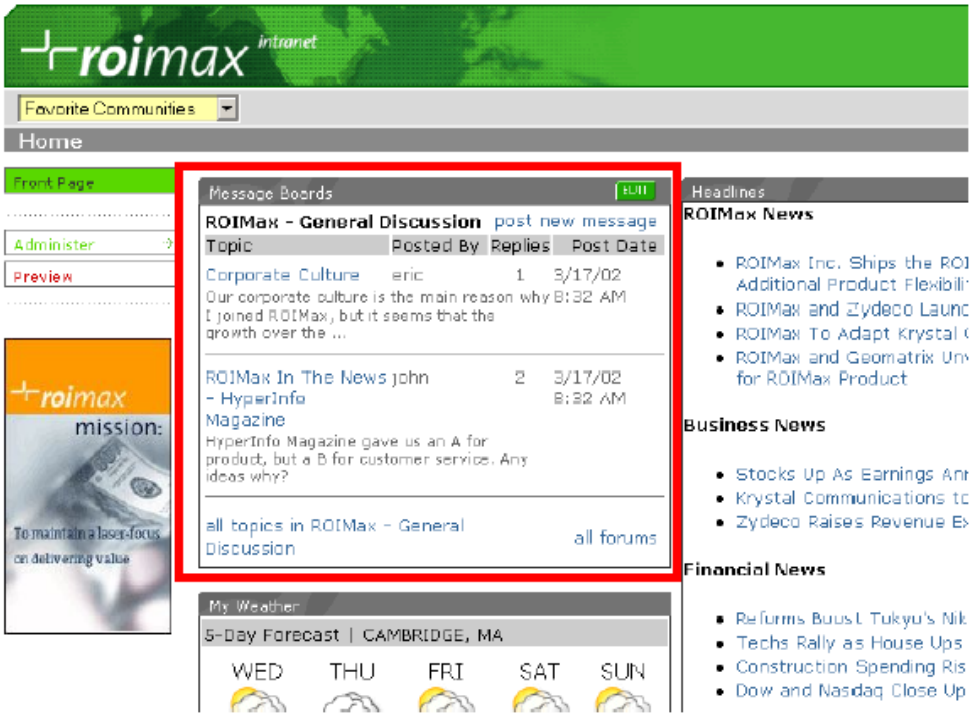


Figura 6: Gears en ATG.

Se pueden desarrollar *templates* de Gears, de modo que su registro posterior en el framework ATG los hace disponibles como componentes a cualquier portal. También se tiene la opción de diseñar para un mismo gear más de un perfil de visualización, de modo que se pueda mostrar, por ejemplo, mediante HTML y mediante WML en el caso de dispositivos móviles.

Veamos como ejemplo ilustrativo el código de la versión HTML de un *gear* mínimo:

```
<%@ page import="java.io.*,java.util.*,atg.portal.servlet.*,atg.portal.framework.*" %>
<%@ taglib uri="/jakarta-i18n-1.0" prefix="i18n" %>

<%
  GearServletResponse gearServletResponse =
    (GearServletResponse)request.getAttribute(Attribute.GEARSERVLETRESPONSE);
  GearServletRequest gearServletRequest =
    (GearServletRequest)request.getAttribute(Attribute.GEARSERVLETREQUEST);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
  GearContextImpl gearContext  = null;
  if((gearServletResponse != null) &&
     (gearServletRequest != null))
    gearContext = new GearContextImpl(gearServletRequest);
%>

<i18n:bundle baseName="atg.gears.helloworld.helloworld" localeAttribute="userLocale" changeResponseLocale="false"
/>

<h3><i18n:message key="helloworld-shared"/></h3>
<%
  //Full Mode
  gearContext.setDisplayMode(DisplayMode.FULL);
%>
<p>
 <i18n:message key="clickhere">
    <% String newFullGearUrlString = "<a href=\"" + gearServletResponse.encodeGearURL(gearServletRequest.getPor-
talRequestURI(),gearContext) + "\">"; %>
    <i18n:messageArg value="<%= newFullGearUrlString %>"/>
    <i18n:messageArg value="</a>"/>
 </i18n:message>
</p>

<h3><i18n:message key="gear-environment"/></h3>
<ul>
 <li><i18n:message key="orig-request-uri-label"/><%= gearServletRequest.getPortalRequestURI() %></li>
 <li><i18n:message key="community-label"/><%= gearServletRequest.getCommunity() %></li>
 <li><i18n:message key="page-label"/><%= gearServletRequest.getPage() %></li>
 <li><i18n:message key="displaymode-label"/><%= gearServletRequest.getDisplayMode() %></li>
 <li><i18n:message key="gear-label"/><%= gearServletRequest.getGear() %></li>
</ul>
```

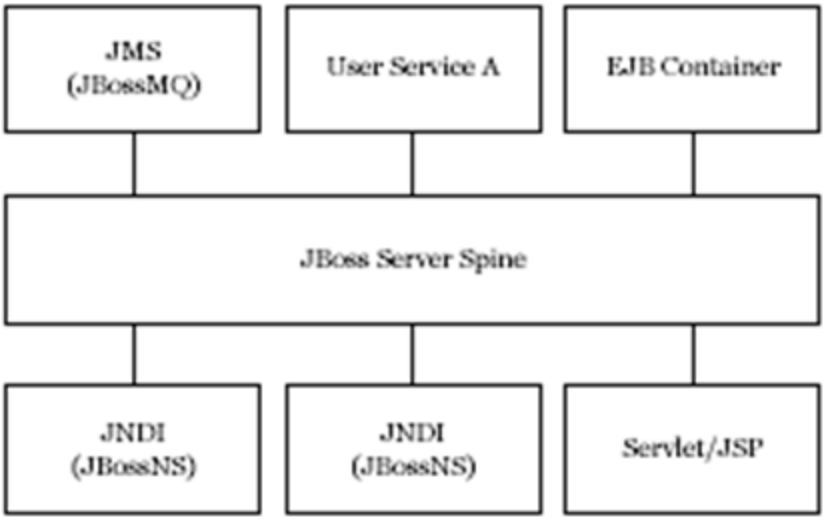
Como puede verse en el ejemplo anterior, tenemos simplemente un código JSP que invoca a ciertas funciones especiales, y muestra en la página información sobre sí mismo. A través de la instancia gearContext se puede controlar el comportamiento del mismo. El *gear* está además compuesto por otros ficheros de configuración, y posiblemente otros ficheros para versiones en otros lenguajes como WML o cHTML.

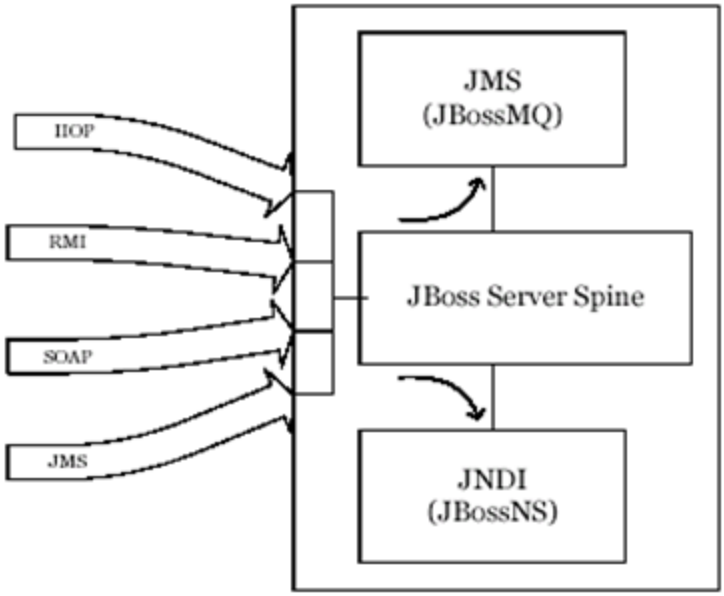
Jboss

JBoss es un servidor de aplicaciones J2EE de código abierto implementado en Java puro. Al estar basado en Java, JBoss puede ser utilizado en cualquier sistema operativo que lo soporte. Los principales desarrolladores trabajan para una empresa de servicios, JBoss Inc., adquirida por Red Hat en Abril del 2006, fundada por Marc Fleury, el creador de la primera versión de JBoss. El proyecto está apoyado por una red mundial de colaboradores. Los ingresos de la empresa están basados en un modelo de negocio de servicios.

JBoss implementa todo el paquete de servicios de J2EE. A modo de ejemplo, Los Sims online, utilizan JBoss para sus juegos multiusuario.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc





Jboss Portal

Es una plataforma de código abierto para albergar y servir un interfaz de portales Web, publicando y gestionando el contenido así como adaptando el aspecto de la presentación.

Como características principales categorizadas cabe destacar:

Tecnología y arquitectura

- JEMS: hace uso del potencial de JBoss Enterprise Middleware Services : JBoss Application Server, * JBoss Cache, Jgroups e Hibernate.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- DB Agnóstico: funciona con cualquier SGBD soportado por Hibernate
- SSO/LDAP: hace uso de las soluciones de single sign on (SSO) de Tomcat y JBoss
- Autenticación JAAS: módulos de autenticación adaptables vía JAAS
- Caché: utiliza cacheado en la capa de visualización para mejor rendimiento
- Clusterizable: soporte de Cluster que permite que un portal pueda ser desplegado en varias instancias
- Hot-Deployment: hace uso de las características de autodespliegue dinámico incluido en JBoss
- Instalador SAR: instalación basada en web que hace que la instalación y configuración inicial sea muy sencilla.

Estándares soportados

- Portlet Specification and API 1.0 (JSR-168)
- Content Repository for Java Technology API (JSR-170)
- Java Server Faces 1.2 (JSR-252)
- Java Management Extensión (JMX) 1.2
- Compatibilidad 100% con J2EE 1.4 al utilizar JBoss AS

Contenedor de Portales

- Múltiples Instancias de Portales: habilidad para ejecutar múltiples portales desplegados en un único contenedor.
- IPC (Inter-Portlet Communication): la API habilita a los portlets crear enlaces a otros objetos como páginas, portales o ventanas.
- Dynamicity: permite a administradores y usuarios crear y eliminar objetos como portlets, páginas, portales, temas y composición en tiempo de ejecución.
- Internacionalización: permite utilizar recursos de internacionalización para cada portlet.
- Servicios empotrables: la autenticación realizada por el contenedor de servlets y JAAS posibilita cambiar el esquema de autenticación.
- Arquitectura basada en Páginas: permite for the grouping/division of portlets on a per-page basis.
- Soporte de Frameworks existentes: los Portlets pueden utilizar Struts, Spring MVC, Sun JSF-RI, AJAX o MyFaces.

Temas y Layouts

- Temas y Layouts fácilmente intercambiables: los temas y layouts nuevos que contienen imágenes se pueden desplegar en ficheros WAR.
- API Flexible: la API de Temas y Layout están diseñados para separar la lógica de negocio de la capa de presentación.
- Estrategia de layout por página: a cada página se le puede asignar layouts distintos.

Funcionalidades de Usuarios y Grupos

- Registro y validación de usuarios: parámetros configurables del registro permite la validación de usuarios vía email previa a la activación.
- Acceso de usuarios: hace uso de la autenticación del contenedor de servlets.
- Crear/Modificar usuarios: habilita a los administradores crear/modificar perfiles de usuarios.
- Crear/Modificar roles: habilita a los administradores crear/modificar roles.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Asignación de roles: habilita a los administradores asignar roles a los usuarios.

Gestión de Permisos

- API extensible de permisos: permite asignar permisos de acceso a portlets basados en la definición de roles.

- Interfaz de administración: asignación de permisos a roles en cualquier momento para portlets, páginas o instancias de portal desplegados.

Sistema de gestión de contenidos

- Compatible JCR: el CMS utiliza Apache Jackrabbit, una implementación en código abierto del estándar Java Content Repository API.
- Soporte de almacenamiento en SGBD o en el sistema de ficheros.
- Soporte externo de contenidos tipo Blob (binarios): se puede configurar el almacenamiento en el sistema de ficheros de contenido binario de gran tamaño y los nodos con las referencias y propiedades residan en el SGBD.
- Control de versiones: Todo contenido modificado/creado es autoversionado con el historial de cambios, que pueden ser revisados en cualquier momento.
- Contenidos mostrados en URLs amigables para los motores de búsqueda: http://yourdomain/portal/content/index.html (sin incluir las acciones de los portlets)
- URLs del portal sencillas: mostrar descarga de binarios con URLs fáciles de recordar. (http://domain/files/products.pdf)
- Soporte de múltiples instancias de Portlets HTML: permite que instancias extra de contenido estático del CMS sean publicadas en ventanas distintas.
- Soporte de directorios: crear, mover, eliminar, copiar y subir árboles completos de directorios.
- Funciones de Ficheros: crear, mover, copiar, cargar y eliminar ficheros.
- Explorador de directorios embebido: cuando se copia, mueve, elimina o se crean nuevos ficheros, los administradores pueden simplemente navegar por el árbol de directorios hasta encontrar la colección para que quieren realizar la acción.
- Arquitectura fácil de usar: todas las acciones que se pueden realizar sobre ficheros pueden hacerse a base de clicks de ratón.
- Editor HTML: con modo WYSIWYG, previsualización y edición de código HTML. Soporta la creación de tablas, fuentes, zoom, enlaces a imágenes y URLs, soporte de películas flash, listas con viñetas o numéricas...
- Soporte de editor de estilos: el editor WYSIWYG muestra la hoja de estilo actual del Portal, para un sencillo intercambio de clases.
- Soporte de Internacionalización: los contenidos pueden ser asignados para una zona regional determinada y ser mostrada en función de la configuración de usuario o basado en las opciones del navegador web .

Tablón de mensajes

- Respuesta inmediate mediante un sólo click.
- Respuesta con cita: se puede citar un tema existente al responder.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- Control del flujo: previene el abuso de envío masivo de mensajes mediante una ventana de tiempo configurable.
- Creación de categorías contenedoras de foros.
- Operaciones sobre Foros: se puede crear un foro y asignarlo a una categoría específica, además se puede copiar, mover, modificar y eliminar.
- Reordenación de foros y categorías: se puede establecer el orden en el que se quiere que aparezcan los foros y categorías en las páginas.

Otras alternativas

Dentro del mundo Java existen otras alternativas opensource para la creación de portales que contienen todos los servicios anteriormente mencionados por las otras plataformas y que cumplen los estándares de industria, y que son alternativas perfectamente válidas para un proyecto de grandes prestaciones.

- Open nuke: https://opennuke.dev.java.net/
- LifeRay: http://www.liferay.com/web/guest/home
- InfoGlue: http://www.infoglue.org/infoglueDeliverLive/
- Apache Lenya: http://lenya.apache.org/index.html
- Apache JetSpeed: http://portals.apache.org/

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Páginas web de consulta.

- Área de descargas de BEA
- [http://commerce.bea.com/index.jsp]
- Área de descargas de ATG
- [http://www.atg.com/en/myatg/mydownloads.jhtml]
- The Server Side: Web de recursos sobre servidores de aplicaciones basados en J2EE.
- [http://theserverside.com]

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Adrián Sánchez-Carmona, Sergi Robles, Carlos Borrego (2015). Improving Podcast Distribution on Gwanda using PrivHab: a Multiagent Secure Georouting Protocol. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
2. Anderson Sergio, Sidartha Carvalho, Marco Rego (2014). On the Use of Compact Approaches in Evolution Strategies. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
3. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. Information Sciences, 180(10), 2029–2043. https://doi.org/10.1016/j.ins.2009.12.032
4. Buciarelli, E., Silvestri, M., & González, S. R. (2016). Decision Economics, In Commemoration of the Birth Centennial of Herbert A. Simon 1916-2016 (Nobel Prize in Economics 1978): Distributed Computing and Artificial Intelligence, 13th International Conference. Advances in Intelligent Systems and Computing (Vol. 475). Springer.
5. Canizes, B., Pinto, T., Soares, J., Vale, Z., Chamoso, P., & Santos, D. (2017). Smart City: A GECAD-BISITE Energy Management Case Study. In 15th International Conference on Practical Applications of Agents and Multi-Agent Systems PAAMS 2017, Trends in Cyber-Physical Multi-Agent Systems (Vol. 2, pp. 92–100). https://doi.org/10.1007/978-3-319-61578-3_9
6. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
7. Casado-Vara, R., de la Prieta, F., Prieto, J., & Corchado, J. M. (2018, November). Blockchain framework for IoT data quality via edge computing. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (pp. 19-24). ACM.
8. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
9. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
10. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
11. Chamoso, P., de La Prieta, F., Eibenstein, A., Santos-Santos, D., Tizio, A., & Vittorini, P. (2017). A device supporting the self-management of tinnitus. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 10209 LNCS, pp. 399–410). https://doi.org/10.1007/978-3-319-56154-7_36
12. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
13. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
14. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
15. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
16. Choon, Y. W., Mohamad, M. S., Deris, S., Illias, R. M., Chong, C. K., Chai, L. E., … Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. PLoS ONE, 9(7). https://doi.org/10.1371/journal.pone.0102744
17. Constantino Martins, Ana Rita Silva, Carlos Martins, Goreti Marreiros (2014). Supporting Informed Decision Making in Prevention of Prostate Cancer. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
18. Costa, Â., Novais, P., Corchado, J. M., & Neves, J. (2012). Increased performance and better patient attendance in an hospital with the use of smart agendas. Logic Journal of the IGPL, 20(4), 689–698. https://doi.org/10.1093/jigpal/jzr021

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

19. David Griol, Jose Manuel Molina, Araceli Sanchís De Miguel (2014). Developing multimodal conversational agents for an enhanced e-learning experience. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
20. Eva L. Iglesias, Lourdes Borrajo, R. Romero (2014). A HMM text classification model with learning capacity. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
21. Fernández-Riverola, F., Díaz, F., & Corchado, J. M. (2007). Reducing the memory size of a Fuzzy case-based reasoning system applying rough set techniques. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 37(1), 138–146. https://doi.org/10.1109/TSMCC.2006.876058
22. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. Expert Systems with Applications, 41(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003
23. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. BMC Bioinformatics, 10. https://doi.org/10.1186/1471-2105-10-187
24. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
25. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
26. González-Briones, A., De La Prieta, F., Mohamad, M., Omatu, S., & Corchado, J. (2018). Multi-agent systems applications in energy optimization problems: A state-of-the-art review. Energies, 11(8), 1928.
27. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
28. Hafewa Bargaoui, Olfa Belkahla Driss (2014). Multi-Agent Model based on Tabu Search for the Permutation Flow Shop Scheduling Problem. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
29. Jamal Ahmad Dargham, Ali Chekima, Ervin Gubin Moung, Sigeru Omatu (2014). The Effect of Training Data Selection on Face Recognition in Surveillance Application. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
30. Jorge Agüero, Miguel Rebollo, Carlos Carrascosa, Vicente Julián (2013). MDD-Approach for developing Pervasive Systems based on Service-Oriented Multi-Agent Systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
31. Juan Castro, Pere Marti-Puig (2014). Real-time Identification of Respiratory Movements through a Microphone. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
32. Leonor Becerra-Bonache, M. Dolores Jiménez López (2014). Linguistic Models at the Crossroads of Agents, Learning and Formal Languages. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
33. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
34. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). Random finite set-based Bayesian filters using magnitude-adaptive target birth intensity. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637788&partnerID=40&md5=bd8602d6146b014266cf07dc35a681e0
35. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. Applied Mathematics and Computation, 270, 756–767. https://doi.org/10.1016/j.amc.2015.08.059
36. Margherita Brondino, Gabriella Dodero, Rosella Gennari, Alessandra Melonio, Daniela Raccanello, Santina Torello (2014). Achievement Emotions and Peer Acceptance Get Together in Game Design at School. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

37. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. Expert Systems with Applications, 36(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
38. Miki Ueno, Naoki Mori, Keinosuke Matsumoto (2014). Picture models for 2-scene comics creating system. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
39. Ming Fei Siyau, Tiancheng Li, Jonathan Loo (2014). A Novel Pilot Expansion Approach for MIMO Channel Estimation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
40. Omar Jassim, Moamin Mahmoud, Mohd Sharifuddin Ahmad (2014). Research Supervision Management Via A Multi-Agent Framework. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
41. Pablo Chamoso, Henar Pérez-Ramos, Ángel García-García (2014). ALTAIR: Supervised Methodology to Obtain Retinal Vessels Caliber. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
42. Paula Andrea Rodríguez Marín, Néstor Duque, Demetrio Ovalle (2015). Multi-agent system for Knowledge-based recommendation of Learning Objects. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
43. Rodríguez, S., De La Prieta, F., Tapia, D. I., & Corchado, J. M. (2010). Agents and computer vision for processing stereoscopic images. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6077 LNAI). https://doi.org/10.1007/978-3-642-13803-4_12
44. Rodríguez, S., Gil, O., De La Prieta, F., Zato, C., Corchado, J. M., Vega, P., & Francisco, M. (2010). People detection and stereoscopic analysis using MAS. In INES 2010 - 14th International Conference on Intelligent Engineering Systems, Proceedings. https://doi.org/10.1109/INES.2010.5483855
45. Román, J. A., Rodríguez, S., & de da Prieta, F. (2016). Improving the distribution of services in MAS. Communications in Computer and Information Science (Vol. 616). https://doi.org/10.1007/978-3-319-39387-2_4
46. Sigeru Omatu, Tatsuyuki Wada, Pablo Chamoso (2013). Odor Classification using Agent Technology. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
47. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. International Journal of Ambient Computing and Intelligence, v 1, n 1(1), 15–26. https://doi.org/10.4018/jaci.2009010102

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Aplicaciones Móviles En Entornos Servidor

Roberto Casado-Vara [1]

[1] University of Salamanca, Plaza de los Caídos s/n – 37002 – Salamanca, Spain
rober@usal.es

Resumen: En este documento vamos a analizar desde diferentes facetas la transmisión de datos en los entornos celulares. Empezaremos analizando en profundidad las distintas características de las redes móviles celulares que implicarán consecuencias realmente interesantes de estudiar en la transmisión de datos en estas redes. Posteriormente analizaremos algunos modelos de negocio que se establecen en las aplicacio-nes de descargas de contenidos y de acceso a Internet. En este apartado haremos un análisis en profundidad de los modelos de WAP e i-mode estableciendo semejanzas y diferencias. Finalmente, los dos últimos apartados giran en torno a la parte más tecnológica de las aplicaciones cliente-servidor. La descarga segura de contenidos es un tema importante en el mundo móvil porque ha generado bastante dinero y porque se considera como una de las mayores fuentes de ingresos futuros. Sobre este tema han ido apareciendo tecnologías propietarias que finalmente han desembocado en un estándar común tanto de los procedimientos de descarga como de los métodos de protección de los contenidos. Además, también analizaremos los protocolos de transmisión, las características y los forma-tos de los contenidos multimedia. La transmisión de sonidos, imágenes e incluso vídeos es considerada como una de las grandes oportunidades para aprovechar los grandes anchos de banda que están disponibles en las redes 3G.

Palabras clave: aplicaciones móviles

Abstract. In this document we will analyze from different facets the data transmission in cellular environments. We will begin analyzing in depth the different characteristics of cellular mobile networks that will imply really interesting consequences of studying in the data transmission in these networks. Later we will analyze some business models that are established in the applications of content downloads and Internet access. In this section we will make an in-depth analysis of the WAP and i-mode models, establishing similarities and differences. Finally, the last two sections revolve around the more technological part of client-server applications. Safe downloading of content is an important issue in the mobile world because it has generated a lot of money and because it is considered to be one of the biggest sources of future income. Proprietary technologies have been appearing on this subject, which have finally resulted in a common standard of both downloading procedures and content protection methods. In addition, we will also analyse the transmission protocols, characteristics and forms of multimedia content. The transmission of sounds, images and even videos is considered to be one of the great opportunities to take advantage of the large bandwidths that are available on 3G networks.

Keywords: mobile applications

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

1. Introduction

Todo el mercado de las telecomunicaciones móviles celulares está sufriendo un importante cambio de mentalidad y objetivos. Hasta hace poco la estrategia giraba en torno a conseguir el mayor número de clientes para los operadores y vender el mayor número de equipos y terminales para los fabricantes. Pero en un sector en el que la penetración de los usuarios ronda el 100%, esas metas han dejado de tener sentido.

Los objetivos ahora son distintos, teniendo un número alto y estable de clientes las estrategias giran en torno a otros fines, sin olvidar por supuesto el aumento de usuarios a costa de la competencia. Si suponemos una base importante de clientes, interesa mejorar la calidad de los mismos (que sean los más posibles de contrato), que gasten lo más posible (aumentar el ARPU, *Average Revenue Per User*) y que cambien lo más posible de terminal.

Para conseguir estos dos últimos ejecutivos es importante definir y desarrollar nuevas tecnologías que permitan por un lado proporcionar nuevos servicios atractivos que den valor añadido al cliente y que permitan mayor número de ingresos por cada usuario, y que por otro lado motivarán que los plazos de cambio de terminal se acorten.

En particular, los operadores están siguiendo claramente una línea de acción con la intención de alejarse lo más posible del modelo de negocio con el que suelen trabajar las *utilities* (energía, agua, etc.) y que se suelen basar en productos sin diferenciar con precios muy ajustados en costes [1-6].

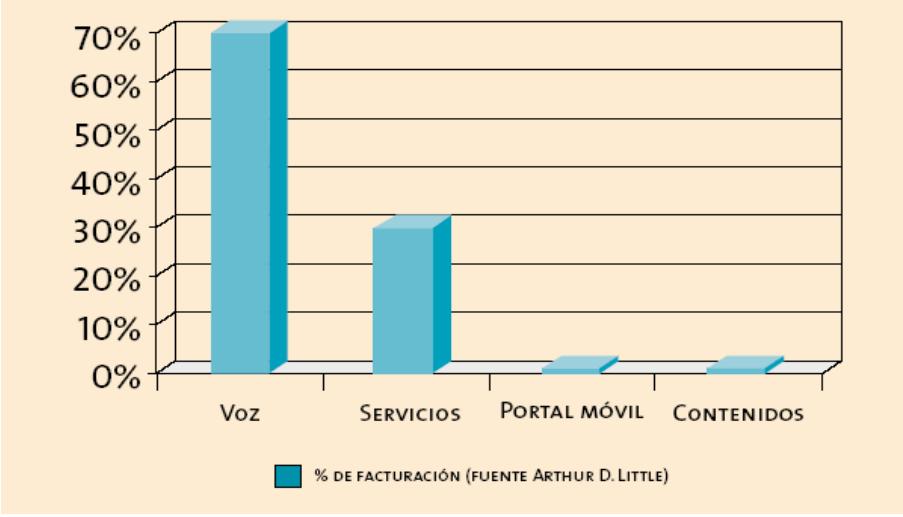


Figura 0.21: Distribución tradicional de la facturación en los operadores móviles

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Las operadoras móviles actualmente están intentando que los servicios de voz sean parte del conjunto de servicios más amplio que les permita diversificar sus ingresos y aumentarlos sin necesidad de modificar las tarifas o aumentar los usuarios.

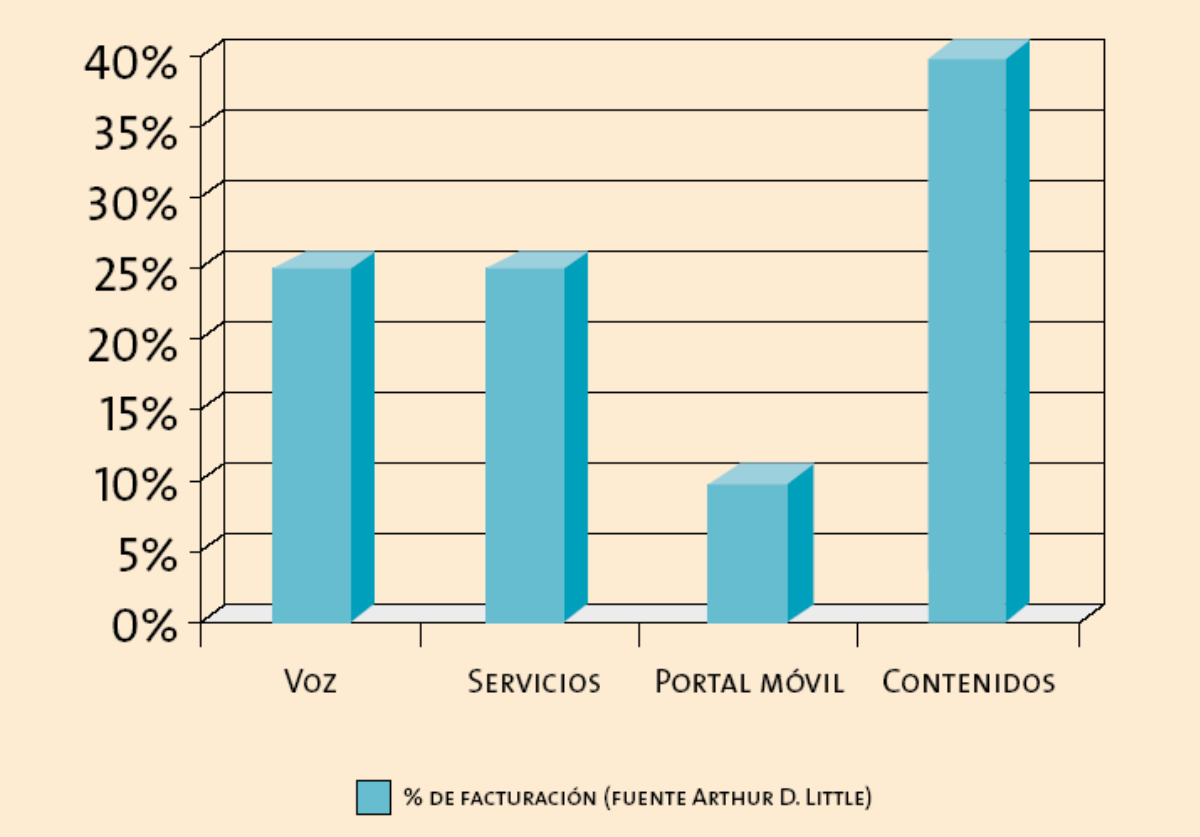


Figura 0.22: Distribución futura de la facturación en los operadores móviles

En este contexto es donde cobra gran importancia todas las tecnologías y modelos de negocio referentes a las aplicaciones cliente-servidor en el mundo móvil. Mediante las aplicaciones que generen tráfico de datos en la red móvil los ingresos aumentarán fácilmente ya sea por el coste del propio tráfico o por el precio de los contenidos o las aplicaciones utilizadas.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

2. TRANSMISIÓN DE INFORMACIÓN EN ENTORNOS MÓVILES

La transmisión de información con redes de telecomunicaciones es un hecho muy habitual en nuestros días. La sociedad asume como normal que cualquier aplicación informática pueda conectarse a otras máquinas (servidores) para obtener o mandar información. Lo que está cambiando poco a poco, es que también los terminales móviles están empezando a necesitar y a promover intercambios de datos más allá de la voz.

Durante los últimos años hemos asistido a un proceso por el cual las redes de comunicación fijas se han ido adaptando al patrón de intercambio basado en datos. Las aplicaciones han ido apareciendo y las redes han evolucionado para proporcionar una velocidad y unas características óptimas para ellas.

En cambio, en las redes móviles estamos en fases anteriores. Hasta ahora la aplicación estrella ha sido la voz y para ella estaban pensadas todas las redes celulares. La única excepción a este hecho ha sido la mensajería a través de mensajes cortos de texto. Aunque incluida en el estándar GSM, no se había previsto ningún tipo de éxito a esta funcionalidad por lo que las redes no estaban preparadas para soportar el tráfico actual. Es el primer ejemplo de cómo las redes han tenido que ir evolucionando tanto en estándares como en planificación para dar soporte a aplicaciónes con intercambio de datos [7-10].

Además de esta esperada evolución, la comunicación inalámbrica tiene intrínsecamente ciertas peculiaridades que afectan directamente a la transmisión de información a través de ellas. Por ejemplo, existen fenómenos como los desvanecimientos o el multitrayecto que no suceden en entornos basados en cables y que provocan que aplicaciones en las que se necesita un caudal constante de datos haya que tener en cuenta que éste se puede cortar y haga necesarios mecanismos para solventar estos problemas.

Aún con todo lo contado, no todo lo relativo a las tecnologías móviles es negativo. En el tema de la seguridad de la información, aunque se utilice un medio compartido y accesible como los enlaces radio, las tecnologías digitales han permitido la transmisión segura tanto de voz como de datos.

En este apartado vamos a desglosar todas las características importantes de la transmisión de datos en los entornos móviles celulares intentando identificar que puntos van a mejorar o a cambiar en las nuevas redes 3G. Además, se va a analizar las consecuencias de estas peculiaridades en el diseño y desarrollo de aplicaciones móviles [11-17].

2.1.Características particulares de la comunicación móvil

Vamos a analizar una por una las principales características de la transmisión de información mediante redes móviles celulares, distinguiendo claramente de qué tipo de redes estamos hablando, (1G, 2G, 2,5G o 3G). En algunos casos, para facilitar la comprensión de esas peculiaridades, vamos también a establecer comparaciones entre las generaciones.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Gran parte de estas propiedades son consecuencia de la orientación de las redes a la comunicación de voz y la antigüedad de la mayoría de estándares (GSM data sus comienzos en 1982).

2.1.1 Uso de un medio compartido

Por definición las comunicaciones móviles utilizan el aire para transmitir la información. Evidentemente, este medio es accesible por cualquier agente con el equipo adecuado. Esto podría suponer un problema grave de seguridad en las comunicaciones de datos (y por supuesto también de voz) ya que cualquier persona que se situase lo bastante cerca de nosotros podría "pinchar" la línea.

De hecho, en los primeros sistemas analógicos 1G de telefonía celular escuchar una conversación era tan sencillo como utilizar un analizador de espectro y sintonizar la frecuencia de la conversación. Con la aparición de los sistemas celulares digitales la situación dio un giro de 360 grados. Las redes móviles celulares pasaron de tener una seguridad inexistente o ridícula a tener un sistema prácticamente inexpugnable. A día de hoy existen mecanismos para romper la seguridad de conversaciones GSM pero son realmente complejos y por lo tanto extremadamente caros. Y aún así no se puede realizar en tiempo real. La seguridad de una transmisión móvil con un sistema digital se puede considerar por ahora como muy segura [18-21].

2.1.2 Conmutación de circuitos vs. conmutación de paquetes

Las primeras redes móviles de comunicación estaban orientadas a la comunicación de voz. En esa época, los datos todavía no tenían la importancia actual y era considerados como contenidos alternativos. Eso provocó que hasta las tecnologías 2G, como GSM, estuvieran orientadas a la transmisión de voz mediante canales reservados. Este tipo de comunicación se denomina conmutación de circuitos y se basa en la reserva de un cierto ancho de banda para la comunicación. Por lo tanto en este esquema se utiliza un canal de la misma capacidad siempre se use parcialmente o totalmente. Dependiendo del tipo de tráfico, esto provoca ineficiencias en el uso del sistema y que el usuario tenga que pagar por el tiempo de conexión y no por el uso real que ha hecho.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

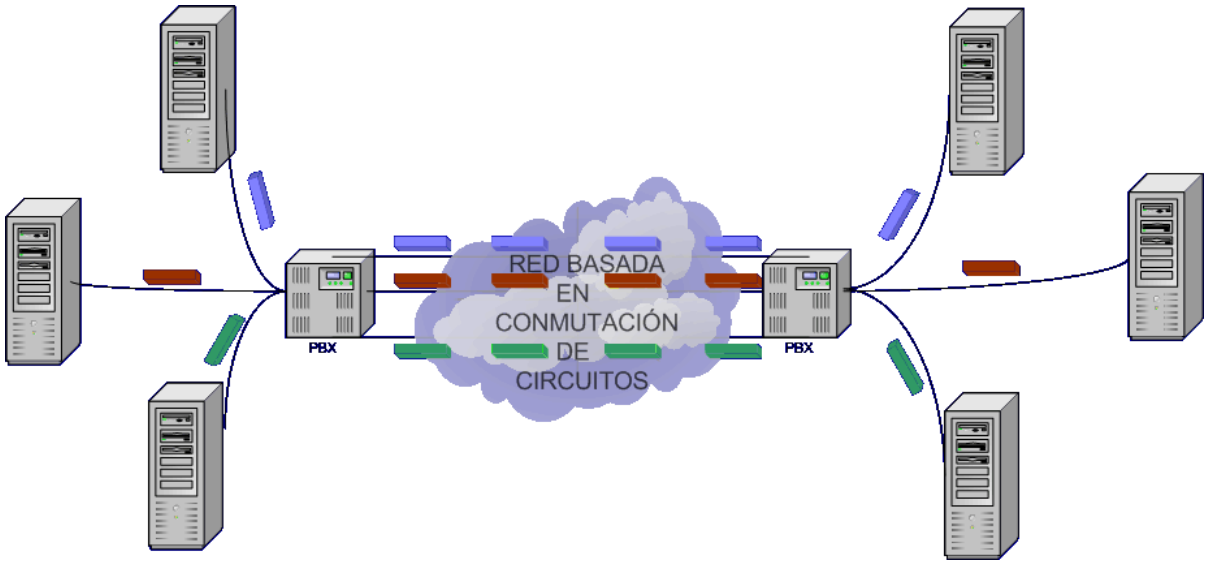


Figura 0.23: Red basada en conmutación de circuitos

A partir de la aparición de las redes 2,5G y 3G, como GPRS y UMTS las redes permiten el uso de conmutación de paquetes en la transmisión de datos. De esta forma se comparten los canales de transmisión de datos entre los diferentes usuarios optimizando el uso de la red. Como consecuencia los usuarios sólo pagan por la información que realmente transmiten en ambos sentidos.

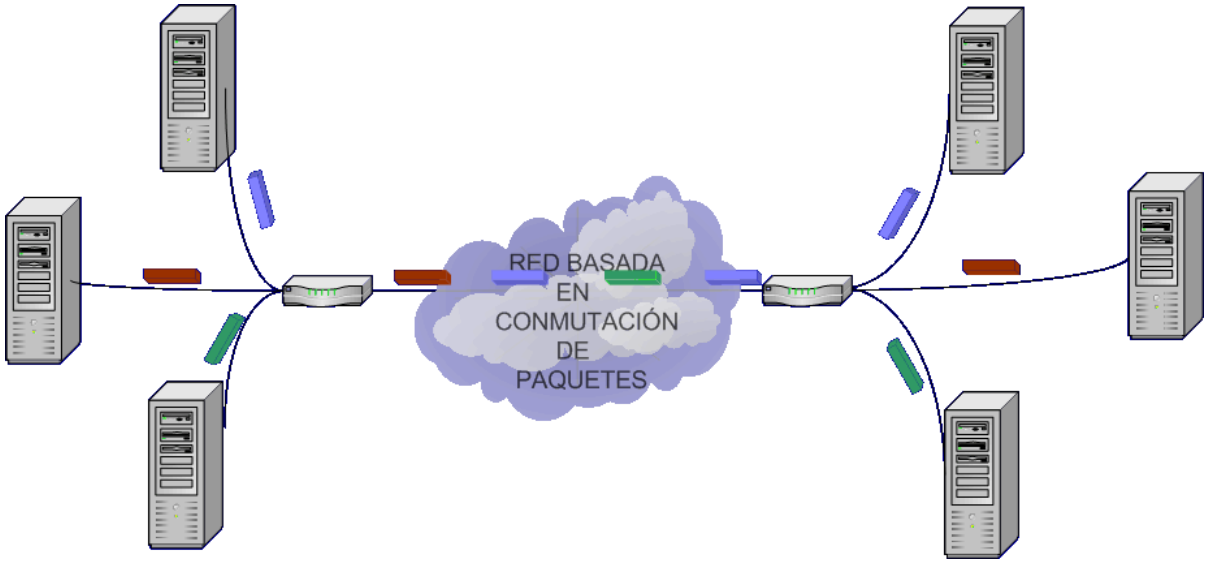


Figura 0.24: Red basada en conmutación de paquetes

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Es de destacar, que aunque estas características diferencian a las redes móviles de diferentes generaciones, no son peculiaridades de las redes inalámbricas. En las redes fijas existen los mismos esquemas y se producen los mismos problemas y diferencias.

2.1.3 Desvanecimientos de la comunicación

Lo que sí que es una característica especial de las redes móviles son los desvanecimientos de la conexión debido a problemas en la propagación de la señal por el aire. Evidentemente, fluctuaciones en la capacidad de la transmisión se pueden encontrar en cualquier comunicación con compartición del canal (conmutación de paquetes), pero en las redes fijas o con cables, si se reserva un canal para una conmutación, éste va a mantenerse.

En las redes móviles, incluso con conmutación de circuitos podemos encontrar desvanecimientos de la comunicación debido a problemas como falta de cobertura, problemas con los traspasos, efectos negativos del multitrayecto, etc.

2.1.4 Ancho de banda escaso

Otra característica fundamental de las comunicaciones móviles es la limitación que supone el espectro radioeléctrico. Puesto que este recurso es escaso y regulado por la administración competente, los operadores no pueden ampliar las frecuencias a usar de forma arbitraria. Esto se traduce en un ancho de banda limitado y por lo tanto caro. Hasta ahora, cualquier de comunicación de datos mediante redes móviles tenía menor velocidad y un precio comparativamente mucho mayor que la equivalente en las redes fijas.

2.1.5 Retardo alto

Otro problema que tienen actualmente las redes móviles es el retardo elevado que se produce en el establecimiento de conexiones. Este problema cada vez va siendo menor con las mejoras que en este aspecto están introduciendo las redes de tercera generación.

2.1.6 Simetría en las comunicaciones

Otra consecuencia de la orientación de las redes 1G y 2G hacia las comunicaciones de voz es la simetría de la capacidad en la transmisión. El ancho de banda que se proveía en las transmisiones (tanto de voz como de datos) era igual tanto en el enlace ascendente como en el descendente. Esto era claramente un desperdicio según el patrón de comunicaciones de la mayoría de aplicaciones de Internet, en la que el gasto de ancho de banda en el canal descendente es unas diez veces mayor que el correspondiente del descendente [22-25].

Con la aparición de las redes 2,5 esto se ha solucionado en parte puesto que se permite usar más canales compartidos de descarga que de subida. En algunas tecnologías 3G diseñadas especialmente para la transmisión de datos la asimetría es totalmente configurable .

2.1.7 Limitaciones en los terminales

En las redes fijas, sobre todo conectadas a terminales inteligentes como ordenadores, la capacidad de los terminales era más que suficiente para gestionar todos los contenidos y servicios que se podían implementar y transmitir mediante las redes.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

En las redes móviles, los terminales han ido evolucionando a la par que las redes, pero en muchos casos éstas han ido por delante de los dispositivos. Esto ha provocado que muchas veces lo que impedía nuevos servicios y contenidos más avanzados hayan sido los propios terminales y no las redes.

Las limitaciones más importantes de los dispositivos móviles, son la capacidad de conmutación, la calidad de la reproducción de los contenidos multimedia como gráficos y sonidos, las pantallas y por supuesto la interfaz de usuario con teclados numéricos. Estas limitaciones vienen impuestas por la necesidad de que los terminales sean portátiles, con lo que la autonomía, el tamaño y el peso son requisitos imprescindibles.

Este es uno de los apartados en los que la brecha que existe entre las redes fijas y móviles ha sido más grande en tiempos pasados. También es cierto que la diferencia que se está haciendo más pequeña cada vez y que los terminales de dentro de pocos años van a ser pequeños ordenadores portátiles con conexiones a redes celulares, con lo que las diferencias en este aspecto van a ser mínimas e irrelevantes en un plazo corto de tiempo.

2.1.8 Gran variedad de terminales

Uno de los aspectos más característicos de las redes móviles es la gran variedad de fabricantes de terminales y modelos que existen en el mercado. Además las diferencias entre modelos pueden ser realmente importantes. Si comparamos un dispositivo de hace dos años con uno actual, las diferencias entre las respectivas capacidades son sencillamente abismales. También se producen grandes contrastes entre terminales contemporáneos de gama alta y gama baja.

Esta gran variedad de modelos y capacidades introduce una gran complejidad en el desarrollo de aplicaciones y dificulta el óptimo aprovechamiento de las posibilidades de los modelos.

3.2 Consecuencias en el desarrollo de aplicaciones cliente-servidor

En este apartado vamos a tratar de analizar las características de la transmisión de datos en redes móviles desde el punto de vista del desarrollo de aplicaciones en las que se intercambia información con servidores. Así, vamos a tratar de ver las consecuencias y los efectos que producen estas propiedades vistas en el apartado anterior a la hora de transmitir información de todo tipo entre el terminal y el servidor correspondiente [26-30].

2.2.1 Consideraciones de eficiencia en el uso del canal

Como ya hemos visto el espectro y por lo tanto el ancho de banda es un recurso escaso y caro. Por lo tanto en todas las transmisiones que realicemos sobre un canal móvil deben de estar lo más optimizadas posible. De esa forma el usuario tendrá que pagar menos por usar nuestra aplicación, el operador gastará menos recursos de la red para una misma aplicación y la aplicación tendrá unos tiempos de respuesta mejores.

Esto son varios casos que a modo de ejemplo ilustran las diferentes decisiones de diseño que ayudarán a que nuestra aplicación sea eficiente en el uso del ancho disponible:

- En caso de querer transmitir datos textuales entre aplicaciones, hay que evitar utilizar XML. Primero porque su lectura gastará bastante CPU en el terminal, y segundo y más importante

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

porque mete una sobrecarga no necesaria. Hay que buscar formatos sencillos pero que a la vez ocupen el menor espacio posible. En caso de datos muy complejos, podría merecer utilizar XML, pero siempre sería conveniente que las etiquetas fueran realmente cortas. En los siguientes cuadros se puede ver la misma información codificada de diferentes formas más o menos óptimas en su transmisión.

```
<?xml version="1.0" encod-
ing="UTF-8"?>
<ChessGame>
      <PlayerOneName>John</Play-
erOneName>
      <PlayerTwoName>Peter</Play-
erTwoName>
      <BoardOfTheGame>BXBABBZBBBB
BBPRRBBBBBBBBBBBBBB</BoardOfTheG
ame>
      <MoveNumber>23</MoveNumber>
      <GameTime>12:34</GameTime>
      <Turn>Player1</Turn>
</ChessGame>
```

```
<?xml version="1.0" encod-
ing="UTF-8"?>
<CG>
      <p1>John</p1>
      <p2>Peter</p2>
      <b>BXBABBZBBBBBBPRRBBBBB
BBBBBBBBBB</b>
      <m>23</m>
      <g>12:34</g>
      <t>Player1</t>
</CG>
```

```
John;Peter;BXBABBZBBBBBBPRRBBBBBBBBBBBBBB;23;12:34;Player1
```

- Si estamos en una aplicación basada en WAP, hay que cuidar mucho las páginas WML. Una de las funcionalidades más interesantes para minimizar los datos transmitidos son los datos almacenados en las variables WML del navegador, tal y como se puede ver en el siguiente ejemplo.

Figura 1 <?xml version="1.0" encoding="ISO-8859-1"?>
Figura 2 <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
Figura 3 <wml>
Figura 4 <template>
Figura 5 <do type="options" label="Menu">
Figura 6 <go href="index.jsp"/>
Figura 7 </do>
Figura 8 <do type="prev" label="Atras">
Figura 9 <prev/>
Figura 10 </do>
Figura 11 </template>
Figura 12 <card title="Compra">
Figura 13 <p>Escribe finalmente para la compra del $(producto) tu número de tarjeta
Figura 14 <input name="tarjeta"/>
Figura 15
Figura 16 y la caducidad de la misma

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
Figura 17          <input name="mes"/>/<input name="anyo"/>
Figura 18          <a href="#c" title="Comprar">Comprar</a>
Figura 19        </p>
Figura 20      </card>
Figura 21      <card title="Confirmacion">
Figura 22        <p>¿Estas seguro de la compra?<br/>
Figura 23          <anchor title="si">Sí<go href="comprar.jsp">
Figura 24            <postfield name="tarjeta" value="$(tarjeta)"/>
Figura 25            <postfield name="numero" value="$(numero)"/>
Figura 26            <postfield name="productoID" value="$(productoID)"/>
Figura 27            <postfield name="mes" value="$(mes)"/>
Figura 28            <postfield name="anyo" value="$(anyo)"/>
Figura 29          </go>
Figura 30          </anchor>
Figura 31          <anchor title="no">No<prev/>
Figura 32          </anchor>
Figura 33        </p>
Figura 34      </card>
Figura 35    </wml>
```

Otras buenas normas de uso es utilizar los elementos *template* para agrupar los botones y utilizar nombres cortos en todo tipo de variables y parámetros.

Al final, la opción más eficiente para reducir la cantidad de datos transmitidos es usar si es posible una codificación del XML. En este sentido se ha definido el estándar WBXML (*Wireless Binary XML*) que permite reducir la sobrecarga que introduce XML en las comunicaciones móviles siendo usado en WML y WMLScript.

2.2.2 Consideraciones sobre la calidad de la transmisión

Ya hemos visto que el ancho de banda en una red móvil puede sufrir desvanecimientos, sobre todo si el usuario está en movimiento. Estas caídas en la calidad de la transmisión afectan principalmente a las aplicaciones “en tiempo real”, como por ejemplo juegos multijugador o *streaming* de contenidos multimedia.

Las únicas medidas posibles para solventar este tipo de comportamientos son crear *buffers*, almacenes de datos intermedios de los que tirar en caso que la comunicación se resienta. Evidentemente, esto no siempre es posible puesto muchas veces los datos se tienen que entregar y procesar en el mismo instante en el que se generan.

2.2.3 Consideraciones sobre conmutación de circuitos y conmutación de paquetes

En la actualidad, los datos transmitidos cada vez van más por redes con conmutación de paquetes y esta tendencia va a seguir imparable en los próximos años. Aunque conmutación de paquetes tiene grandes ventajas como compartición de recursos, mayores velocidades y facturación por tráfico, no siempre es la mejor opción para todas las aplicaciones

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En el siguiente gráfico podemos ver cómo se puede comportar el ancho de banda de una transmisión móvil por conmutación de circuitos y por conmutación de paquetes. La clave reside en que mientras que en una de ellas el canal de datos está reservado en la otra se comparte.

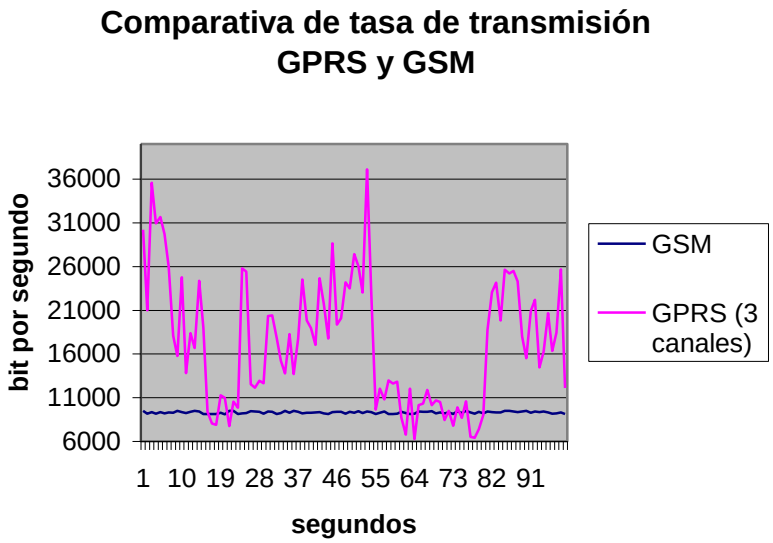


Figura 0.25: Comparativa de tasa de transmisión en GPRS y GSM

El problema con la compartición de los canales es que en caso de que haya muchos usuarios transmitiendo simultáneamente en un momento dado se produce un efecto "desvanecimiento" que provoca una caída en la velocidad de la transmisión. En contrapartida, el hecho de compartir la red con otros usuarios permite que se puedan llegar a mayores tasas de transmisión puesto que se optimiza el uso de la red [31-34].

En general es mejor el uso de la conmutación de paquetes por las mayores velocidades que ofrece y por la posibilidad de facturar por tráfico. Es especialmente recomendable para aplicaciones en las que el tráfico sea discontinuo y no haya consideraciones de tiempo real. En cambio cuando nos enfrentamos a aplicaciones con tráfico continuo y en tiempo real, como *streaming*, las conclusiones no son las mismas. Si con el ancho de banda que nos ofrece la conmutación de circuitos nos vasta, ésta sería la mejor opción. Como esto no suele ser así normalmente se utiliza la conmutación de paquetes, que aunque suele provocar caídas en el flujo de datos, en general da una considerablemente mayor velocidad de transmisión.

2.2.4 Consideraciones sobre seguridad

Como ya hemos visto la seguridad en las comunicaciones celulares digitales es realmente alta. Pero no debemos dar por solucionado el asunto por este motivo. Esta seguridad adicional debemos tenerla

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

en cuenta, pero tenemos que ser conscientes que sólo funciona en un tramo de las comunicaciones inalámbricas.

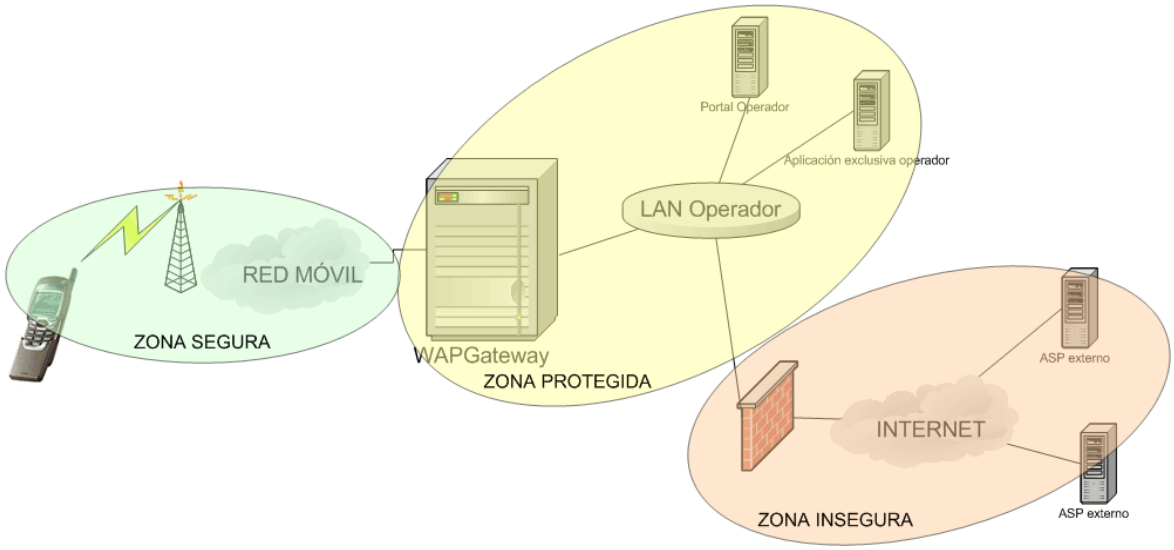


Figura 0.26: Zonas de seguridad en las aplicaciones móviles

Cuando diseñemos una aplicación móvil con acceso a servidores debemos ser conscientes por el número de redes que pasa. Debemos plantearnos qué grado de seguridad queremos. Como se puede ver en el gráfico superior, en el caso de aplicaciones de mensajería o comunicaciones con todas las máquinas servidoras residentes en redes protegidas del operador móvil, la seguridad se puede considerar alta pues en ningún tramo está expuesta a Internet ni a posibles pinchazos (aunque la red del operador no resulta 100% segura).

En el momento que las comunicaciones salgan fuera de la red del operador la cosa cambia y si queremos más seguridad tendremos que buscarla. Una solución más o menos sencilla es establecer un servidor intermedio que establezca comunicaciones seguras con el servidor externo [35-38].

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

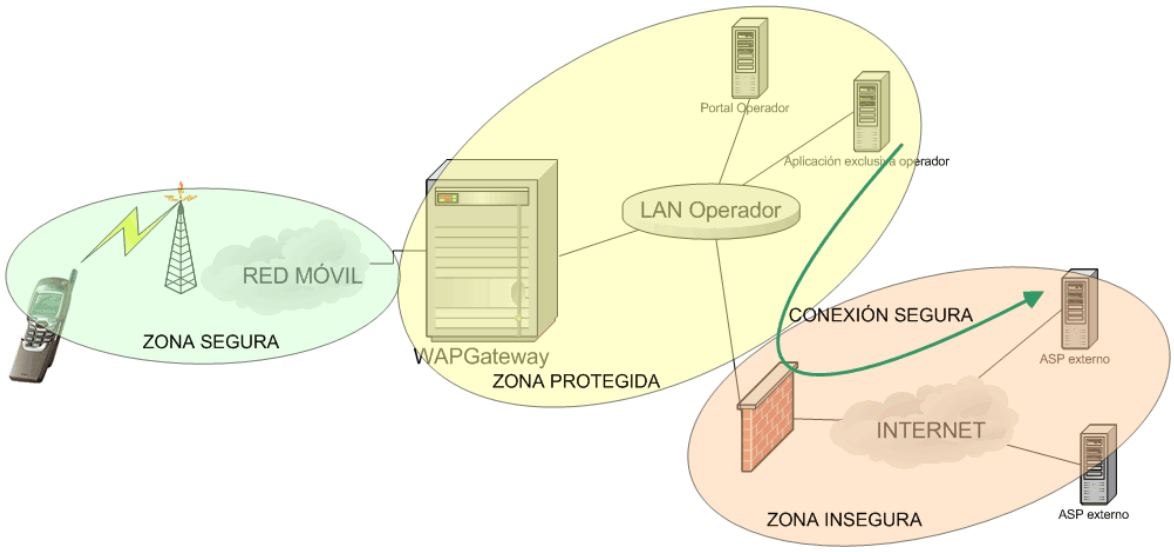


Figura 0.27: Soluciones seguras para las aplicaciones móviles

Aún así esta seguridad puede resultar demasiado limitada para determinadas operaciones de comercio electrónico. En ese tipo de aplicaciones lo óptimo y necesario es realizar una conexión cifrada punto a punto. Y es aquí donde ha estado uno de los problemas más graves en las aplicaciones móviles. Primero porque WAP 1.0 ponía como condición necesaria el uso de un *WAP Gateway* que limitaba la seguridad punto a punto. Segundo porque J2ME MIDP 1.0 tampoco incluía conexiones cifradas.

Una vez detectado este problema se han puesto soluciones en las nuevas versiones de los estándares. El primero que incorporó las conexiones cifradas punto a punto fue el estándar japonés *i-appli*. Posteriormente, J2ME MIDP 2.0 también lo ha incorporado así como WAP 2.0, que evitando el uso del WAP *Gateway* permite las conexiones seguras punto a punto [39-42].

2.2.5 Consideraciones de optimización para los terminales

La gran variedad de modelos de terminal móvil, así como su constante evolución y su gran heterogeneidad provocan que sea realmente difícil realizar una aplicación que esté pensada para un amplio número de modelos y simultáneamente esté lo más optimizada posible para cada uno de ellos.

En el caso de querer realizar una aplicación que llegue a un gran número de modelos y por lo tanto de usuarios, lo más normal es implementarla para funcionar en el caso peor, es decir para que funcione bien en los terminales con capacidades más restrictivas, por lo que la optimización es imposible.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

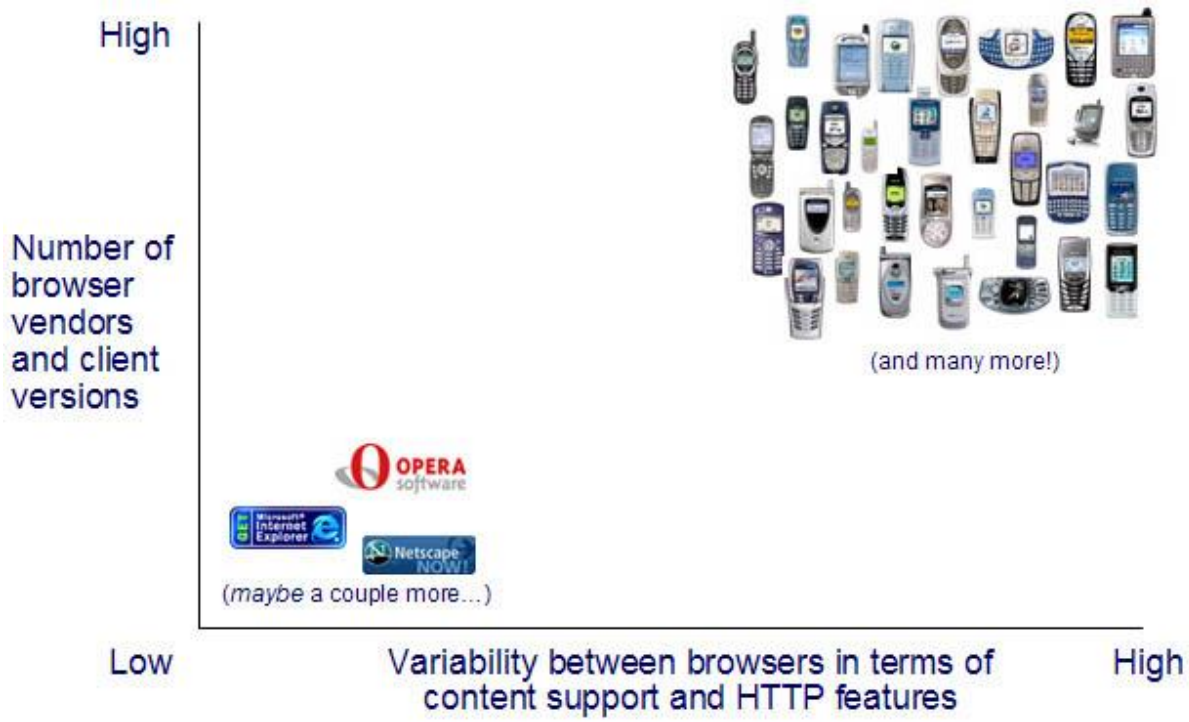


Figura 0.28: Comparación entre la variedad de terminales en el mundo fijo y el móvil

Fuente: OMA

Cuando hablamos de capacidades de terminales, hablamos de cuestiones como la potencia del procesador, la memoria disponible, la implementación de la tecnología utilizada, y sobre todo las capacidades gráficas de la pantalla del terminal. Este último tipo de capacidades son seguramente la más importante a la hora de desarrollar toda clase de aplicaciones, tanto de mensajería multimedia, como páginas WAP o aplicaciones J2ME.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

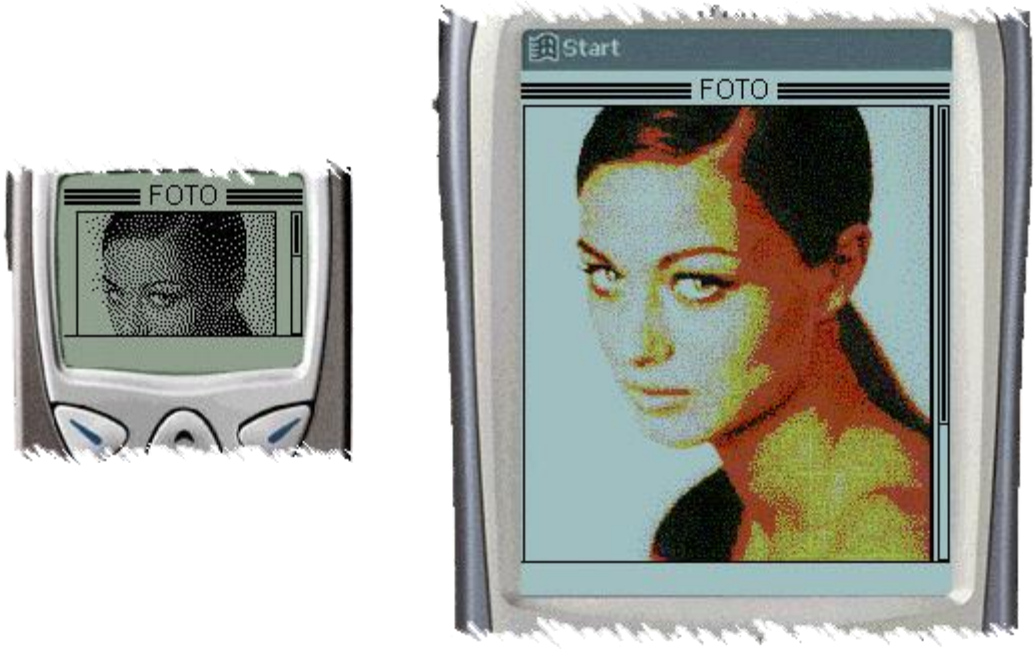


Figura 0.29: Diferencias en la visualización de imágenes en distintos terminales

Las posibles soluciones para este tipo de problemas son dos desde el punto de vista del proveedor de la aplicación:

- Establecer familias de terminales y realizar diferentes versiones de la aplicación para cada una de ellas manteniendo un núcleo común de código.
- Adaptar la aplicación en tiempo real según las características del terminal. Para esto se puede optar por analizar las características mediante el sistema UAProf que analizaremos después y que permite saber las propiedades del terminal.

Por otra parte los fabricantes de terminales y equipos, a través del consorcio OMA ya han recogido estos inconvenientes en diversos estudios y tecnologías. La primera solución optada fue el estándar UAProf, que permitía conocer todas las características del terminal en tiempo real. Evidentemente, esto facilitaba mucho la adaptación de los contenidos.

Aún así esa solución ha distado de ser considerada satisfactoria por varias razones:

- Gran complejidad de muchas adaptaciones, sobre todo de contenidos multimedia
- Son necesarias tecnologías y conocimientos muy avanzados que no siempre tienen los proveedores de contenidos
- Alto coste de creación de contenidos móviles
- Imposibilidad de conseguir adaptaciones correctas por la gran variedad de terminales

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

La línea de acción que está siendo más estudiada actualmente es intentar homogeneizar lo más posible los terminales y el software que incorporan para facilitar el desarrollo de aplicaciones presentando de forma parecida las mismas páginas.

3. MODELOS DE NEGOCIO EN LA TRANSMISIÓN DE INFORMACIÓN EN ENTORNOS MÓVILES

La transmisión de datos en entornos móviles es considerada por muchos la siguiente revolución tanto en Internet como en el mundo móvil. Pero todas las aplicaciones desarrolladas con este fin, se encontrarán y se encuentran con modelos de negocios con particularidades propias y en algunos casos con grandes diferencias con el mundo de Internet [43].

La principal diferencia es la presencia omnipresente y dominante del operador móvil. Es el agente con mayor fuerza en la cadena de valor y el que mayor interés tiene en concentrar todas las aplicaciones y todos los contenidos. Su principal ventaja estratégica reside en su total control sobre la red de acceso y las facturas de los usuarios así como la falta de competidores, otros operadores móviles del mismo país cuyo número suele ser extremadamente reducido. Como veremos más adelante, el modelo más exitoso de transmisión de datos con el móvil corresponde al sistema i-mode, desarrollado y controlado por el operador que más dominio ha ejercido a sus proveedores y a sus clientes, el japonés NTT-DoCoMo.

Además el mundo móvil tiene otras características que modifican los esquemas estratégicos como son la posibilidad de vender terminales preconfigurados con las conexiones del operador de turno, o la habitualidad con que los usuarios aceptan que los operadores sean los que vendan terminales y provean aplicaciones para ellos.

Con estas fortalezas estratégicas los operadores móviles, cuyo negocio original era la provisión de la red, empiezan a posicionarse en otros puntos de la cadena de valor como los portales móviles, la provisión de contenidos o la fabricación y venta de terminales.

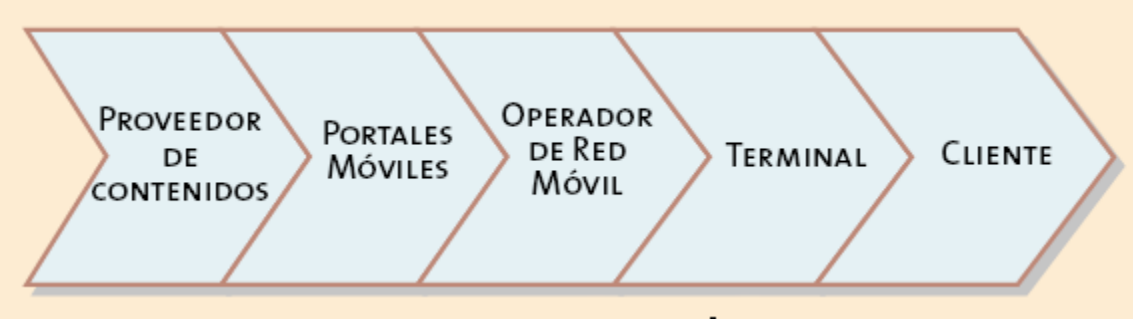


Figura 0.30: Cadena de valor en la transmisión móvil de información

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En este apartado vamos a ver primero los diferentes modelos de negocio posibles entre los proveedores de contenidos y los operadores para la provisión de aplicaciones y contenidos. También analizaremos los modelos que se han desarrollado para el acceso a estos contenidos, haciendo especialmente hincapié en las correspondientes comparaciones entre el éxito de i-mode, y el fracaso de WAP.

3.1 Modelos de negocio en la provisión de contenidos

En la provisión de contenidos en la industria de la telefonía móvil podemos encontrar los siguientes roles:

- Cliente (usuario de telefonía móvil con cierta avidez de contenidos)
- Operador (proveedor de la conexión y de los elementos de red necesarios para la provisión de contenidos, como por ejemplo el servidor de descargas o el centro de mensajes multimedia)
- Proveedor del servicio (agente que se encarga de proporcionar el portal desde el que se a realizar el descubrimiento de los contenidos por parte del usuario y que prepara la descarga)
- Proveedor de contenidos (agente que se encarga de proporcionar los contenidos para su posterior uso)

Según las empresas cumplan los tres últimos roles podremos establecer los modelos de negocio o escenarios posibles en la provisión de contenidos:

- En el primer modelo de negocio, el proveedor del contenidos se convierte también en proveedor del servicio (o viceversa), y el operador se reservaría el elemento de red que facilitaría la descarga. De esa forma ésta se produciría de forma segura y el servicio tendría una facturación eficiente y sencilla. Evidentemente se tendrían que llevar a acuerdos con los operadores para compartir los ingresos correspondientes a los contenidos. En este modelo se podría dar el caso que los proveedores de servicios alquilasen al operador el servicio de descargas o que se fuese a un modelo de compartición de beneficios.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

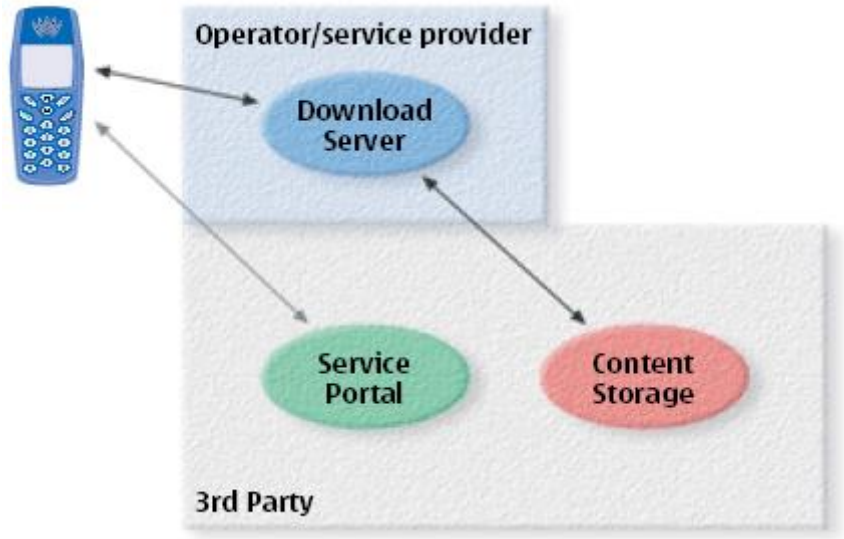


Figura 0.31: Modelo de negocio con el operador como *carrier* Fuente: Nokia

- En un posible segundo modelo de negocio, los operadores actuarían como agregadores de contenidos. Además del mecanismo de descarga y facturación aportarían al sistema el portal del servicio que, de hecho, podría ser común a varios proveedores de contenidos. De esta forma el operador podría conseguir la exclusividad del servicio frente a usuarios de otros operadores. Además el modelo sería más eficiente económicamente puesto que el portal sólo tendría que ser desarrollado y mantenido una sola vez y lo podrían utilizar los proveedores de contenidos que autorizase el operador. Evidentemente se tendrían que llevar a acuerdos con los operadores para compartir los ingresos correspondientes a los contenidos.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

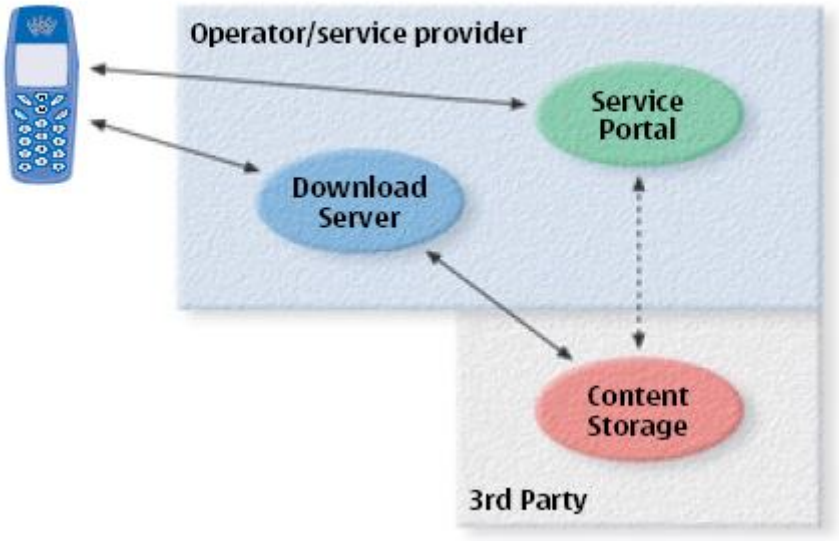


Figura 0.32: Modelo de negocio con el operador como agregador de contenidos Fuente: Nokia

- Finalmente en el tercer modelo, el operador crea o compra los contenidos y provee totalmente el servicio. En este escenario es posible que la compartición de beneficios no fuese la opción a elegir sino que el operador pagaría por los contenidos de forma independiente al éxito de estos posteriormente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

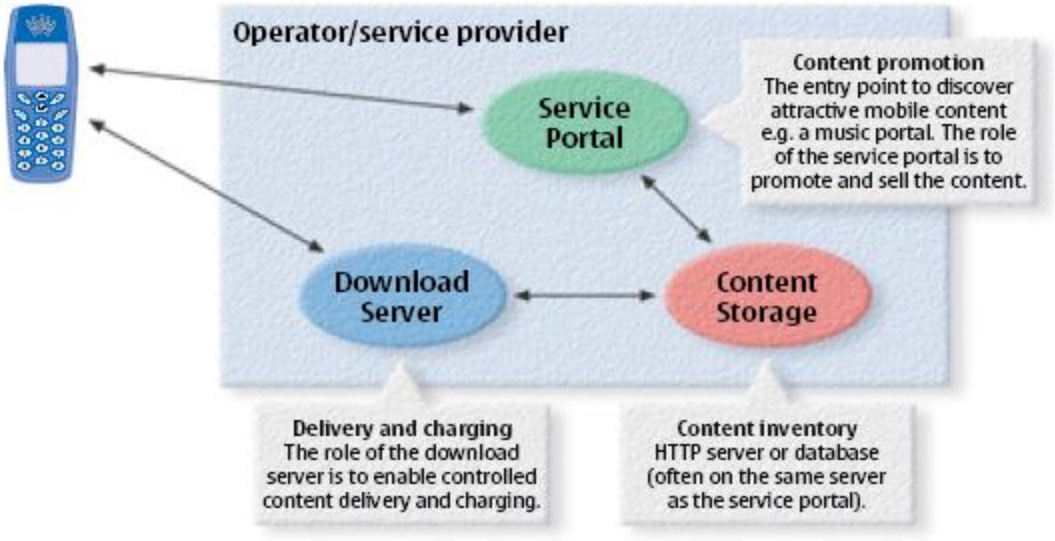


Figura 0.33: Modelo de negocio con el operador como proveedor de contenidos Fuente: Nokia

3.2 Modelos de negocio en el acceso a los contenidos

En este apartado vamos a tratar de analizar por qué el modelo tecnológico y empresarial de NTT-DoCoMo (propietaria de i-mode) ha resultado ser muy superior (por lo menos en resultados) que el desarrollado por la operadoras que han apostado por WAP. Primero analizaremos a fondo la estrategia y el éxito de i-mode para posteriormente compararlo con WAP y finalmente, para sacar las conclusiones pertinentes.

3.2.1 I-mode

No se puede simplificar el concepto de i-mode y por lo tanto no es fácil definirlo en pocas palabras. Va más a allá de un conjunto de servicios o una especificación técnica, se podría decir que es un modelo de explotación integral del acceso a contenidos de Internet mediante el móvil. También se podría decir que es una forma o filosofía de llevar la tecnología hasta los usuarios.

Nace en 1999 de la mano de la operadora móvil japonesa NTT-DoCoMo. De forma resumida y como veremos incompleta, se puede decir que nace un servicio similar a WAP en el que el lenguaje de programación es c-HTML, un lenguaje de marcado reducido basado en HTML y en el que las comunicaciones se basan en estándares propietarios que permitían en esa época velocidades de hasta 9,6 kbps mediante conmutación de paquetes.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

A partir de esa fecha inicial, el crecimiento ha sido espectacular. No sólo en clientes subscritos, sino también en el aumento de los servicios ofrecidos y en la mejora de las tecnologías utilizadas. Aparecen los teléfonos a color, con soporte para i-appli (un J2ME mejorado), cámaras de fotos, nuevos servicios como i-motion (videos) o iarea (localización). También mejoran substancialmente las comunicaciones llegando a 28,8 kbps e incluso a 384 kbps con FOMA, la apuesta 3G de NTT-DoCoMo.

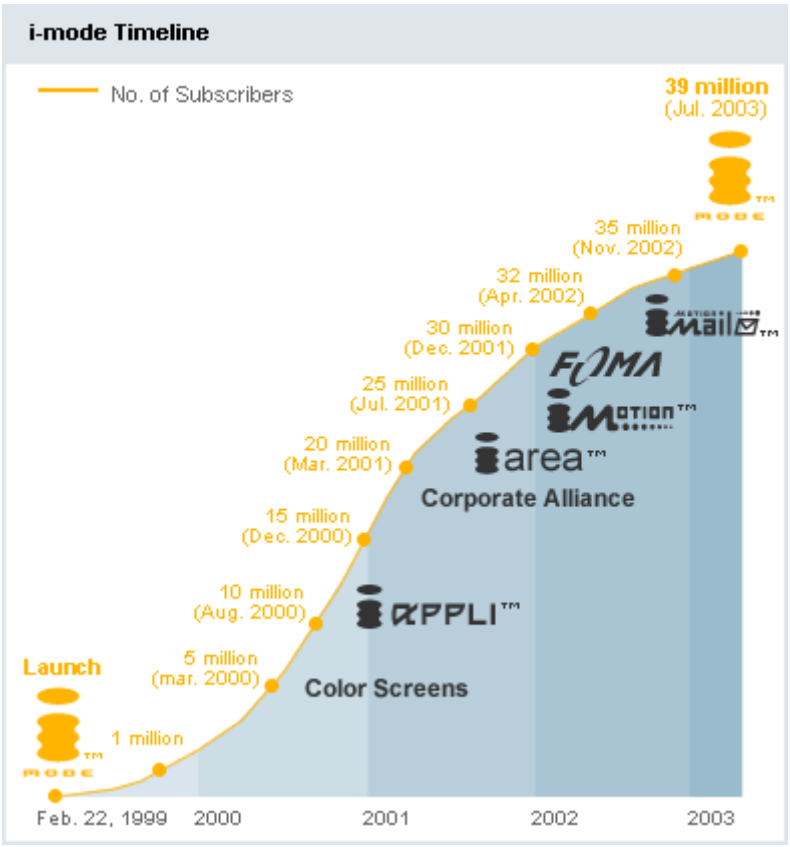


Figura 0.34: Evolución de los clientes y servicios de i-mode Fuente: NTT-DoCoMo

No vamos a detallar en profundidad las tecnologías utilizadas ni los servicios publicados. Nuestro interés en este apartado es ver cómo un modelo de negocio puede significar un éxito rotundo. Para ello vamos a analizar como funciona el modelo de negocio de i-mode para luego estudiar las razones del éxito abrumador de i-mode en Japón [44].

I-mode opera bajo un modelo de negocio en el que el usuario, después de darse de alta con una cuota de 300 ¥/mes (2,6 €/mes), puede acceder a cualquiera de los más de 3.000 sitios registrados oficiales de más de 900 proveedores de contenidos (el 30% ofrecen contenidos *premium* mediante suscripción), listados en orden de popularidad bajo diferentes menús y categorías. El usuario además abona

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

0,3 ¥/130 bytes (0,026 €/ 130 bytes) por el volumen de datos trasmitidos o recibidos (por ejemplo, un parte del tiempo puede costar en torno a un tercio de euro).

Al ser un servicio proporcionado directamente por el operador móvil, los subscriptores no necesitan facilitar un número de tarjeta de crédito para efectuar sus compras o abonarse a los contenidos *premium* (en los sitios oficiales). El coste de los servicios *premium* de los sitios oficiales (el coste suele estar entre 100-500 ¥/mes o 1,15-4,3 €/mes) y de los productos adquiridos aparece directamente en la factura del abonado y DoCoMo revierte este dinero al proveedor de contenidos, no sin antes retener para sí un 9% del total en concepto de comisión.

En el caso de los sitios no oficiales (más de 60.000), los contenidos o los productos que requieran de pago por parte del usuario deberán ser abonados mediante tarjeta de crédito o cualquier otro mecanismo decidido y proporcionado por el sitio no oficial. En este caso DoCoMo sólo recibe el dinero correspondiente al tráfico generado. Además estos sitios no tienen porque cumplir las estrictas normas del operador y pueden ofertar contenidos no autorizados (contenidos para adultos, ejemplo) y además pueden establecer tarifas de suscripción mayores de 500 ¥. De todas formas, más de dos tercios de estas páginas son gratuitas.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

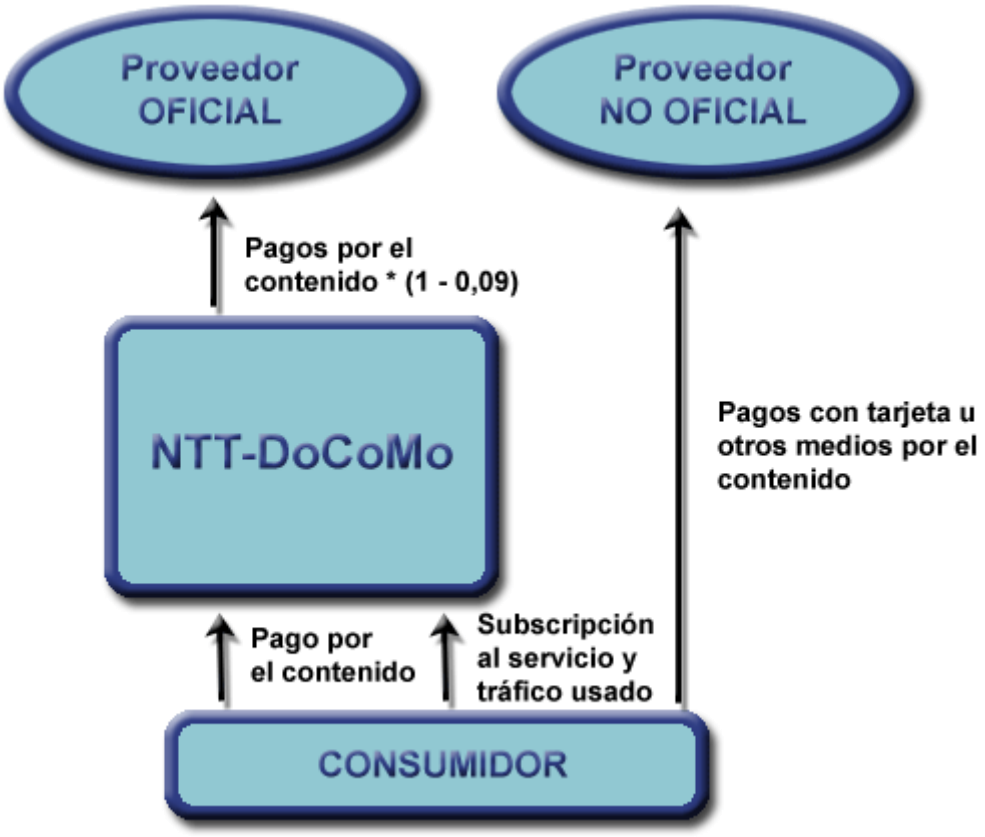


Figura 0.35: Diagrama del flujo del dinero en los servicios i-mode

Las razones del éxito de NTT-DoCoMo son varias, aunque la mayoría se pueden resumir en la capacidad de la compañía para coordinar todas las etapas de la cadena de valor y responder a las necesidades del cliente final. Podemos dividir las razones del éxito de i-mode entre las achacables al operador y las causadas por las especiales características del mercado. Empecemos por las primeras.

- NTT-DoCoMo siempre ha dado una alta prioridad a la simplicidad y usabilidad de los servicios sobre la innovación. Las decisiones técnicas siempre estaban supeditadas a la necesidad de desarrollar aplicaciones que fuesen útiles a los usuarios desde el primer momento.
- NTT-DoCoMo ha mantenido un férreo control sobre los terminales del servicio. Siempre ha procurado que fuesen los terminales los que se adaptasen a los contenidos, y no al contrario. La compañía ha trabajado de forma muy cercana con los proveedores de los terminales, especificando todas las características técnicas de los dispositivos. De esta forma, los terminales se comercializan bajo la marca de NTT-DoCoMo, aunque la primera letra del modelo indique el fabricante (N503i es de Nec, F503i es de Fujitsu). Finalmente, cuando se lanzó el servicio ya se contaba con un buen número de terminales en manos de los usuarios pues se había subvencionado fuertemente los teléfonos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- La operadora japonesa ha creado un departamento con más de 60 personas para ya ayudar a los proveedores de contenidos y educar al usuario final sobre las oportunidades de negocio de la plataforma i-mode.
- Además, también utiliza a un riguroso proceso de selección para dar la calificación de "oficial" a una página i-mode. Aún así no desanima a los 1.000-6.000 candidaturas mensuales que recibe la compañía y provoca que haya actualmente una lista de espera de más de 6 meses.
- Además del control sobre los fabricantes de terminales y los proveedores de contenidos, NTT-DoCoMo también mantiene unas fuertes relaciones con los canales de distribución. Por un lado dispone de una importante red de tiendas propias que proporcionan un canal directo óptimo. Pero además también se relaciona intensamente con los canales indirectos, todas las tiendas de electrónica, extremadamente populares en Japón. Para motivar la suscripción al servicio, NTT paga a estas tiendas 17 € por cada usuario que se registra al servicio cuando compra un terminal. Esta comisión es superior a lo que se suele dar en Europa o América y provoca que los comercios estén realmente motivado en la venta del servicio con lo que se estima que entre el 20% y el 40% de los usuarios de i-mode sólo usa el terminal para servicios de voz.
- Una de las facetas que más ha apoyado NTT-DoCoMo es el de las alianzas estratégicas con empresas líderes. Claros ejemplos son las alianzas con AOL, Sony o Disney.
- La elección del mercado objetivo y la forma de promocionar el servicio han sido perfectas. Por un lado optaron por orientarse a un público joven, menor de 30 años, con menor resistencia al cambio y mayor costumbre con la tecnología (mediante i-mode un usuario medio envía 8 e-mails y visita 11 páginas al día). Además nunca promocionaron el servicio como Internet en el mundo móvil, cualquier referencia a "tecnología", "web" o "navegador" fue cuidadosamente evitada.

Evidentemente, Japón no es un mercado normal o comparable con el resto, especialmente en cuando hablamos de electrónica o entretenimiento. Estas especiales características han influido considerablemente en el éxito de i-mode.

- La penetración de los móviles era muy alta.
- Características de la cultura japonesa, largos desplazamientos en tren, perfil entusiasta ante las innovaciones tecnológicas, carácter cerrado de los japoneses que evitan contactos directos, etc..

3.2.2 WAP

WAP es sencillamente un conjunto de estándares. Incluyen una pila de protocolos para realizar las comunicaciones móviles y un entorno de ejecución de aplicaciones en el terminal. En ningún caso se entró en la forma de incorporar esta tecnología desde el punto de vista de negocio o marketing. Estos apartados siempre se reservaron para los operadores de cada país.

Cada operador tuvo la libertad (y la sigue teniendo) de lanzar las aplicaciones que quiera, al precio que quiera y en el momento deseado. Actualmente WAP sólo define los medios técnicos necesarios para hacerlo, un poco al estilo de Internet. Como podemos imaginar, la situación es muy diferente a la de i-mode. Los operadores han tenido unas relaciones mucho menores con los proveedores de

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

contenidos y los fabricantes de terminales. Además el modelo de negocio está totalmente abierto. Aún así, la mayoría de operadores han optado por cobrar la conexión según el tiempo o el tráfico utilizados y poniendo un suplemento (*premium*) a los contenidos más interesantes o valiosos. Las tarifas planas han brillado por su ausencia (aunque algún intento ha habido) por lo que los usuarios siempre han tenido que pagar proporcionalmente al uso de las aplicaciones.

Desde los primeros lanzamientos comerciales del año 2000, ningún operador ha conseguido encontrar la clave del éxito de las aplicaciones sobre WAP. Partiendo de las expectativas generadas en esa época, seguramente desmesuradas y poco realistas, para el año siguiente ya se empezaba a hablar de un rotundo fracaso de uso. Las razones de las dificultades que encontraron las aplicaciones WAP en su momento son varias:

- Al contrario que en Japón, donde había un gran número de terminales con soporte i-mode en su lanzamiento, los operadores europeos lanzaron servicios WAP antes que hubiese un mínimo número de terminales en manos de los usuarios. Además, durante bastante tiempo no hubo terminales con atractivo diseño, capacidades técnicas avanzadas y a un precio razonable.
- La orientación que se le dio en la mayoría de países fue Internet en el móvil. No sólo se buscó adaptar las aplicaciones de Internet al móvil sin aportar más valor añadido, sino que además se promocionó y publicitó de esa forma, llevando a la frustración y la decepción a los primeros usuarios.
- Falta de contenidos atractivos y servicios demandados por usuarios. Mientras NTT-DoCoMo se preocupaba de incluir contenidos interesantes y con un alta calidad, los operadores europeos han pecado de dejadez en este terreno.
- En un comienzo las comunicaciones se realizaron sobre GSM y conmutación de circuitos. Esto provocaba que los tiempos de acceso fuesen realmente altos y que se cobrase por el tiempo de uso con lo que los usuarios nunca se atrevieron a utilizar masivamente las aplicaciones.

Lo cierto es, que pasados varios años desde su lanzamiento WAP empieza a ver la luz. Por un lado las mejoras de las tecnologías, tanto de las de transmisión, como las de los terminales, han permitido poder realizar aplicaciones más interactivas, rápidas y espectaculares llegando además a un mayor número de usuarios. Además, los operadores han replanteado estrategias y actualizado expectativas analizando y actuando con mayor tranquilidad y realismo.

De alguna forma u otra, las aplicaciones de Internet tendrán que ser usadas a través del móvil. El número de usuarios de telefonía móvil (altísimo) y la mayor familiaridad de la ciudadanía con las aplicaciones de Internet han de provocar que tarde o más bien temprano, el uso de móvil para conectarse a Internet se extienda. Y WAP parece bastante bien posicionado para ser la tecnología utilizada en un buen número de países.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

DESCARGA DE CONTENIDOS

La descarga de imágenes, sonidos, películas, programas o documentos es uno de los mayores usos de Internet actualmente. El incremento del ancho de banda y el uso de PC como terminales de conexión han permitido que la cantidad y la variedad de las descargas de archivos de la red sean muy elevadas.

En el mundo móvil la situación es un poco diferente. Por un lado las descargas son menores en cantidad y calidad. Lo que más abundan son melodías e imágenes de baja calidad y tamaño para personalizar el terminal, pero esto se compensa con los resultados económicos de estas descargas. Estos ingresos han sido realmente espectaculares, tanto para los operadores como para los proveedores de los contenidos. De hecho, tal y como podemos ver en el siguiente gráfico, las descargas realizadas a través de teléfonos móviles suponen un negocio más de dos veces mayor que las correspondientes a los terminales PC.

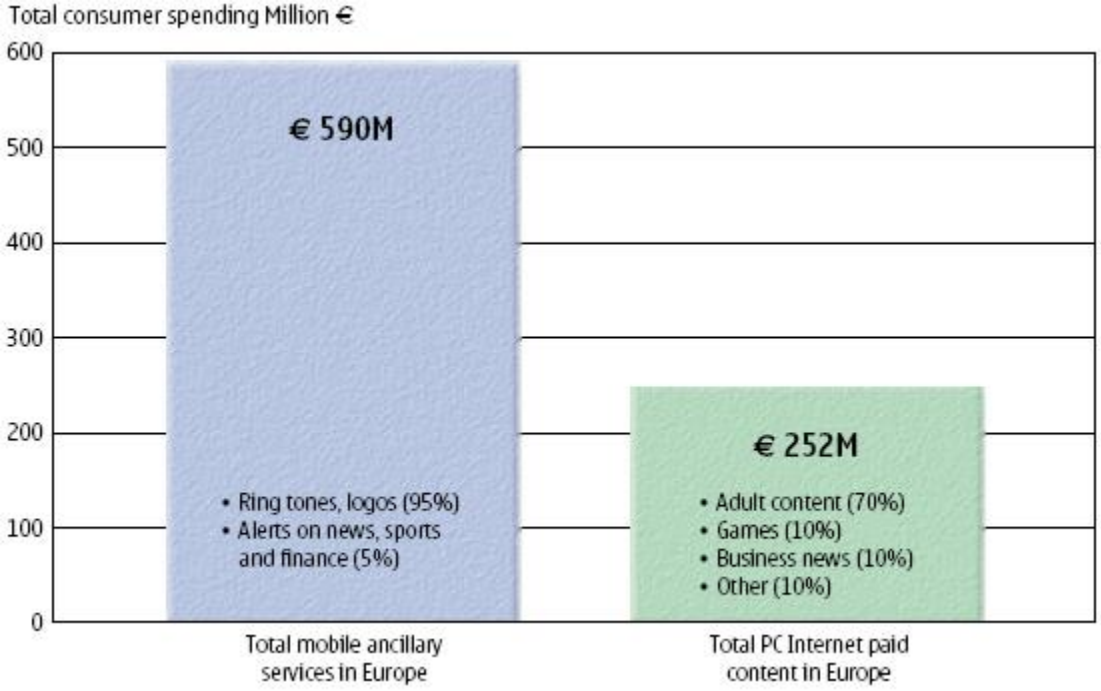


Figura 0.36: Ingresos en las descargas de contenidos Fuente: Jupiter

Las razones de este éxito han sido varias:

- Por un lado las facilidades tecnológicas que ofrece la telefonía móvil para este tipo de aplicaciones. Al usuario le resulta muy sencillo mandar un mensaje o realizar una llamada para recibir posteriormente un mensaje con el contenido deseado.
- Las facilidades que aporta el móvil a la hora de realizar micro-pagos o pagos de pequeña cuantía. Gracias a la posibilidad de incluir el cobro de los contenidos en la factura del usuario

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

ha resultado muy sencillo cobrar al usuario todas las descargas realizadas. Además ha permitido crear una cultura que no existe en Internet, que permite acostumbrar al usuario a pagar por los contenidos descargados. Esto va a tener gran importancia en las posibilidades de éxito de los futuros planes de negocio para los servicios móviles.

- El terminal móvil es un objeto personal que además se suele llevar permanentemente. Esto provoca que los usuarios quieran personalizar su terminal así como personalizan el resto de su imagen exterior. De esta forma es fácil comprender el uso de las descargas móvil con objeto de dotar al terminal de las imágenes y tonos más modernos o curiosos.
- Dificultad para compartir los contenidos entre los usuarios. En los primeros años de las descargas de tonos y logos era razonablemente complejo mandar estos contenidos con otros usuarios, esto provocaba que fuese más fácil y barato comprar por un precio bajo los archivos.

Estos positivos resultados de los últimos años, junto con la evolución constante de las capacidades de los terminales permiten pronosticar un importante éxito empresarial en el futuro. Además uno de los principales problemas que en el pasado han existido ha sido la gran variedad de tecnologías de descarga disponibles en el mercado. Nokia creó el *Smart Messaging* sobre los SMS para mandar contenidos, además también empezó a especificar una tecnología para la descarga segura sobre WAP. Otros fabricantes hicieron lo propio, por ejemplo, Openwave con su tecnología *DownloadFun* fue pionera en la descarga de contenidos mediante conexiones WAP.

Afortunadamente, tanto los fabricantes de teléfonos, como los desarrolladores de aplicaciones móviles y navegadores para los terminales se han puesto de acuerdo para empujar desde el principio el consorcio OMA (*Open Mobile Alliance*). Este consorcio tiene entre trabajos la especificación de las tecnologías para desarrollar aplicaciones basadas en descargas de contenidos desde un terminal móvil.

A simple vista, las tecnologías involucradas para la descarga de contenidos ya deberían estar superadas. En la WEB se producen todos los días millones de descargas mediante el protocolo HTTP y en el entorno móvil además ya existe la mensajería multimedia. Pero la utilización solitaria de estas tecnologías provoca tres grandes problemas:

- Un usuario puede intentar descargar un contenido que no cabe en su terminal o que no acepta el terminal porque el formato es demasiado avanzado. El usuario no tiene porque tener conocimientos sobre formatos gráficos y en caso de realizarse la descarga posteriormente no se podría instalar o utilizar el contenido. Esto provocaría que o bien se le cobra al usuario un contenido que no podrá usar (con la consiguiente mala experiencia para el cliente) o bien se genera un tráfico que luego no se factura. Cualquiera de estas situaciones no deseable en un entorno en el que se cobra el contenido o el tráfico o ambos.
- Un problema más grave es si el terminal sufre una desconexión a mitad de la descarga o por cualquier otra causa no puede descargar del todo el contenido e instalarlo si procede. En ese caso estaríamos en una situación parecida a la anterior siendo otra vez necesario no cobrar al usuario por un contenido que no podrá utilizar.
- Finalmente también sería deseable que se protegiesen los contenidos de pago para que el usuario no pudiese mandarlos fácilmente. Con lo terminales más avanzados estas posibilidades cada vez son más fáciles. Además, también sería muy interesante poder establecer polí-

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

ticas para el uso de los contenidos. Por ejemplo, sería muy interesante poder especificar cuántas veces se puede utilizar un contenido o durante cuánto tiempo, de esa forma se podrían hacer varias tarifas o realizar descargas de prueba a modo de previsualización.

Para solucionar estos puntos se han desarrollado nuevas tecnologías complementarias a WAP y a MMS. Por un lado se ha diseñado lo que se denomina *OMA Download*, que no es sino un conjunto de estándares para solucionar los dos primeros problemas antes comentados. En el próximo apartado analizaremos sus puntos más importantes.

En el segundo apartado entraremos a estudiar la solución que en el consorcio OMA se ha dado al tercer problema, la protección de los contenidos. La tecnología propuesta para realizar esta labor se denomina DRM (*Digital Rights Management*)

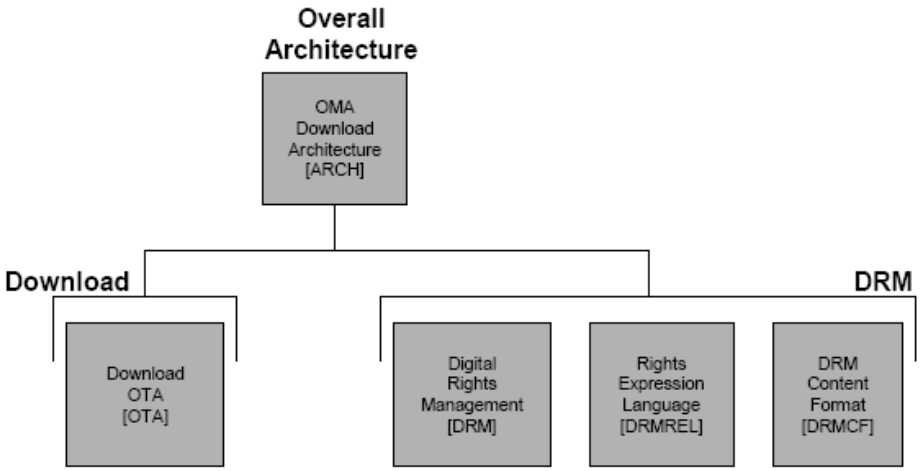


Figura 0.37: Arquitectura global de las descargas Fuente: OMA

3.3 OMA Download

Como hemos visto, la descarga de contenidos se debe realizar mediante tecnologías que permitan realizar transacciones robustas entre el terminal y el servidor de descargas con el fin de proporcionar al usuario la mejor y más sencilla experiencia de cliente posible y al operador o proveedor de servicios la posibilidad de cobrar el contenido de forma simple y con protección de copia.

Para realizar estas funciones la industria de la telefónica móvil, a través del consorcio OMA, ha definido el estándar *OMA Download* para facilitar la descarga de todo tipo de contenidos, como melodías MIDI o salvapantallas en formatos JPEG, GIF animados o PNG entre otros muchos tipos de formatos y contenidos. Las ventajas de esta tecnología son varias:

- Evita la fragmentación de las aplicaciones de descarga que se podría producir si todos los fabricantes de terminales implementasen sus propias tecnologías de descarga. En ese escenario los desarrolladores de aplicaciones de descarga tendrían que primero detectar qué terminal

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

está intentando descargar el contenido y luego enviárselo según la tecnología que correspondiese.
- Basada en la descarga de contenidos a través de HTTP, las tecnologías que involucra y la arquitectura que tiene son realmente sencillas, permitiendo desarrollar servicios de descarga en unos plazos de tiempo realmente cortos.
- Permite realizar extensiones sobre ella, como el caso de DRM, facilitando crear mecanismos de gestión de los derechos de los contenidos.

La primera tecnología de descargas y la más sencilla es la descarga HTTP. En ella el cliente realiza una petición HTTP al servidor indicándole el contenido a descargar y éste contesta mandando el contenido e indicando con el tipo MIME la clase de contenido que es. *OMA Download* se basa en este tipo de descargas para realizar las diferentes fases de la que se compone. Pero además, su arquitectura es muy similar a las descarga de aplicaciones J2ME, es decir a la tecnología *Java MIDLet Download.*

La tecnología *Java MIDLet Download* es más antigua que *OMA Download* y se creó para facilitar las descargas de aplicaciones J2ME. El modelo de uso es exactamente igual al de *OMA Download* con algunos cambios que iremos viendo según vayan apareciendo. Lo que es evidente, es que la descargas especificadas por OMA surgen de inspirarse en las descargas de *MIDLets* que se habían demostrado eficaces y sencillas de implementar.

Vamos a dividir el estudio de *OMA Download* en tres apartados. Primero veremos las diferentes arquitecturas que puede adoptar, después veremos las distintas fases de una descarga y luego veremos el formato del archivo que describe el contenido y sobre el cual gira toda la especificación. Durante todo el análisis de esta tecnología debemos tener en cuenta que para realizar la descarga el usuario estará navegando por páginas en las que se listarán todos los contenidos descargables. Estas páginas las generará el servidor de presentación que accedería al repositorio de contenidos para saber cuáles hay. Cuando el usuario decida descargarse alguno realizará una petición al servidor de descargas el cuál será el que implemente *OMA Download* y el cuál también accederá al repositorio de contenidos para facilitar la descarga.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

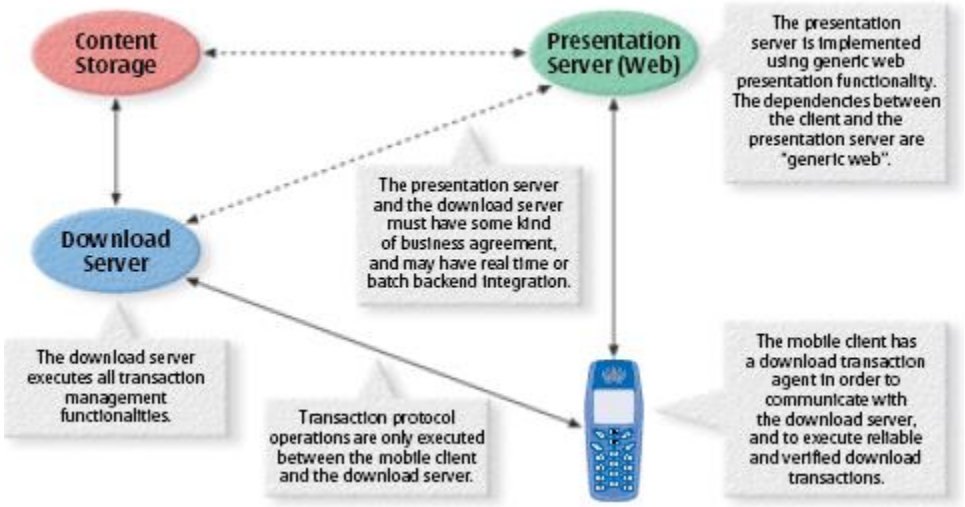


Figura 0.38: Elementos que intervienen en un proceso de descargas Fuente: Nokia

3.3.1 Arquitectura

OMA Download se basa en varias peticiones HTTP para realizar toda la descarga. En esas peticiones se realizan principalmente tres acciones:

- Descarga de un descriptor del contenido. Más adelante veremos cuál es el formato de este descriptor, pero como veremos tendrá información tal como el formato del contenido, su tamaño, etc.
- Descarga del propio contenido.
- Notificación al servidor de descargas de que el contenido ha sido correctamente descargado e instalado.

El último punto es optativo. Además los dos primeros se pueden realizar en dos peticiones HTTP diferentes o en una sola. De esta forma tendremos dos posibles arquitecturas.

- *OMA Download* con descarga separada del contenido y su descriptor

  En esta arquitectura cuando el usuario selecciona el contenido a descargar la petición que se realiza al servidor de descargas es para obtener el descriptor del contenido exclusivamente. En este descriptor se informa de la URL del contenido que se desea descargar. Si el terminal decide posible la descarga y el usuario la confirma se realizará otra petición con el objeto de descargar realmente el contenido.

  Finalmente, se realizará una petición al servidor de descargas indicando el éxito o fracaso de la descarga. Para realizar esta petición también se obtendrá la dirección del descriptor anteriormente obtenido.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

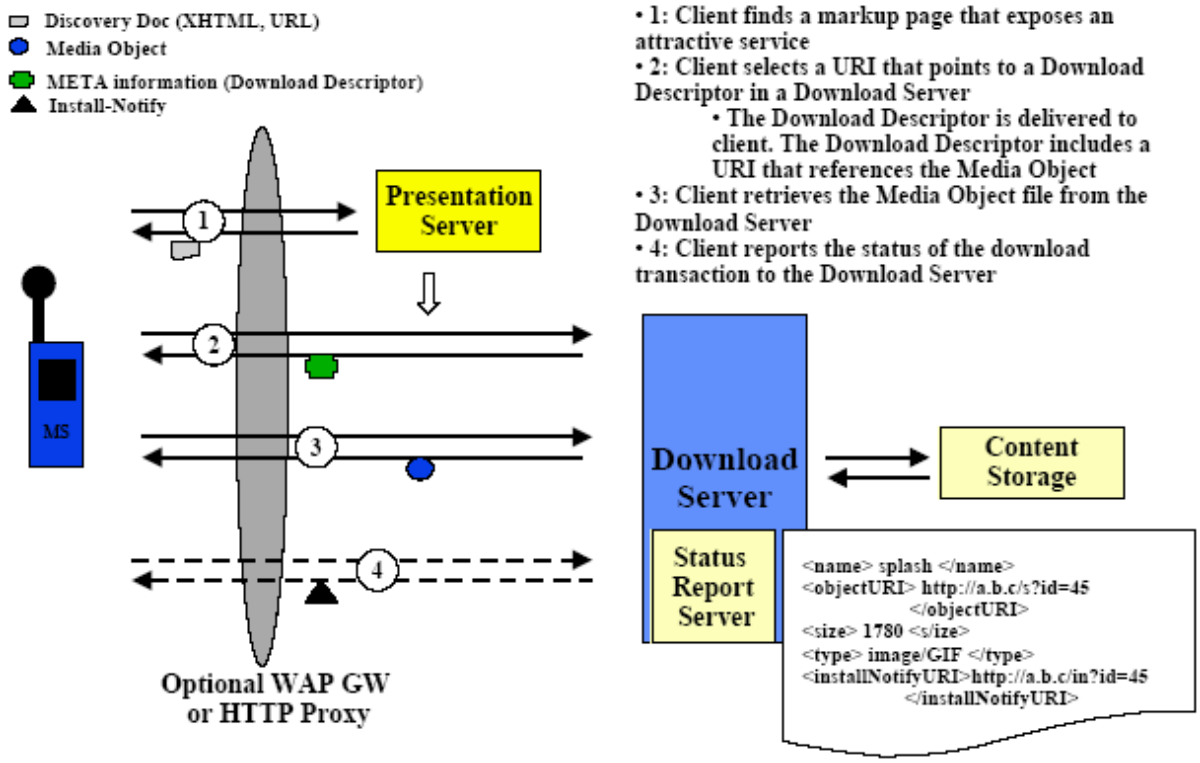


Figura 0.39: Arquitectura de descargas OMA con envío separado de descriptor y contenido Fuente: OMA

- *OMA Download* con descarga conjunta del contenido y su descriptor

En esta arquitectura el envío del contenido y su descriptor se realiza conjuntamente en una sola petición-respuesta HTTP. Este modelo es utiliza considerablemente menos, porque apenas aporta ventajas y supone perder parte de los beneficios esperados, como por ejemplo la comprobación previa del terminal sobre el formato y tamaño del contenido.

Para la realización del envío simultáneo del contenido con su descriptor se debe usar el formato *multipart/related* de HTTP.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

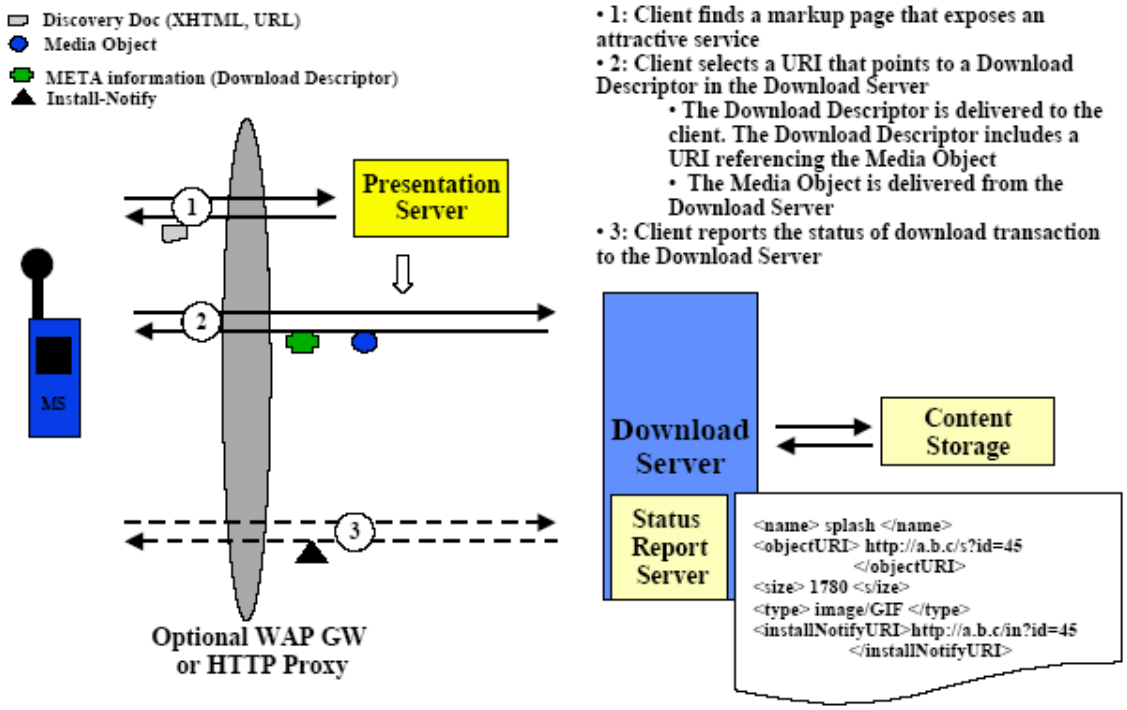


Figura 0.40: Arquitectura de descargas OMA con envío conjunto de contenido y descriptor Fuente: OMA

En ambas arquitecturas el descriptor contiene una URL contra la que al final se puede realizar una petición informando del estado final de la descarga. Aún así esta funcionalidad es optativa y se puede considerar la arquitectura sin necesidad de realizar esta petición. En este caso sacrificaríamos la confirmación de la descarga por una mayor optimización del ancho de banda usado.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

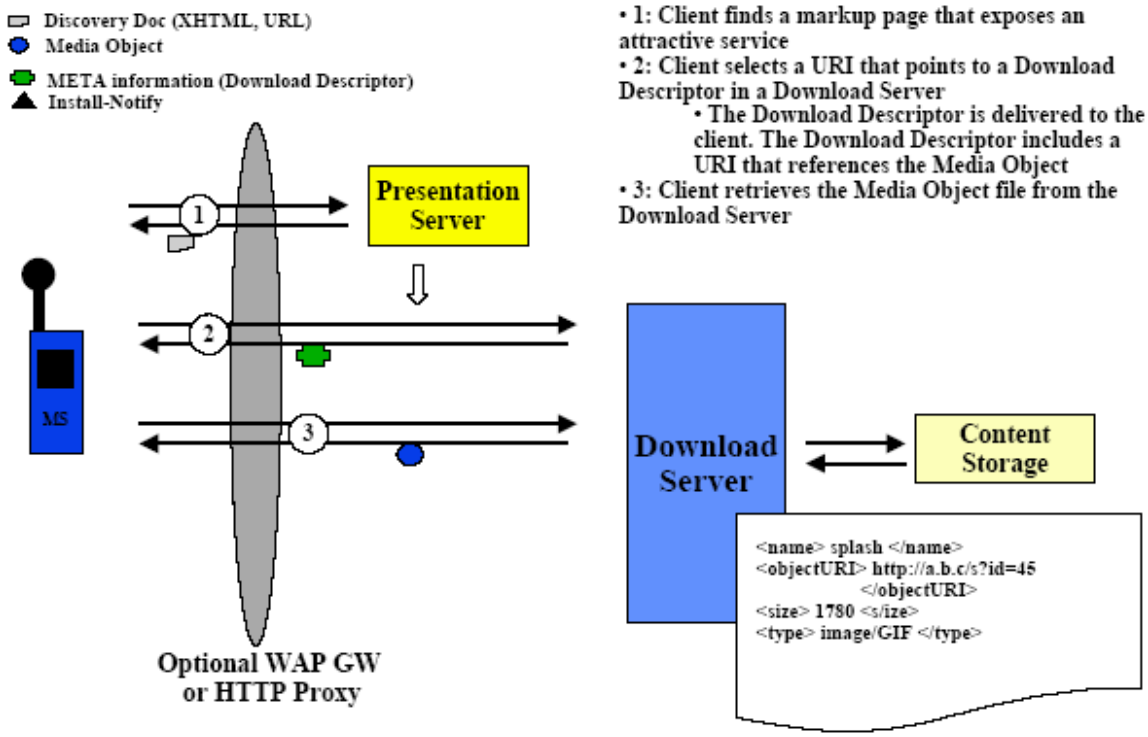


Figura 0.41: Arquitectura sin confirmación de descarga de contenido Fuente: OMA

3.3.2 Fases de la descarga

Durante todo el proceso de la descarga se pueden identificar diferentes fases en las que se van realizando diferentes acciones o peticiones para completar con éxito toda la transacción.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

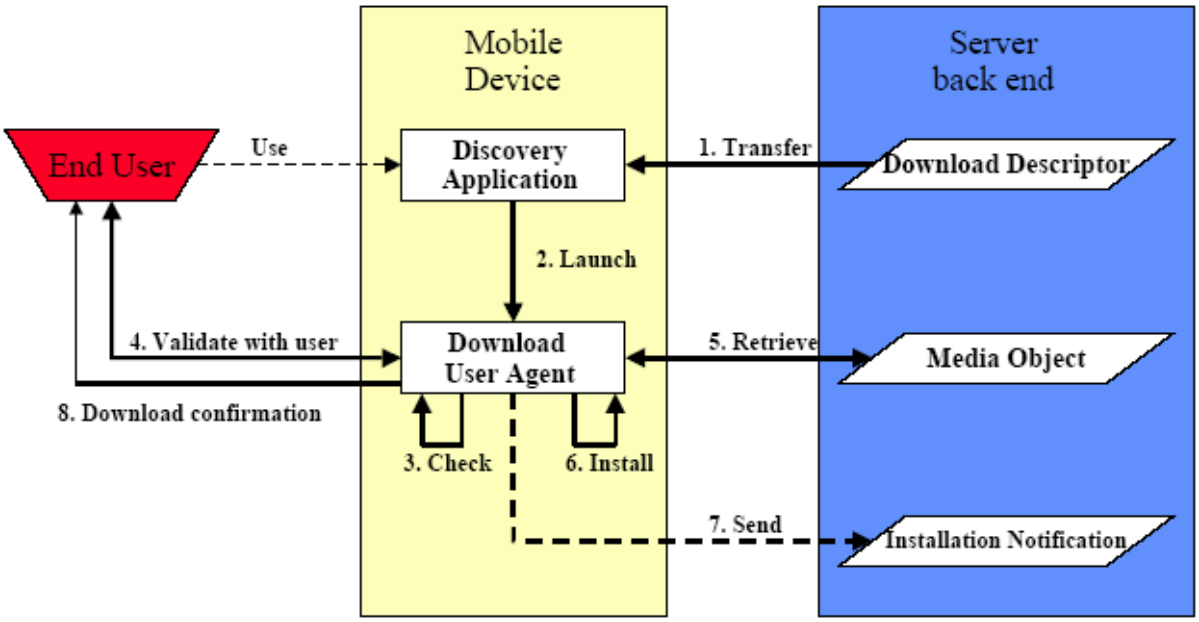


Figura 0.42: Fases de la descarga Fuente: OMA

A partir de la segunda fase, el terminal ya ha obtenido el descriptor del contenido. Con este descriptor se puede saber si hay que notificar el éxito de la descarga o cualquier error que suceda. En las siguientes pases posteriores a la primera cualquier error que suceda podrá ser notificado a la dirección de notificación mediante del envío de una petición HTTP POST con el número del error correspondiente y un texto descriptivo separado por un espacio del código de error. En la siguiente tabla se listan los errores más comunes y sus códigos de error.

Código del error	Mensaje del error
900	Success
901	Insufficient memory
902	User cancelled
903	Loss of service
905	Attribute mismach
906	Invalid descriptor
951	Invalid DDVersion
952	Device aborted
953	Non-Acceptable content
954	Loader error

Tabla 0.1: Lista de los errores posibles en el proceso de descarga OMA

- Fase 1: Descarga del descriptor del contenido

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El usuario después de navegar por las páginas que el servidor de presentación le provee escogerá un contenido a descargar. Cuando seleccione el enlace correspondiente se producirá una descarga HTTP, pero no del contenido, sino de su descriptor.

Como hemos visto el descriptor puede venir acompañado del propio contenido. En ese caso vendrá en una respuesta HTTP *multipart/related* en la que el descriptor vendrá primero y el contenido después. Las siguientes fases se ejecutarán igual con excepción de la número 5, descarga del contenido.

- Fase 2: Lanzamiento del agente de descargas y procesado del descriptor

En esta fase se lanza el programa en el cliente que gestiona las descargas y empieza su ejecución leyendo y procesando la información contenida en el descriptor. Cualquier error que se detecte en el descriptor parará la descarga y se notificará al servidor.

- Fase 3:Comprobación de capacidades del terminal

El terminal deberá comprobar si puede descargar e instalar el contenido. Chequeará si se dispone de suficiente memoria y si el formato es comprensible por el software instalado. En caso de que algo impida realizar la descarga con seguridad se notificará la cancelación de la misma indicando las causas.

En caso de problema, también es posible pasar a la siguiente fase para pedir al usuario confirmación indicando el problema encontrado.

- Fase 4: Confirmación del usuario

Una vez leído el descriptor del contenido y comprobado la utilidad del contenido, se debe preguntar al usuario si desea descargar el archivo indicándole por lo menos:

  - El nombre del contenido
  - El tamaño
  - La descripción
  - El proveedor
  - El tipo del contenido

- Fase 5: Descarga del contenido

Cuando el usuario de su conformidad, el terminal descargará el contenido mediante la dirección especificada en el descriptor del contenido. Cualquier error que se produzca durante dicha comunicación se deberá notificar.

- Fase 6: Instalación

Esta fase depende del tipo de contenido descargado. A grandes rasgos se puede decir que consiste en la preparación del contenido para su ejecución o visualización. Antes de mostrarlo o ejecutarlo, el terminal deberá enviar la notificación de éxito si procede.

- Fase 7: Notificación de la descarga

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Mediante el parámetro de notificación del descriptor el terminal puede saber si es necesaria y en caso de serlo la dirección a la que hay que mandar el código con el estado y la descripción del mismo.

- Fase 8: Confirmación de la descarga y siguiente paso

Finalmente el terminal mostrará al usuario el resultado de la descarga. Además podrá permitir al usuario escoger una opción de continuar, que enlazará con la dirección especificada en el correspondiente parámetro del descriptor.

3.3.3 Descriptor

El descriptor es un archivo de texto en el que se incluye toda la información relativa al contenido y así como los datos necesarios para su descarga. Este punto es la principal diferencia entre la descarga de MIDLets y la descarga de contenidos tal y como define OMA.

En la descarga de MIDLets el descriptor es un archivo de texto en el que se listan las propiedades de forma sencilla, sin usar XML. Se denomina JAD (*Java Application Descriptor*).

Figura 36
Figura 37 MIDlet-Name: Show Properties MIDlet
Figura 38 MIDlet-Version: 1.0.1
Figura 39 MIDlet-Vendor: Core J2ME
Figura 40 MIDlet-Jar-URL: ShowProperties.jar
Figura 41 MIDlet-Jar-Size: 1190
Figura 42 MIDlet-1: ShowProps, , ShowProperties
Figura 43 MIDlet-Description: A simple property list example
Figura 44 JadFile-Version: 1.5
Figura 45 MIDlet-Data-Size: 500

En el caso del descriptor de *OMA Download* el archivo se basa en XML con las propiedades necesarias para la descarga. En este caso se le suele denominar DD (*Download Descriptor*) y se puede ver un ejemplo en el siguiente código:

Figura 46 <media xmlns="http://www.openmobilealliance.org/xmlns/dd">
Figura 47 <type>image/gif</type>
Figura 48 <objectURI>http:/download.example.com/image.gif</objectURI>
Figura 49 <size>100</size>
Figura 50 <installNotifyURI>http:/download.example.com/image.gif?id=image</installNotifyURI>
Figura 51 </media>

3.4 DRM

A partir de la experiencia de Internet y los PCs, en donde el pirateo de contenidos es la normal general, los operadores, proveedores y fabricantes de terminales han visto la necesidad de proveer mecanismos para la protección de los contenidos descargados.

Esta protección pasa por añadir en la descarga de los contenidos información en la que se especifiquen los permisos que ha comprado el usuario para el uso de ese contenido. Estos permisos indican si se

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

puede reproducir, guardar, copiar, las veces que se puede hacer o incluso el tiempo que se puede hacer. Evidentemente en el terminal, el sistema operativo tiene que proteger estos archivos a bajo nivel según esta información.

Debemos recordar en este momento que la descarga de contenidos se puede realizar bien mediante un mensaje MMS bien mediante algún tipo de conexión a Internet (WAP por ejemplo) utilizando el estándar *OMA Download*. Además, también soporta la transmisión de derechos para realizar *streamings* de contenidos multimedia.

3.4.1 Arquitectura

Como hemos visto la especificación DRM gira en torno al envío de un fichero que especifique los derechos que tiene el usuario para utilizar el contenido descargado. A este respecto hay tres posibles casos que corresponden con las tres arquitecturas definidas para DRM.

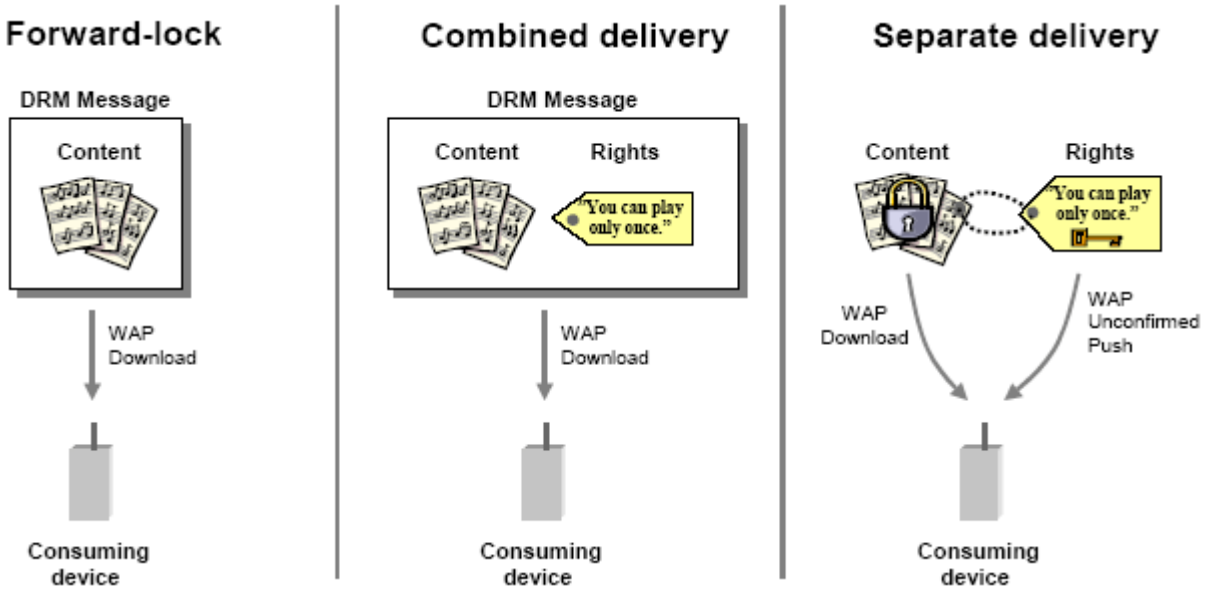


Figura 0.43: Diferentes escenarios de uso de la tecnología DRM Fuente: OMA

Las tres posibles arquitecturas se distinguen entre sí en la forma de mandar la información de los derechos. En la segunda se manda junto con el contenido, en la tercera se manda por separado y en la primera no envían derechos sino que se utilizarán unos por defecto.

- *Forward-lock*

  En este método el contenido se encapsula en un mensaje DRM y se envía al terminal sin incluir ningún derecho explícitamente. Como el nombre indica, el terminal podrá ejecutar, presentar, guardar el contenido al usuario pero no dejará reenviarlo a otro usuario ni sacarlo del terminal.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

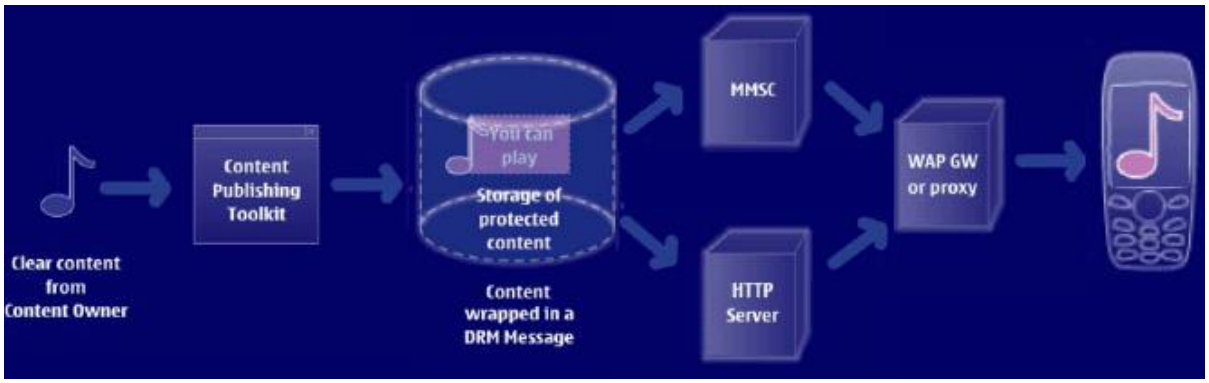


Figura 0.44: Caso de uso *Forward-Lock* Fuente: Nokia

- *Combined delivery*

En este método se especifican explícitamente los derechos de los que dispone el usuario para el uso del contenido. Estos derechos se expresan mediante un lenguaje específico y se envían en un mensaje DRM junto con el contenido.

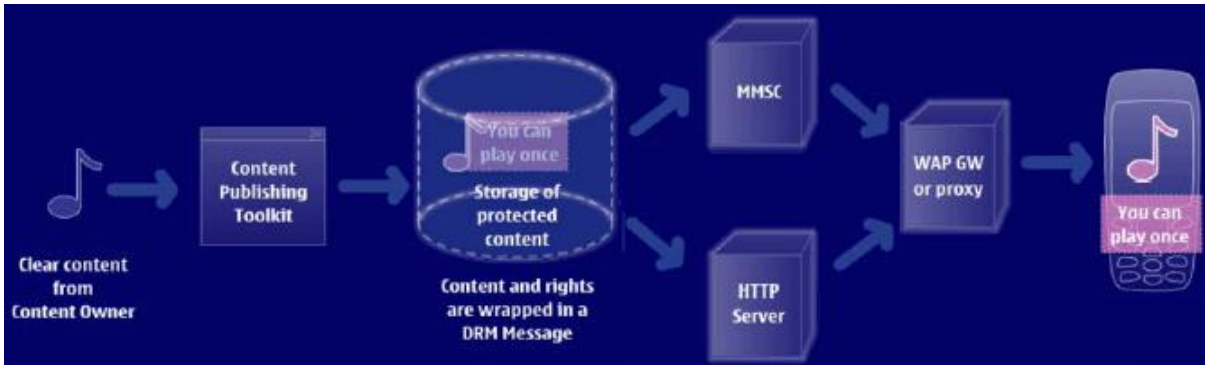


Figura 0.45: Caso de uso *Combined Delivery* Fuente: Nokia

- *Separate delivery*

Finalmente el tercer método consiste en enviar por separado los contenidos de los derechos. El contenido irá encapsulado en un mensaje DRM así como encriptados simétricamente. Posteriormente, se enviarán de algún modo (típicamente WAPPush) los derechos así como la clave necesaria para obtener los contenidos.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

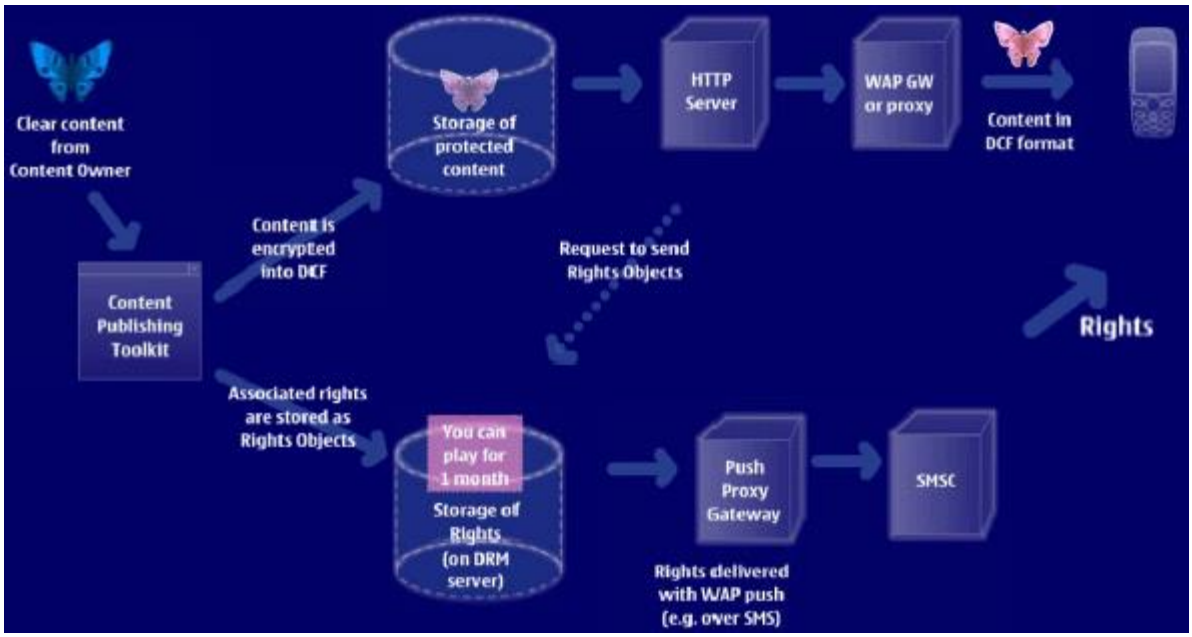


Figura 0.46: Caso de uso *Separate Delivery* Fuente: Nokia

3.4.2 Lenguaje de declaración de derechos

El lenguaje sobre el que se declaran los derechos se basa, como no podía ser de otra forma, en XML. Especifica lo que el terminal debe dejar hacer al usuario con el contenido descargado. Incluirá, por ejemplo, cuantas veces puede el usuario imprimir una imagen o durante cuanto tiempo puede el usuario escuchar una canción.

Figura 52 <o-ex:rights xmlns:o-ex="http://odrl.net/1.1/ODRL-EX" xmlns:o-dd="http://odrl.net/1.1/ODRL-DD">
Figura 53 <o-ex:context>
Figura 54 <o-dd:version>1.0</o-dd:version>
Figura 55 </o-ex:context>
Figura 56 <o-ex:agreement>
Figura 57 <o-ex:asset>
Figura 58 <o-ex:context>
Figura 59 <o-dd:uid>cid:4567829547@foo.com</o-dd:uid>
Figura 60 </o-ex:context>
Figura 61 </o-ex:asset>
Figura 62 <o-ex:permission>
Figura 63 <o-dd:play/>
Figura 64 </o-ex:permission>
Figura 65 </o-ex:agreement>
Figura 66 </o-ex:rights>

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Además en el caso de estar en la arquitectura en la que se mandan los derechos por separado, hay que mandar la clave con la que se podrán desencriptar los contenidos. Esta clave viene en el elemento del XML denominado *KeyValue*.

Figura 67 <o-ex:rights xmlns:o-ex="http://odrl.net/1.1/ODRL-EX" xmlns:o-dd="http://odrl.net/1.1/ODRL-DD">
Figura 68 <o-ex:context>
Figura 69 <o-dd:version>1.0</o-dd:version>
Figura 70 </o-ex:context>
Figura 71 <o-ex:agreement>
Figura 72 <o-ex:asset>
Figura 73 <o-ex:context>
Figura 74 <o-dd:uid>cid:4567829547@foo.com</o-dd:uid>
Figura 75 </o-ex:context>
Figura 76 <ds:KeyInfo>
Figura 77 <ds:KeyValue>vUEwR8LzEJoeiC+dgT1mgg==</ds:KeyValue>
Figura 78 </ds:KeyInfo>
Figura 79 </o-ex:asset>
Figura 80 <o-ex:permission>
Figura 81 <o-dd:play/>
Figura 82 </o-ex:permission>
Figura 83 </o-ex:agreement>
Figura 84 </o-ex:rights>

3.4.3 Formato de archivo para los contenidos

El formato de los archivos que se envían al terminal se basa siempre en envíos *multipart* de HTTP. En el caso del método *forward-lock* y en el de *combined delivery* el envío se realiza sin encriptar y por lo tanto no hace falta mayor explicación, lo único a tener en cuenta es que en el caso de enviar derechos estos deben ir antes que el contenido en la estructura del mensaje.

En el caso del método *separated delivery* el contenido se encapsula en un formato en el que se incluyen varias cabeceras y posteriormente los datos encriptados.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Field name	Type	Purpose
Version	Uint8	Version number
ContentTypeLen	Uint8	Length of the ContentType field
ContentURILen	Uint8	Length of the ContentURI field
ContentType	ContentTypeLen octets	The MIME media type of the plaintext data.
ContentURI	ContentURILen octets	The unique identifier of this content object.
HeadersLen	Uintvar	Length of the Headers field
DataLen	Uintvar	Length of the Data field
Headers	HeadersLen octets	Headers define additional meta data about this content object.
Data	DataLen octets	The encrypted data

4. TRANSMISIÓN DE CONTENIDOS MULTIMEDIA

Una de las grandes aplicaciones de la transmisión de datos en el mundo móvil hace referencia a los contenidos multimedia. La importancia de estos contenidos no hace sino aumentar con el tiempo según van aumentando las capacidades de la red. De hecho se vuelven servicios necesarios para poder utilizar todo el ancho de banda de las redes 3G. Por este tipo de contenidos son de máxima actualidad y la estrella de las nuevas redes móviles celulares.

A la hora de hablar de transmisión de contenidos multimedia, para ser estrictos realmente podemos de tres tipos de transmisiones:

- Transmisiones con envíos de mensajes multimedia con los contenidos.

  El nuevo estándar MMS permite mandar todo tipo de contenidos (incluso video) mediante un mensaje multimedia. Evidentemente es un servicio diferido y, puesto que se envía todo el contenido y luego se visualiza, el tamaño de los contenidos multimedia no puede ser muy grande. Además, los mensajes son gestionados y almacenados para su envío (arquitectura *Store&Forward*) en el centro de mensajes (MMSC) por lo que también limitará la capacidad de éste el tamaño máximo del mensaje. Normalmente, como estos centros tienen que gestionar cantidades ingentes de mensajes no suelen permitir un tamaño muy alto (200 Kb) para no saturarse.

- Transmisiones mediante descargas

  En el anterior apartado hemos estado hablando de las distintas formas de descarga de contenidos y cómo funcionan. En resumen, los usuarios mediante la navegación por distintas páginas residentes en servidores seleccionarán un contenido a descargar y lo recibirán mediante una transacción estándar. Al igual que en la mensajería, la visualización del contenido se realiza después de descargarlo entero por lo que tampoco el contenido puede ser muy grande. Aún así, no tiene la limitación de estar en medio un centro de mensajes por lo que aún siendo pequeño el tamaño máximo razonable que se puede descargar siempre podría ser mayor que el de un MMS.

- Transmisiones mediante *streaming*

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

A grandes rasgos, los servicios de transmisión de contenidos multimedia mediante *streaming* consisten en el envío de los contenidos multimedia como un flujo continuo de datos y la reproducción de los mismos en el terminal a la vez que se reciben y sin tener que almacenarlos en el mismo. De esta forma el tamaño de los contenidos transmitidos pueden ser realmente más grandes puesto que no hace falta recibirlos enteros para empezar la reproducción y los datos del contenido multimedia no tienen porque ser almacenados en el terminal. Este tipo de trasmisiones está claramente orientado a la emisión de contenidos de audio y video.

Las diferencias entre este sistema y las descargas son claras. Por poner un ejemplo, en la televisión los datos recibidos y reproducidos simultáneamente se pueden considerar *streaming*, mientras que la analogía con las descargas podrían ser el visionado de una película de vídeo que ya está almacenada antes de reproducirla.

Pero no todo son ventajas, puesto que se reproduce a la vez que se van recibiendo los datos necesarios para seguir con la reproducción, es necesario que el caudal que se recibe se mantenga constante o no sufra caídas bruscas, por lo que se puede decir que los requisitos son de tiempo real, si no llegan los siguientes datos la reproducción se parará con la consiguiente falta de calidad y mala experiencia para el cliente

.

A lo largo de este apartado vamos a ver los principales puntos a estudiar en la transmisión de contenidos multimedia, como los formatos aceptados por el grupo 3GPP para las redes móviles o los protocolos necesarios para realizar la transmisión de los contenidos. Finalmente, analizaremos la adaptación de contenidos, cuestión importante en un entorno en la que la variedad en las capacidades de los terminales es tal que se han definido estándares para establecer las características de los dispositivos.

Puesto que ya hemos visto anteriormente tanto la mensajería multimedia como las descargas, vamos a centrarnos en el análisis del *streaming* de contenidos multimedia en los posteriores apartados. Aún así, muchas cuestiones como la adaptación de contenidos o los formatos de los mismos son cuestiones generales que afectan tanto a los MMS, como las descargas, así como incluso a la creación de páginas WAP o WEB para terminales móviles.

Muchos de los puntos que vamos a ver están todavía en proceso de estandarización y evolución. Por ejemplo, ya se está pensando en proporcionar *streaming* con los mensajes multimedia y también se empieza a estandarizar la gestión de derechos de los contenidos transmitidos mediante *streaming* (DRM).

4.1. Formatos

Los formatos disponibles actualmente para los contenidos en Internet son realmente muy numerosos. Sólo hace falta una breve navegación por Internet para darnos cuenta la cantidad de formatos de sonido, imágenes y vídeos que existen. Debido a esta gran variedad y a las restricciones debidas a las capacidades limitadas de los terminales, el grupo 3GPP ha definido los formatos que se recomienda implementar en los terminales. De esa forma los proveedores de contenidos y aplicaciones pueden

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

saber qué formatos deben utilizar para codificar sus contenidos y que puedan ser utilizados por la mayoría de terminales.

En la siguiente tabla se ilustran los formatos y *codecs* recomendados por el 3GPP.

Tipo de contenido	**Formato o Codecs**
Voz	AMR
Audio	MPEG-4 AAC Low Complexity MPEG-4 AAC Long Term Prediction (opcional)
Audio sintetizado	Midi
Video	H.263 MPEG4 (opcional)
Imágenes	JPEG
Gráficos	PNG, GIF
Gráficos vectoriales	SVG
Texto	XHTML MP SMIL

4.2. Protocolos

Respecto a los protocolos para transmitir contenidos multimedia, primero debemos recordar que ya hemos estudiado la mayoría cuando analizamos el funcionamiento de MMS, WAP o las descargas. Aún así, nos queda por ver un importante conjunto que hace referencia al *streaming* de contenidos.

El servicio de *streaming* de contenidos multimedia ha sido especificado por el grupo 3GPP basándose en estándares de Internet como los protocolos RTP (*Real Time Transport Protocol*), RTCP (*Real Time Control Protocol*), RTSP (*Real Time Streaming Protocol*) y SDP (*Session Description Protocol*). En las siguientes páginas, vamos a analizar las principales funciones y características de cada uno de ellos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

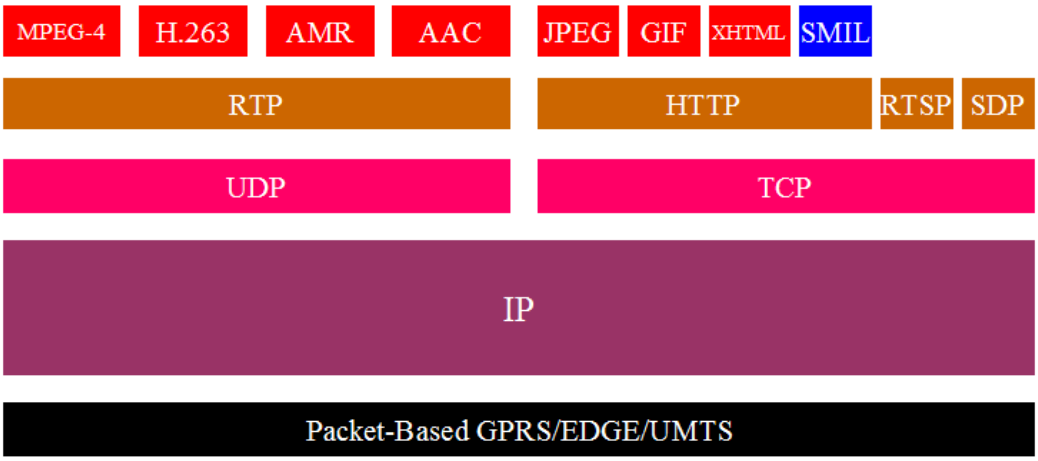


Figura 0.47: Pila de protocolos en la transmisión de contenidos multimedia Fuente Emblaze Research

4.2.1. RTP

El protocolo de transporte en tiempo real RTP es un protocolo IP que proporciona soporte para la transmisión de datos en tiempo real, como por ejemplo *streams* de contenidos multimedia (video y audio). Las funciones que RTP proporciona incluyen la reconstrucciones de temporizaciones, la detección de pérdida de paquetes y la seguridad e identificación de los contenidos transmitidos.

Aunque RTP ha sido diseñado para funcionar en modo *multicast* también puede hacerlo en modo *unicast*. Se puede usar igualmente para transporte unidireccional, como video bajo demanda, o para servicios interactivos como la telefonía por Internet.

RTP trabaja conjuntamente con el protocolo auxiliar de control que vamos a ver a continuación, y que sirve para obtener realimentación de la calidad de la transmisión así como para obtener información sobre los participantes de la sesión.

Puesto que se basa en IP, funciona sobre una red que transmite datagramas por lo que los paquetes enviados se reciben con un retardo no predecible y en algunos casos desordenados (si trabajamos encima de UDP), sin embargo las aplicaciones multimedia requieren temporizaciones adecuadas en la transmisión de datos y su reproducción. RTP proporciona marcas temporales, numeración de secuencias y otros mecanismos para proporcionar un soporte robusto a la temporización. Con estos mecanismos, RTP provee un transporte punto a punto en tiempo real sobre la red basada en datagramas o paquetes.

La temporización es la información más importante en las aplicaciones de tiempo real, no sólo porque haya que saber qué datos son los que hay que utilizar en cada momento, sino también para sincronizar los diferentes contenidos que pueden estar llegando simultáneamente (por ejemplo, audio y video).

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Quien envía asigna una marca temporal (*timestamp*), que corresponde al momento en el que el primer *byte* del paquete fue muestreado. El receptor utilizará esta marca temporal para reconstruir la temporización y reproducir los datos en el momento y a la velocidad correctos. Como hemos dicho, antes la temporización también es importante para sincronizar distintas secuencias de datos recibidas simultáneamente, aunque RTP no es responsable de esta sincronización, sino que es realiza por los protocolos superiores de aplicación.

Los dos protocolos de transporte más usados sobre IP son TCP y UDP. TCP proporciona un flujo fiable y orientado a conexión entre dos terminales, mientras que UDP provee un servicio de datagramas no fiable (pueden llegar datos repetidos, desordenados o incluso no llegar) y no orientado a conexión. Aunque RTP se puede montar encima de TCP, está pensado para funcionar sobre todo encima de UDP. Esta elección se debe principalmente a que RTP ha sido diseñado para datos en tiempo real y por lo tanto la fiabilidad del transporte no es tan importante como la recepción en el tiempo adecuado. Es más, TCP implementa la fiabilidad de la comunicación mediante retransmisiones de paquetes IP, lo cual en el caso de tráfico de datos en tiempo real es un inconveniente. En el caso de congestión de la red, los paquetes perdidos, simplemente, redundarán en una calidad inferior, pero aceptable, en cambio, si el protocolo insiste en la transmisión fiable, retransmitiendo paquetes perdidos, el retardo posiblemente aumentará, degradando más la calidad de la recepción.

Puesto que UDP no despacha los paquetes ordenados en el tiempo, se usa la numeración de secuencias para, en recepción, ordenar los paquetes recibidos. La numeración de secuencias se usa también para la detección de pérdidas de paquetes. En la práctica, RTP se implementa generalmente dentro de la aplicación, pues temas como la recuperación de paquetes perdidos o de control de las congestiones de red, tienen que ser implementados al nivel de aplicación.

Otra función de RTP es identificar el formato de los datos así como el codec con el que se han comprimido los mismos. Para realizar esta labor utiliza un identificador de carga *payload*, con el que la aplicación receptora sabe como interpretar y reproducir los datos. Además, el protocolo RTP permite la identificación de las fuentes de los datos. Permite a la aplicación receptora conocer de dónde vienen los datos, por ejemplo, en una audio conferencia, a partir del identificador de la fuente se puede saber quién está hablando.

Para establecer una sesión RTP, la aplicación define una pareja de direcciones destino de transporte (una dirección de red con un par de puertos para RTP y RTCP). En una sesión de transmisión de datos multimedia, cada tipo de datos es transportado por una sesión separada de RTP, con sus propios paquetes de RTCP, que informan de la calidad de recepción para esa sesión. A continuación vamos a ver más en profundidad el protocolo auxiliar de control para RTP, el RTCP.

4.2.2. RTCP

RTCP (*Real Time Control Protocol*) es un protocolo diseñado para trabajar conjuntamente con RTP. En una sesión RTP, los participantes se envían cada cierto tiempo paquetes RTCP para tener información sobre la calidad de la recepción. Se definen cinco tipos de paquetes RTCP para transportar la información de control, a saber:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- *RR (*informe de receptor). Son los informes enviados por los participantes que actúan como receptores. Estos informes contienen datos acerca de la calidad de la recepción, incluyendo el número más alto de secuencia recibido, el número de paquetes perdidos, la información sobre paquetes desordenados y las marcas temporales para calcular el retardo de ida y vuelta entre receptor y el transmisor.
- *SR* (informe de emisor). Los informes de emisor son generados por los emisores activos. Además de datos sobre la calidad de la recepción, como en los RR, contienen datos de sincronización, de contadores acumulativos de paquetes y del número de *bytes* enviados.
- *SDES* (datos de descripción de fuente*).* Contienen información descriptiva de la fuente de datos.
- *BYE*. Indica el fin de la participación.
- *APP* (datos específicos de la aplicación). Se han reservado para usos experimentales, mientras se desarrollan nuevas funciones y tipos de aplicaciones.

A través de estos paquetes, RTCP puede proporcionar las funciones para las que ha sido diseñado y que a continuación se detallan:

- Monitorización de la calidad de servicio (QoS) y congestión de red

La función primaria de RTCP es proporcionar realimentación a una aplicación sobre la calidad de la distribución de los contenidos. Esta información es útil a los emisores, a los receptores y a otras terceras partes interesadas (monitores de aplicación, por ejemplo). El emisor puede ajustar su transmisión basándose en estos informes. El receptor puede determinar si la congestión de red es local, regional o global.

- Identificación de las fuentes

Las fuentes de datos se identifican en los paquetes RTP con identificadores de 32 bits generados aleatoriamente. Estos identificadores no son apropiados para usuarios humanos. Los paquetes SDES contienen identificadores únicos globales e información textual, como el nombre de los participantes, el número de teléfono, la dirección de e-mail, etc.

- Sincronización intermedia

RTCP envía informes con información de tiempo real que corresponde con una determinada marca temporal RTP. Esa información puede ser utilizada para sincronizar fuentes de datos que procedan de distintas sesiones RTP.

- Escalado de la información de control

Los paquetes RTCP se envían periódicamente entre los participantes. Cuando el número de participantes aumenta, se hace necesario establecer un compromiso entre la obtención de información actualizada y la sobrecarga por tráfico de red. RTP limita el tráfico de control al 5 por ciento del tráfico total de la sesión, esto se consigue ajustando el tráfico RTCP a un régimen acorde al número de participantes.

4.2.3. RTSP

El concepto de *streaming* se basa en que en vez de almacenar grandes ficheros multimedia y reproducirlos, los datos multimedia se envían a través de la red secuencialmente en *streams*. El proceso de

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

streaming rompe los datos en paquetes del tamaño apropiado para su transmisión entre servidor y cliente. Un cliente puede reproducir el primer paquete, mientras decodifica el segundo y recibe el tercero. Así, el usuario puede disfrutar de los contenidos sin esperar a que finalice la transmisión.

RTSP (*Real Time Streaming Protocol*) es el protocolo de *streaming* en tiempo real, un protocolo multimedia cliente-servidor de presentación para permitir el despacho controlado de los datos multimedia a través de la red IP. Proporciona una interfaz, al estilo de un reproductor de vídeo, con funciones de control remoto como "pausa", "avance rápido", "atrás", e "ir a posición".

RTSP es un protocolo a nivel de aplicación, diseñado para trabajar conjuntamente con protocolos de bajo nivel como RTP. Proporciona mecanismos para seleccionar canales de envío (como UDP, UDP *multicast*, y TCP). Se puede ver RTSP como un control remoto en red entre el servidor y el cliente.

RTSP pretende proporcionar, para presentaciones multimedia, los mismos servicios que HTTP proporciona para textos y gráficos, de hecho se ha diseñado intencionadamente con una sintaxis similar, de tal modo que la mayor parte de los mecanismos de extensión de HTTP se pueden añadir a RTSP.

En RTSP cada presentación y cada *stream* multimedia es identificado por una URL RTSP. La presentación completa y las propiedades de los medios se definen en un fichero de descripción de presentación, entre la información se incluye el tipo de codificación, el idioma, las URLs RTSP, las direcciones de destino, los puertos y otros parámetros. El cliente puede obtener este fichero mediante HTTP, *e-mail* u otros métodos.

Aún así, RTSP difiere de HTTP en muchos aspectos. En primer lugar, HTTP es un protocolo sin estados y RTSP ha de mantener los estados de la sesión para enlazar las peticiones con los *streams* relacionados. En segundo lugar, HTTP es asimétrico, cuando el cliente realiza una petición el servidor responde, mientras que en RTSP ambos, cliente y servidor, pueden realizar peticiones.

Los servicios y operaciones soportados son:

- *Options*. El cliente o el servidor comunican a la otra parte las opciones que pueden aceptar.
- *Describe*. El cliente recupera la descripción de una presentación o medio identificado por una URL.
- *Announce*. Cuando se envía desde el cliente al servidor, *Announce* envía la descripción de una presentación identificada por su URL. Cuando se envía desde el servidor al cliente, *Announce* actualiza la descripción de sesión en tiempo real.
- *Setup*. El cliente le pide al servidor que reserve recursos para un *stream* y para iniciar una sesión RTP.
- *Play*. El cliente le pide al servidor que comience a enviar datos para el *stream* reservado, vía *Setup*.
- *Pause*. El cliente, temporalmente, para el flujo del *stream* sin liberar los recursos asociados.
- *Teardown*. El cliente le pide al servidor que detenga el envío de un determinado *stream* y libere los recursos asociados.
- *Get_Parameter*. Recupera el valor de un parámetro de una presentación o un *stream* identificado por su URI.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- *Set_Parameter*. Asigna el valor de un parámetro de una presentación o *stream* identificado por su URL.
- *Redirect*. El servidor informa al cliente de que se debe conectar a otra dirección de servidor. La cabecera obligatoria de localización indica el URL que el cliente debe contactar.
- *Record*. El cliente inicia la grabación de unos datos multimedia de acuerdo a la descripción de la presentación.

Algunos de estos métodos pueden ser enviados tanto por el cliente como por el servidor, pero otros sólo pueden ser emitidos en una dirección. No todos los métodos son necesarios para tener un servidor plenamente funcional. Por ejemplo, un servidor de medios, alimentado con datos en vivo, puede no soportar el método *Pause*.

Las solicitudes RTSP se envían, como normal general, por un canal independiente del canal de datos. Pueden ser transportadas por conexiones persistentes o creando una conexión por petición-respuesta.

4.2.4. SDP

El protocolo SDP (*Session Description Protocol*) es una sintaxis de descripción de la sesión basada en texto, que informa de la dirección del servidor, de los contenidos de los *streams*, y *codecs* necesarios y de los contenidos solicitados durante la sesión por el cliente.

Seguramente, la mejor comparación que podamos hacer sobre los archivos SDP sean los descriptores de descargas que veíamos en el capítulo anterior. Al fin y al cabo el objetivo es el mismo, mandar información al cliente receptor sobre el contenido multimedia que se va a transmitir y cómo se va a hacer tal y como podemos ver en el siguiente ejemplo:

Figura 85 v=0
o=ghost 2890844526 2890842807 IN IP4 192.168.10.10
s=3GPP Unicast SDP Example
i=Example of Unicast SDP file
u=http://www.infoserver.com/ae600
e=ghost@mailserver.com
c=IN IP4 0.0.0.0
t=0 0
Figura 86 a=range:npt=0-45.678
m=video 1024 RTP/AVP 96
Figura 87 b=AS:128
a=rtpmap:96 H263-2000/90000
a=fmtp:96 profile=3;level=10
a=control:rtsp://mediaserver.com/movie.3gp/trackID=1
a=framesize:96 176-144
a=recvonly

4.3. Adaptación de contenidos

En la transmisión de contenidos multimedia en el entorno móvil hay un apartado que nunca debemos olvidar. Los terminales móviles tienen grandes diferencias entre sí y además existen gran cantidad de modelos diferentes. Esto provoca que los contenidos deban ser adaptados en la medida de lo posible

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

al terminal receptor. Unas veces habrá que recodificar los contenidos cambiándoles el formato a uno que el dispositivo entienda, otras veces habrá que cambiar propiedades del contenido como altura y anchura, e incluso otras no habrá que listar el contenido al usuario porque no lo va a soportar su dispositivo.

Una de las decisiones más importantes cuando hablamos de adaptar contenidos es el coste computacional de realizar la codificación de contenidos multimedia en tiempo real. Si creemos que puede ser demasiado alto para nuestras expectativas de tráfico y nuestras capacidades hardware deberemos plantearnos soluciones basadas en agrupar los terminales en familias y realizar la adaptación en tiempo de *deployment* una sola vez.

El mayor problema que tienen las aplicaciones multimedia que no incluyen la adaptación de contenidos al terminal es que tienden a codificar los contenidos para el caso peor, es decir para que funcione por lo menos en el terminal con peores características. Esto lleva a que en terminales avanzados se vean los contenidos de forma muy pobre. Un ejemplo característico de este problema son las imágenes WAP pequeñas y en blanco y negro en terminales con pantallas grandes y en color. La sensación que dan estas páginas es muy perjudicial y es una de las causas del fracaso de WAP.

Además, otro motivo para adaptar los contenidos al terminal es optimizar el ancho de banda usado. Por ejemplo, no tiene sentido mandar una imagen a un terminal mayor que su pantalla puesto que el terminal o bien la descartará o bien la reducirá. El primer caso es nefasto, pero el segundo tampoco es óptimo porque hemos gastado dinero y ancho de banda innecesarios.

La industria enseguida ha sido consciente de esta problemática y ha optado por darle soluciones por dos vías. Por un lado ha optado por estandarizar los formatos de los contenidos multimedia de tal forma que casi todos los terminales nuevos soportan los mismos formatos. Además se han diseñado mecanismos para que el terminal pueda comprobar si soporta el contenido multimedia antes de descargarlo (aunque lo óptimo sería no mostrar los contenidos que el usuario no pudiese utilizar).

La segunda solución pensada es la comunicación de todas las características relevantes del terminal a la aplicación para que pueda particularizar y adaptar los contenidos a éstas. Para conseguir este objetivo se ha diseñado la tecnología UAProf (*User Agent Profile*) que permite obtener un fichero XML con toda la información necesaria sobre el terminal.

En los posteriores apartados vamos a analizar primero el estándar UAProf, para luego entrar en las distintas posibilidades de adaptación que se pueden realizar para los tipos más importantes de contenidos multimedia, las imágenes, los sonidos y los vídeos.

4.3.1. UAProf

El estándar UAProf (*User Agent Profile*) define las transacciones que se deben realizar entre el terminal, los proxies intermedios y el servidor final para poder comunicar con éxito las capacidades del terminal. Esta tecnología, también conocida como CPI (*Capability and Preference Information*) usa el modelo emergente en Internet llamado CC/PP (*Composite Capability/Preference Profile*).

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

La definición de la información enviada referente a las características del terminal engloba las siguientes áreas entre otras menos importantes:

- Características del hardware del terminal (pantalla, teclas, fabricante)
- Características del software del terminal (sistema operativo, codecs de audio y video, soporte para aplicaciones nativas)
- Preferencias del usuario y de las aplicaciones (versión y modelo del navegador, lenguajes de marcado soportados, etc.)
- Características WAP (versión de WAP, librerías WMLScript, tamaño máximo de WML, etc.)

Todas estas características son incluidas en un archivo XML-RDF que puede ser analizado para personalizar las aplicaciones móviles. En la siguiente figura se puede ver un ejemplo de un fichero de un terminal real.

```
<?xml version=”1.0”?>
<!DOCTYPE rdf:RDF [
<!ENTITY ns-rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY ns-prf 'http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-
YYYYMMDD#'>
<!ENTITY prf-dt 'http://www.openmobilealliance.org/tech/profiles/UAPROF/xmlschema-
YYYYMMDD#'>
]>
<rdf:RDF xmlns:rdf=”&ns-rdf;”
xmlns:prf=”&ns-prf;”>
<rdf:Description rdf:ID=”MyDeviceProfile”>
  <prf:component>
    <rdf:Description rdf:ID=”HardwarePlatform”>
      <rdf:type rdf:resource=”&ns-prf;HardwarePlatform”/>
      <prf:ScreenSizeChar rdf:datatype=”&prf-dt;Dimension”>15x6</prf:ScreenSizeChar>
      <prf:BitsPerPixel rdf:datatype=”&prf-dt;Number”>2</prf:BitsPerPixel>
      <prf:ColorCapable rdf:datatype=”&prf-dt;Boolean”>No</prf:ColorCapable>
      <prf:TextInputCapable rdf:datatype=”&prf-dt;Boolean”>Yes</prf:TextInputCapable>
      <prf:ImageCapable rdf:datatype=”&prf-dt;Boolean”>Yes</prf:ImageCapable>
      <prf:Keyboard rdf:datatype=”&prf-dt;Literal”>PhoneKeypad</prf:Keyboard>
      <prf:NumberOfSoftKeys rdf:datatype=”&prf-dt;Number”>0</prf:NumberOfSoftKeys>
    </rdf:Description>
  </prf:component>
  <prf:component>
    <rdf:Description rdf:ID=”SoftwarePlatform”>
      <rdf:type rdf:resource=”&ns-prf;SoftwarePlatform”/>
      <prf:AcceptDownloadableSoftware rdf:datatype=”&prfdt;Boolean”>No</prf:Ac-
ceptDownloadableSoftware>
      <prf:CcppAccept-Charset>
        <rdf:Bag>
          <rdf:li rdf:datatype=”&prf-dt;Literal”>US-ASCII</rdf:li>
          <rdf:li rdf:datatype=”&prf-dt;Literal”>ISO-8859-1</rdf:li>
          <rdf:li rdf:datatype=”&prf-dt;Literal”>UTF-8</rdf:li>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Figura 120 <rdf:li rdf:datatype=”&prf-dt;Literal”>ISO-10646-UCS-2</rdf:li>
Figura 121 </rdf:Bag>
Figura 122 </prf:CcppAccept-Charset>
Figura 123 </rdf:Description>
Figura 124 </prf:component>
Figura 125 </rdf:Description>
Figura 126 </rdf:RDF>

De forma resumida, se puede decir que en las peticiones que realizan los clientes, se envía la cabecera *x-wap-profile* con la URL del fichero XML con las características. En realidad la situación es más compleja y a las peticiones que envían los terminales los elementos de red intermedios pueden añadir modificaciones.

4.3.2. Imágenes

Una imagen almacenada en un ordenador puede ser abstraída por una matriz bidimensional de puntos. Cada punto es denominado píxel y son muestras discretas de una imagen real en un espacio continuo. Estos píxeles tienen almacenado el color que se debe ver en este punto de la imagen.

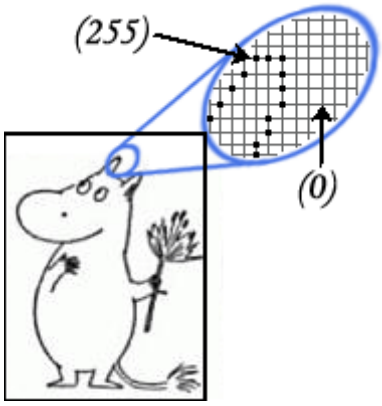


Figura 0.48: Imagen como matriz de píxel

Para almacenar la información de la imagen en tipos de datos conocidos por el ordenador, la imagen se estructura en bandas. Cada banda es una matriz bidimensional que almacena la información de color en una de las dimensiones de los espacios de colores. Por ejemplo, uno de los formatos de color más utilizados es el RGB (Red Green Blue), una imagen almacenada de esta forma poseerá tres bandas. Cada banda contendrá información del valor de cada píxeles en cada color. Si se desea tener una imagen en grises, solamente hará falta una banda para almacenar la información.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

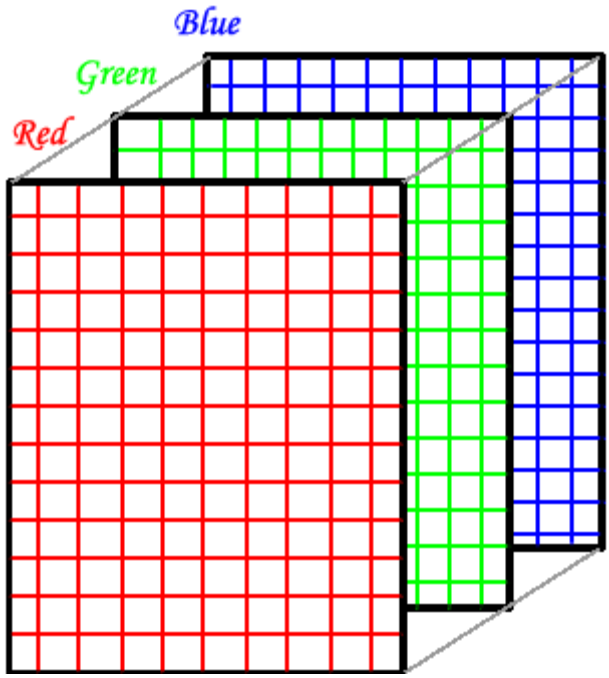


Figura 0.49: Imagen RGB

Para almacenar el valor de cada píxel en cada banda necesitaremos un tipo primitivo como entero, byte o entero largo. No son necesarios los decimales y según la precisión que queramos utilizaremos uno u otro tipo. Esta decisión tendrá como consecuencia una mayor gama de colores disponibles para presentar la imagen. Tal y como hemos descrito una imagen, podría ser almacenada como una matriz tridimensional de enteros o algún tipo de coma fija.

Una vez que hemos visto como se almacena una imagen podemos realizar unos sencillos cálculos que nos demostrarán el gran espacio que ocupan si no se comprimen de alguna forma. Si tenemos una imagen a color de 1000x1000 píxeles (tamaño normal) y se almacena con una profundidad de color de 24 bits (4 bandas de 1 byte), obtenemos:
1000x1000x4= 4 Mbytes.

Un tamaño que es claramente excesivo. Para guardar las imágenes en archivos almacenables en discos o para trasmitirlas por red de forma eficiente se utilizan formatos gráficos especiales. Estos formatos se aprovechan de características estadísticas o del ojo humano para reducir el tamaño que ocupa la imagen en el archivo. Entre estos archivos que comprimen la imagen podemos encontrar JPEG, GIF o PNG.

En las aplicaciones de Internet ya estamos acostumbrados a ver sin problemas todo tipo de imágenes. Pero en el mundo móvil las cosas no son tan fáciles. Como veremos hace falta adaptar la imagen para el terminal que la quiere visualizar o descargar. Las características que se pueden y deben adaptar de una imagen son las siguientes:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- El formato

No todos los terminales aceptan todos los formatos. Por esta razón hay que cambiar el formato de la imagen si el terminal no lo soporta. Los 4909 más normales que soportan los terminales son WBMP, JPEG, GIF y PNG.

- Las dimensiones

Muchos terminales no muestran las imágenes que reciben si éstas son más grandes que la pantalla. Otros reducen la imagen para adaptarla. Ambas situaciones no son deseables, la primera por que la imagen no se ve. La segunda porque supone un desperdicio de ancho de banda. Aún así, a veces el terminal permite realizar *scroll* vertical u horizontal, en estos casos es discutible la reducción de la dimensión correspondiente.

Otra razón para reducir las dimensiones de la imagen es que el tamaño máximo de fichero que soporta el terminal sea menor que el tamaño de la imagen. Muchos terminales tienen limitaciones respecto al número de bytes que se pueden descargar en un mismo fichero.

- Color/Grises

Evidentemente, si una imagen es color y el terminal tiene una pantalla en blanco y negro o con escala de grises habrá que convertir la imagen original.

- Profundidad de color

Finalmente también puede darse el caso de que la imagen posea más colores/grises de los que soporta el terminal. En ese caso hay que reducir los colores/grises. Para realizar esta labor la técnica más avanzada (sobre todo con fotografías) es el *Dithering*. Mediante la utilización de transformaciones basadas en matrices se puede reducir el número de colores con bastante calidad.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

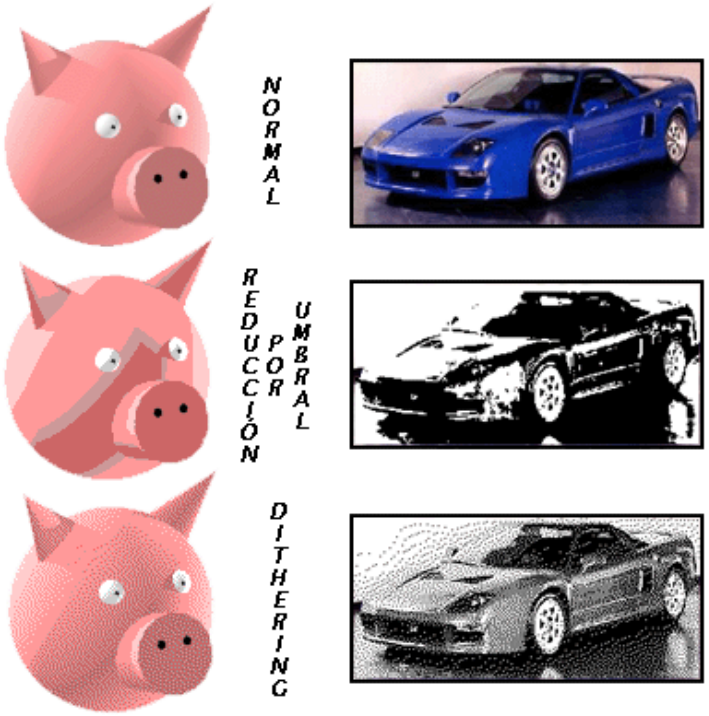


Figura 0.50: Ejemplo del uso de la técnica *Dithering*

En el caso de pasar imágenes a blanco y negro (pantallas muy comunes en los terminales antiguos), obtendremos mejores resultados si además combinamos el *Dithering* con algoritmos de ecualización del histograma y mejora del contraste.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

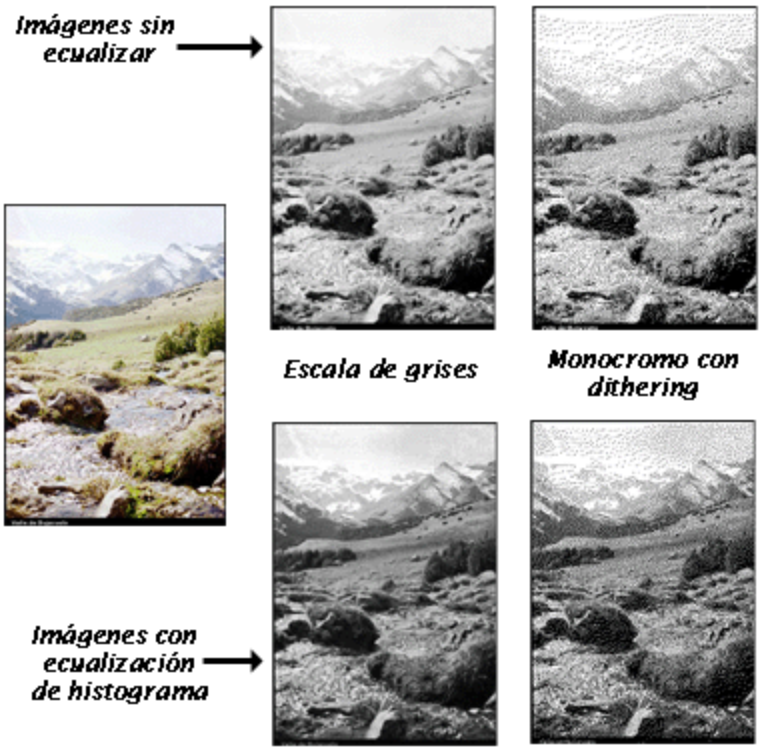


Figura 0.51: Diferentes algoritmos para obtener imágenes monocromo

Otro motivo para reducir la profundidad de color de una imagen es reducir el tamaño del fichero a transmitir con objeto de optimizarlo o reducirlo a menos del máximo del terminal.

4.3.3. Sonidos

Los sonidos son un contenido multimedia totalmente diferente a las imágenes. Para empezar hay dos grandes formas de representar sonidos en el mundo digital. La primera forma se suele denominar como sonido muestreado o "sampleado" (*sampled*). Las muestras (*samples*) son sucesivas fotos instantáneas de la señal. La onda sonora es convertida por los micrófonos en una señal eléctrica analógica, de la cual se extraen muestras a una determinada velocidad y se convierte a digital.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

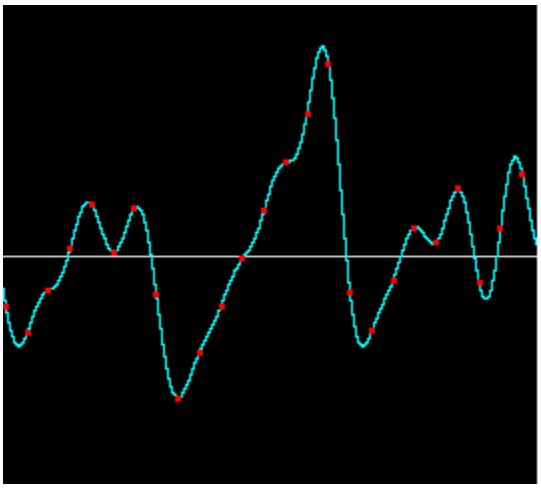


Figura 0.52: Sonido muestreado

Este gráfico representa la amplitud de onda analógica sonora en el eje vertical y el tiempo en el eje horizontal. La amplitud de la onda es medida periódicamente según cierta frecuencia, y convertida a digital mediante el proceso de cuantificación (se redondean los valores continuos a un cierto número de valores discretos). La precisión y calidad del sonido digital dependerán de la resolución temporal (o frecuencia de muestreo, velocidad con la que se obtienen las muestras) y de la resolución en amplitud (o cuantificación, el número de bits que sirven para digitalizar cada muestra. Como referencia, la música almacenada en los CDs de música se muestrea a 44.100 muestras por segundo y se cuantifica a 16 bits por muestra. Si echamos las cuentas veremos que si no comprimimos de alguna forma los sonidos, las muestras digitalizadas ocuparán un gran espacio en el disco o en el canal de transmisión.

La segunda forma de almacenar un sonido es guardar la forma en la que se debe reproducir. Básicamente se almacenan eventos que deben ser reproducidos por un sintetizador. Es como guardar las partituras de los instrumentos que van a tocar la melodía.

Por poner una analogía comparando ambas formas de almacenamiento y codificación, los sonidos muestreados podrían verse como una imagen escaneada, mientras que un sonido que se va a sintetizar podría verse como una imagen vectorial que ha de ser interpretada para poder visualizarla.

Los sonidos son más difíciles de adaptar que las imágenes. Al final lo único que es viable adaptar son los formatos de los sonidos. Si el terminal no soporta la codificación usada hay que cambiarla. Los formatos de sonidos más comunes en los terminales son AMR, WAV (sonido muestreado sin compresión), MIDI (sonido que se ha de sintetizar), MP3, AAC, i-Melody, etc.. Para añadir dificultad, no siempre son posibles todas las transformaciones. Por ejemplo de WAV (sonido muestreado) a MIDI (sonido a sintetizar) es realmente difícil realizar una transformación de calidad.

4.3.4. Videos

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Los vídeos son el contenido multimedia por excelencia. Excluyendo las adaptaciones de los sonidos, en las que se pueden hacer las mismas consideraciones que en el anterior apartado, nos queda un vector de imágenes por adaptar.

Una primera característica a adaptar en los vídeos son los formatos. En este caso la cosa se complica pues recodificar un vídeo tiene una carga computacional muy alta. Pero por suerte no suele ser habitual pues apenas hay codificadores que soporten los terminales (H263 y MPEG4).

Otras características que también se pueden tocar son las dimensiones y el número de imágenes (*frames*) del video. El objetivo sería adaptar el vídeo al tamaño del terminal y al tamaño máximo de fichero del mismo. Aún así también implicaría recodificación del contenido por lo que sólo tiene sentido en caso de vídeos muy cortos.

5. Páginas web de consulta

- www.gsmworld.com
- http://www.onforum.com/tutorials/gsm/
- http://www.wmlclub.com/
- http://www.cellular.co.za/
- http://www.auladatos.movistar.com/Aula-de-Datos/Tutoriales-y-Documentacion/
- http://www.mobilepositioning.com/
- http://telecom.iespana.es/telecom/telef/gsm-info.htm#fr
- http://www.3g-generation.com/
- http://www.moconews.net/
- www.thefeature.com
- http://www.cellular-news.com/
- http://www.openmobilealliance.org/
- http://www.forum.nokia.com/main.html
- http://www.openwave.com/
- http://www.motocoder.com
- http://www.umts-forum.org
- http://www.3gpp.org/
- http://www.itu.int
- http://www.etsi.org/
- http://java.sun.com/j2me/
- http://developers.sun.com/techtopics/mobility/
- http://www.nttdocomo.com/

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Alejandro Jiménez-Rodríguez, Luis Fernando Castillo, Manuel González (2012). Studying the mechanisms of the Somatic Marker Hypothesis in Spiking Neural Networks (SNN). ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
2. André Santos, Regina Nogueira, Anália Lourenço (2012). Applying a text mining framework to the extraction of numerical parameters from scientific literature in the biotechnology domain. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1
3. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. Information Sciences, 180(10), 2029–2043. https://doi.org/10.1016/j.ins.2009.12.032
4. Carlos Carvalhal, Sérgio Deusdado, Leonel Deusdado (2013). Crawling PubMed with web agents for literature search and alerting services. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 1
5. Carolina González, Juan Carlos Burguillo, Martín Llamas, Rosalía Laza (2013). Designing Intelligent Tutoring Systems: A Personalization Strategy using Case-Based Reasoning and Multi-Agent Systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 1
6. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
7. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
8. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
9. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
10. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
11. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
12. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
13. Chamoso, P., Raveane, W., Parra, V., & González, A. (2014). Uavs Applied to the Counting and Monitoring Of Animals. In Advances in Intelligent Systems and Computing (Vol. 291, pp. 71–80). https://doi.org/10.1007/978-3-319-07596-9_8
14. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
15. Corchado, J. A., Aiken, J., Corchado, E. S., Lefevre, N., & Smyth, T. (2004). Quantifying the Ocean's CO2 budget with a CoHeL-IBR system. In Advances in Case-Based Reasoning, Proceedings (Vol. 3155, pp. 533–546).
16. Corchado, J. M., & Aiken, J. (2002). Hybrid artificial intelligence methods in oceanographic forecast models. Ieee Transactions on Systems Man and Cybernetics Part C-Applications and Reviews, 32(4), 307–313. https://doi.org/10.1109/tsmcc.2002.806072
17. Corchado, J. M., Borrajo, M. L., Pellicer, M. A., & Yáñez, J. C. (2004). Neuro-symbolic System for Business Internal Control. In Industrial Conference on Data Mining (pp. 1–10). https://doi.org/10.1007/978-3-540-30185-1_1
18. Corchado, J. M., Corchado, E. S., Aiken, J., Fyfe, C., Fernandez, F., & Gonzalez, M. (2003). Maximum likelihood hebbian learning based retrieval method for CBR systems. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 2689, pp. 107–121). https://doi.org/10.1007/3-540-45006-8_11
19. Corchado, J. M., Pavón, J., Corchado, E. S., & Castillo, L. F. (2004). Development of CBR-BDI agents: A tourist guide application. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 3155, pp. 547–559). https://doi.org/10.1007/978-3-540-28631-8

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

20. Emmanuel Adam, Emmanuelle Grislin-Le Strugeon, René Mandiau (2012). MAS architecture and knowledge model for vehicles data communication. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1
21. Fdez-Riverola, F., & Corchado, J. M. (2003). CBR based system for forecasting red tides. Knowledge-Based Systems, 16(5–6 SPEC.), 321–328. https://doi.org/10.1016/S0950-7051(03)00034-0
22. Fernández-Riverola, F., Díaz, F., & Corchado, J. M. (2007). Reducing the memory size of a Fuzzy case-based reasoning system applying rough set techniques. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 37(1), 138–146. https://doi.org/10.1109/TSMCC.2006.876058
23. Glez-Bedia, M., Corchado, J. M., Corchado, E. S., & Fyfe, C. (2002). Analytical model for constructing deliberative agents. International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications, 10(3).
24. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. BMC Bioinformatics, 10. https://doi.org/10.1186/1471-2105-10-187
25. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
26. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
27. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
28. Joana Urbano, Henrique Lopes Cardoso, Ana Paula Rocha, Eugénio Oliveira (2012). Trust and Normative Control in Multi-Agent Systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1
29. Laza, R., Pavn, R., & Corchado, J. M. (2004). A reasoning model for CBR_BDI agents using an adaptable fuzzy inference system. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 3040, pp. 96–106). Springer, Berlin, Heidelberg.
30. Manuel Rodrigues, Sérgio Gonçalves, Florentino Fdez-Riverola (2012). E-learning Platforms and E-learning Students: Building the Bridge to Success. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
31. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. Expert Systems with Applications, 36(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
32. Méndez, J. R., Fdez-Riverola, F., Díaz, F., Iglesias, E. L., & Corchado, J. M. (2006). A comparative performance study of feature selection methods for the anti-spam filtering domain. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4065 LNAI, 106–120. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-33746435792&partnerID=40&md5=25345ac884f61c182680241828d448c5
33. Casado-Vara, R., & Corchado, J. (2019). Distributed e-health wide-world accounting ledger via blockchain. Journal of Intelligent & Fuzzy Systems, 36(3), 2381-2386.
34. Miki Ueno, Naoki Mori, Keinosuke Matsumoto (2012). Picture information shared conversation agent: Pictgent. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1
35. Nicholas Beliz, José Carlos Rangel, Chi Shun Hong (2012). Detecting DoS Attack in Web Services by Using an Adaptive Multiagent Solution. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
36. Nuno Trindade, Luis Antunes (2013). An Architecture for Agent's Risk Perception. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
37. Pawel Pawlewski, Paulina Golinska, Paul-Eric Dossou (2012). Application potential of Agent Based Simulation and Discrete Event Simulation in Enterprise integration modelling concepts. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1
38. Pérez, A., Chamoso, P., Parra, V., & Sánchez, A. J. (2014). Ground Vehicle Detection Through Aerial Images Taken by a UAV. In Information Fusion (FUSION), 2014 17th International Conference on.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

39. Prieto, J., Alonso, A. A., de la Rosa, R., & Carrera, A. (2014). Adaptive Framework for Uncertainty Analysis in Electromagnetic Field Measurements. Radiation Protection Dosimetry, ncu260.
40. Prieto, J., Mazuelas, S., Bahillo, A., Fernandez, P., Lorenzo, R. M., & Abril, E. J. (2012). Adaptive data fusion for wireless localization in harsh environments. IEEE Transactions on Signal Processing, 60(4), 1585–1596.
41. Prieto, J., Mazuelas, S., Bahillo, A., Fernández, P., Lorenzo, R. M., & Abril, E. J. (2013). Accurate and Robust Localization in Harsh Environments Based on V2I Communication. In Vehicular Technologies - Deployment and Applications. INTECH Open Access Publisher.
42. Rodolfo Salazar, José Carlos Rangel, Cristian Pinzón, Abel Rodríguez (2013). Irrigation System through Intelligent Agents Implemented with Arduino Technology. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
43. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. International Journal of Ambient Computing and Intelligence, v 1, n 1(1), 15–26. https://doi.org/10.4018/jaci.2009010102
44. Vasileios Efthymiou, Maria Koutraki, Grigoris Antoniou (2012). Real-Time Activity Recognition and Assistance in Smart Classrooms. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Introducción A Los Sistemas Móviles De Comunicaciones

Javier Pérez Trigo[1]

[1] RealInnova, Spain
jp_trigo@rinnova.es

Resumen: El presente capítulo se estructura en tres grandes partes. En la primera se analizan las diferentes tecnologías de comunicaciones móviles. Son el equivalente en la telefonía móvil a RDSI o ADSL. La importancia de conocer estos sistemas es doble, primero porque son muy habituales las comparaciones entre las mismas y es imprescindible saber de cada una las ventajas e inconvenientes. La segunda razón hace referencia a la importancia de las capacidades y limitaciones de estas tecnologías. La segunda parte muestra un nivel superio de abstracción y se analizan todas las tecnologías y servicios que están disponibles sobre estas tecnologías móviles y que nos van a permitir con su uso y combinación crear aplicaciones móviles. Entre estas tecnologías podemos encontrar una gran variedad para mensajería móvil, todo tipo de técnicas para localizar geográficamente usuarios, así como tecnologías cliente-servidor que emulan el modelo de la World Wide Web y tecnologías que se ejecutan directamente en el terminal. La tercera parte muestra las tecnologías que se montan encima, se analizarán las aplicaciones finales que existen y van a aparecer en el entorno móvil. Evidentemente las posibilidades aquí son muy grandes, por lo que trataremos de ver varios ejemplos significati-vos y haremos hincapié en las ventajas que las tecnologías móviles aportan para el desarrollo de aplicaciones en contraposición con las aplicaciones “fijas”.

Palabras clave: Tecnología móvil, comunicaciones

Abstract. This chapter is structured in three main parts. The first part analyses the different mobile communications technologies. They are the mobile telephony equivalent of ISDN or ADSL. The importance of knowing these systems is twofold, first because comparisons between them are very common and it is essential to know the advantages and disadvantages of each one. The second reason refers to the importance of the capacities and limitations of these technologies. The second part shows a superior level of abstraction and analyzes all the technologies and services that are available on these mobile technologies and that will allow us with their use and combination to create mobile applications. Among these technologies we can find a great variety for mobile messaging, all kinds of techniques to geographically locate users as well as client-server technologies that emulate the World Wide Web model and technologies that run directly on the terminal. The third part shows the technologies that are mounted on top, will analyze the final applications that exist and will appear in the mobile environment. Obviously the possibilities here are very great, so we will try to see several significant examples and we will place special emphasis on the advantages that mobile technologies bring to the development of applications as opposed to "fixed" applications.

Keywords: Mobile technology, communications

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

1 INTRODUCCIÓN

Durante la pasada década y el comienzo de la actual se ha producido un crecimiento explosivo de la informática y las telecomunicaciones. **Internet** es un hecho ya consumado que ha sido eje central de lo que está siendo una revolución en la gestión de la información. El acceso a esta información está generalizándose y el número de ciudadanos que se conecta diariamente en busca de todo tipo de servicios y aplicaciones es mayor cada día sin que haya dudas del éxito de este modelo.

A su vez, las **comunicaciones móviles** es otro de los sectores que ha experimentado más desarrollo en los últimos años llegando a todos los segmentos de la población. Además del aumento exponencial de usuarios que ha sufrido la telefonía móvil, los diferentes avances de la tecnología en redes y terminales tienen como consecuencia que este sector sea uno de los más dinámicos y cambiantes.

Si bien Internet no ha cumplido todas las expectativas de ingresos económicos que se le suponían, la telefonía móvil sí que ha conseguido grandes beneficios e ingresos. Esta alta rentabilidad y volumen ha supuesto grandes inyecciones de dinero en todo este sector. Tanto los fabricantes de terminales, los proveedores de aplicaciones, como los operadores han conseguido brillantes resultados que han invertido en nuevas tecnologías. De hecho, una de las direcciones más importantes en este desarrollo tecnológico ha sido la convergencia con Internet.

La unión de estas dos tecnologías, **Internet** y **telefonía móvil** va a dar como resultado un punto de inflexión en nuestra concepción del intercambio de información. La posibilidad de acceder a Internet en cualquier momento y lugar, puede ser el punto de partida de la consolidación de Internet como herramienta universal y necesaria para la mayoría de los ámbitos de vida [1].

La facturación por información obtenida y la transmisión de contenidos multimedia, unidas con la evolución imparable de los terminales, están abriendo mercados y negocios como el entretenimiento, comercio móvil, o acceso a información crítica en el tiempo. El comercio se centrará en compras compulsivas o que se necesiten en poco tiempo, la importancia de la información residirá en la necesidad de acceso en tiempo real, y el entretenimiento servirá para completar tiempos muertos.

Lo cierto es que las expectativas son muy favorables en el mercado de Internet móvil, aunque no debemos olvidar la turbulencia con la que este entorno se caracteriza. El factor tecnológico es la clave de los cambios que están sufriendo los mercados y las empresas. Las tecnologías tienen ciclos de vida desconocidos hasta la fecha, puesto que en cuestión de meses una tecnología aparece, se utiliza y se comprueba su éxito o fracaso [2].

El principal objetivo que persigue este documento es introducir al lector en la mayoría de las tecnologías móviles actuales y futuras. Por lo tanto, no se va analizar en profundidad todos los aspectos técnicos sino que el estudio se va a centrar en las características [3], ventajas y desventajas de cada tecnología. Además, también se reflexionará sobre las aplicaciones posibles que se pueden desarrollar, analizando el presente y futuro de los servicios móviles.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

2 MOTIVACIÓN

El presente documento se puede organizar claramente en tres grandes partes. En la primera que comprende del capítulo 4 al 8, se analizan las diferentes tecnologías de comunicaciones móviles. Son el equivalente en la telefonía móvil a RDSI o ADSL. La importancia de conocer estos sistemas es doble, primero porque son muy habituales las comparaciones entre las mismas y es imprescindible saber de cada una las ventajas e inconvenientes. La segunda razón hace referencia a la importancia de las capacidades y limitaciones de estas tecnologías de transmisión en la correcta implementación y uso de todo tipo de aplicaciones móviles.

La segunda parte se puede identificar con el capítulo 9. En este capítulo se sube un nivel de abstracción y se analizan todas las tecnologías y servicios que están disponibles sobre estas tecnologías móviles y que nos van a permitir con su uso y combinación crear aplicaciones móviles. Entre estas tecnologías podemos encontrar una gran variedad para mensajería móvil, todo tipo de técnicas para localizar geográficamente usuarios, así como tecnologías cliente-servidor que emulan el modelo de la *World Wide Web* y tecnologías que se ejecutan directamente en el terminal.

Finalmente, la tercera parte está incluida en el capítulo 10. Una vez vistas las tecnologías de transmisión móvil de datos y voz y las tecnologías que se montan encima, podremos analizar las aplicaciones finales que existen y van a aparecer en el entorno móvil. Evidentemente las posibilidades aquí son muy grandes, por lo que trataremos de ver varios ejemplos significativos y haremos especial hincapié en las ventajas que las tecnologías móviles aportan para el desarrollo de aplicaciones en contraposición con las aplicaciones “fijas”.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3 INTRODUCCIÓN A LOS SISTEMAS CELULARES

Antes de estudiar los sistemas modernos de comunicaciones móviles vamos a realizar una pequeña introducción a los sistemas celulares de telecomunicaciones, que nos va a proporcionar la base de conocimientos necesaria para poder entender con claridad el resto del capítulo.

Empezaremos repasando de forma breve la evolución histórica de las comunicaciones móviles, para posteriormente entrar en ideas y características comunes a todos los sistemas actuales y futuros.

3.1 Principios de las comunicaciones móviles celulares

3.1.1 Concepto de comunicaciones celulares

Básicamente, un sistema de telecomunicaciones móviles se puede definir como un conjunto de redes, servicios y aplicaciones que permiten a los usuarios transmitir voz y datos entre ellos y con otros servicios y aplicaciones permitiendo, además, la movilidad de los usuarios entre conexiones y durante ellas. Entre los objetivos que tiene un sistema de este tipo destacan:

- Proporcionar acceso a las redes de comunicaciones públicas
  Evidentemente, en la mayoría de sistemas de telecomunicaciones móviles se considera requisito imprescindible que los usuarios puedan hablar con otros que utilizan otras redes externas. En el caso de los datos también es necesario permitir comunicaciones con Internet o con otras redes de datos. En general, la red del sistema de telecomunicación móvil deberá poseer la capacidad de interconectarse con otras redes públicas de telefonía, con Internet y con servicios o aplicaciones de otras redes móviles [4].

- Permitir la movilidad de los usuarios
  La movilidad de los usuarios es una de las características básicas de las comunicaciones móviles. El sistema debe ser capaz de tener localizados a los usuarios para pasarles una llamada en cualquier momento de forma ágil.

- Proveer un servicio continuo en las zonas de cobertura
  El sistema deberá ser capaz de soportar una comunicación de datos o voz de forma continuada mientras el usuario se mueve a lo largo de las zonas de cobertura. Como veremos en capítulos posteriores no todos los servicios ni todas las calidades estarán disponibles a cualquier velocidad a la que se mueva el usuario.

- Proporcionar un grado de servicio aceptable
  En todos los servicios de telecomunicaciones hay una calidad de servicio mínima por debajo de la cual el sistema pierde su razón de ser. En el caso de las comunicaciones móviles este concepto cobra gran importancia puesto que se usa el canal radio cuya variabilidad es muy alta provocando subidas y bajadas de la señal. Un buen sistema de comunicaciones móviles tendrá en cuenta todos los efectos de propagación de las señales en el aire, proporcionando mecanismos que optimicen los datos recibidos por el terminal en la mayoría de escenarios.

3.1.2 Sistemas iniciales de comunicaciones móviles

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Los primeros sistemas de telecomunicaciones móviles que se desarrollaron se basaban en los mismos conceptos que la difusión terrenal de televisión. Existía un transmisor muy potente localizado en la zona más alta de la ciudad y que daba cobertura en un radio de 50 kilómetros. Estos sistemas datan de la década de los años 20 y los receptores que requerían eran de dimensiones muy superiores a los actuales.

Figura 0.53: Prototipo inicial de terminal móvil Fuente: Laboratorios Bell

En general este esquema de sistema centralizado de radiofrecuencia adolece de grandes inconvenientes:

- El espectro del que se dispone es limitado
  Siendo como es un recurso limitado, en la mayoría países el espectro radioeléctrico es repartido por organismos oficiales. Esto supone que los sistemas de comunicación en general no disponen de un espectro infinito sino que tienen un rango de frecuencias en el que trabajar. Para un sistema de comunicaciones móviles centralizado el problema reside en que a partir de un número de usuarios simultáneos el sistema no admite más pues se habrá quedado sin frecuencias disponibles.

- La presencia de otros usuarios introduce interferencias y reduce considerablemente la capacidad y/o la calida del servicio
  Si todos los usuarios comparten el mismo espacio radioeléctrico tendremos un claro compromiso entre la capacidad de la red y la calidad del servicio.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- La cobertura que proporciona el sistema está limitada por la potencia de los terminales
  Si consideramos un sistema de telecomunicaciones bidireccional, la cobertura del mismo estará limitada por la potencia de transmisión de los terminales móviles. En sistemas centralizados los terminales tenían que ser muy voluminosos para poder emitir la suficiente potencia y aún así tenían una autonomía muy limitada.

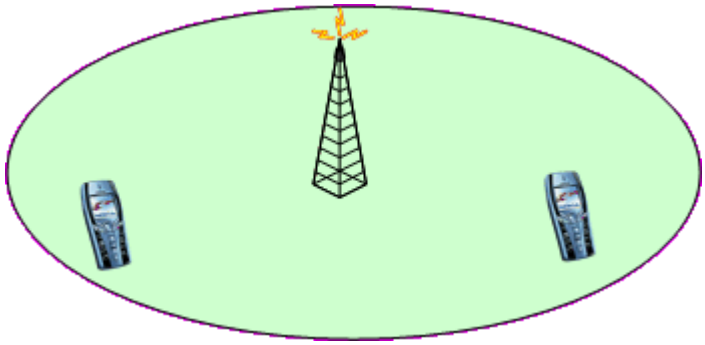


Figura 0.54: Sistema centralizado de telefonía móvil

3.1.3 Conceptos generales de los actuales sistemas de comunicaciones celulares

Para solucionar los problemas que plantea un sistema de comunicaciones móviles centralizado se empiezan a desarrollar los sistemas celulares. Este concepto estructura la red móvil de forma diferente. En vez de usar un único y potente transmisor, se sitúan muchos transmisores menos potentes a largo del área a cubrir. Estos transmisores sólo van a dar cobertura a una pequeña área alrededor de ellos llamada célula.

Este diseño de red permite solventar de forma considerable los problemas del sistema centralizado. Principalmente porque, como veremos, los usuarios de celdas contiguas usan diferentes frecuencias, permitiendo más capacidad y calidad con menos espectro. Además las distancias a la estación base se reducen con lo que los terminales móviles deben emitir menos potencia, con lo que resultan más pequeños y con mayor autonomía.

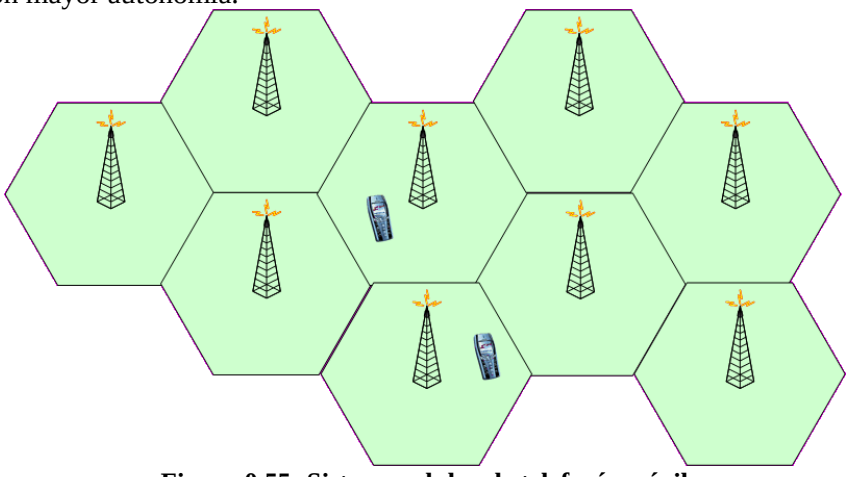


Figura 0.55: Sistema celular de telefonía móvil

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Antes de entrar en la arquitectura de un sistema celular vamos a establecer la definición de una serie de términos de gran trascendencia en este tipo de redes:

- Estación base
  Equipos asociados a un emplazamiento que utilizan una determinada tecnología de transmisión. Son el equivalente al transmisor centralizado que hemos visto antes.

- Emplazamiento
  Lugar físico en el que se asientan una o varias estaciones base.

- Célula
  Área de cobertura de una estación base o un sector. Ampliaremos más adelante este concepto cuando entremos a estudiar la arquitectura de un sistema celular.

- Sector
  Elementos de transmisión y recepción radio que comparten una misma estación base. Una estación base puede tener 1, 2, 3 ó 6 sectores.

- Portadora
  Unidad de radiofrecuencia para la transmisión y recepción de información. Un sector puede transmitir varias portadoras (también se les suele llamar frecuencias).

- Enlace descendente/ascendente
  El enlace descendente hace referencia a las comunicaciones desde la estación base al terminal móvil. En cambio, el enlace ascendente hace referencia a las comunicaciones desde el terminal a la estación.

A partir de ahora, todo nuestro estudio va a girar en torno a los sistemas celulares puesto que son los que más ventajas ofrecen y por lo tanto los más usados hoy en día.

3.2 Arquitectura general de un sistema celular

Como hemos visto, en vez de cubrir una zona con un solo transmisor de gran potencia, se suelen introducir muchos transmisores de menor potencia que dan cobertura a una zona limitada (células). De esta forma se consigue tener un sistema de mayor capacidad y que no obliga a los terminales móviles a transmitir una gran potencia. En compensación, deberemos controlar las interferencias entre las diferentes células.

En este apartado veremos a grandes rasgos la arquitectura básica que siguen todos los sistemas de telefonía celular.

3.2.1 Celdas o células

Una celda es una unidad geográfica de un sistema celular. El término celda proviene de las estructuras hexagonales que realizan las abejas en sus panales. Son áreas geográficas sobre las que transmiten y reciben las estaciones base y que normalmente se representan por hexágonos. De todas formas, por

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

las limitaciones que imponen la orografía del terreno y las construcciones humanas, la verdadera planta de una celda no suele coincidir con un círculo y mucho menos con un hexágono.

3.2.2 Clusters o racimo

Un *cluster* es una agrupación de celdas contiguas. Es la estructura básica que se va repitiendo en toda el área con cobertura. Su tamaño y diseño tiene mucho que ver el concepto de reuso de frecuencias, que vamos a ver a continuación y que es básico para entender la arquitectura de un sistema celular.

3.2.3 Reutilización de frecuencias

El concepto de telefonía celular está íntimamente ligado con la planificación de las portadoras o re-utilización de frecuencias. El espectro se divide siempre en una serie de canales o portadoras. La reutilización de frecuencias se basa en asignar a cada célula un grupo de portadoras que se va a usar en la zona de cobertura de la célula. Para evitar interferencias que complicarían las comunicaciones, las células vecinas nunca usarán las mismas frecuencias. El grupo de células que no reutiliza ningún canal es lo que hemos llamado antes *cluster*. Eso sí, células más alejadas pueden volver a utilizar el mismo conjunto de frecuencias maximizando la capacidad de la red a la vez que se optimiza el espectro utilizado.

En el siguiente gráfico se puede comprobar como se pueden ver el plan de reutilización más sencillo con un factor de 1/7.

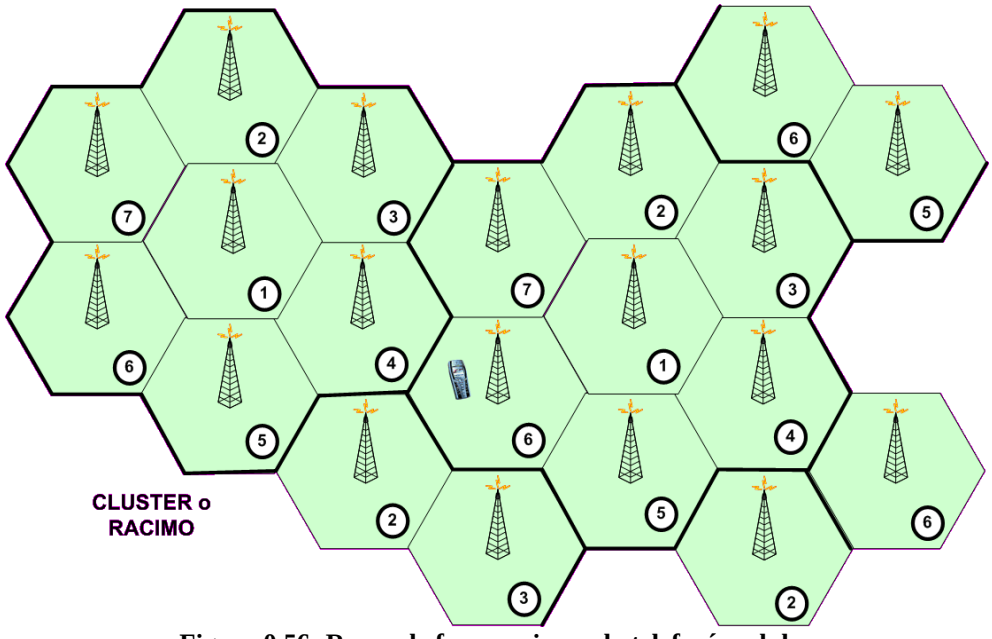


Figura 0.56: Reuso de frecuencias en la telefonía celular

Para maximizar la eficiencia de los planes de reutilización se utilizan diferentes mecanismos:

- Control de potencia

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Para minimizar las interferencias, la potencia utilizada en las transmisiones móviles será regulada según la distancia entre la estación base y el terminal. Al minimizar las interferencias con las células vecinas se pueden utilizar las mismas frecuencias más cerca manteniendo una calidad de servicio.

- Transmisión discontinua

En los momentos que el terminal detecte que no se está hablando dejará de transmitir potencia. De esa forma la interferencia será menor.

- Salto en frecuencia

Consiste en ir cambiando de frecuencia en cada trama transmitida. Además de eliminar interferencias, facilita la robustez de las comunicaciones respecto a desvanecimientos de duración mayor una trama.

- Sectorización

Una estación base puede transmitir diferentes frecuencias en diferentes direcciones (sectores) mediante antenas direccionales. Cuando sucede esto las estaciones base ya no se colocan en el centro de la célula sino que lo hacen en los vértices de la misma. Como se puede ver en el siguiente gráfico sólo tres de las seis estaciones que utilizan la frecuencia 5 interfieren. De esta forma la sectorización permite una mayor reutilización de frecuencias.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

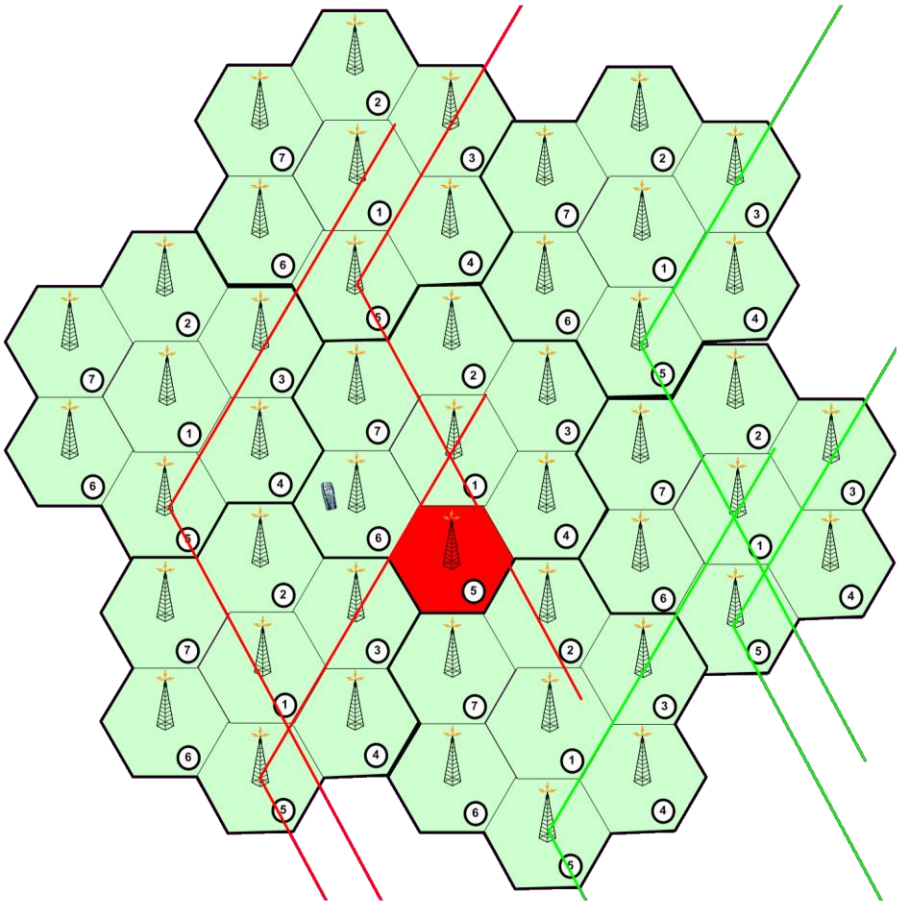


Figura 0.57: Uso de la sectorización en los sistemas celulares

Distintos factores hacen que los planes de reutilización teóricos dejen de ser aplicables de forma estricta en la práctica, sobre todo según se vayan considerando escenarios de planificación más extensos.

- Irregularidades en el terreno
- Distinta potencia de las estaciones base
- No se pueden aplicar de forma inmediata en escenarios que mezclen distintos tipos de estaciones base (trisectoriales, bisectoriales y omnidireccionales,)

3.2.4 División de celdas

Desafortunadamente, las consideraciones económicas y de diseño impiden llevar a la práctica el concepto de crear sistemas completamente compuestos por muchas células pequeñas de parecido tamaño. Para superar esta dificultad, los operadores introdujeron el concepto de división de celdas. Esta estrategia implica dividir en celdas más pequeñas aquellas que tienen una concentración mayor de usuarios. Su uso es especialmente evidente en zonas urbanas en las que la concentración de habitantes es muy heterogénea además de considerablemente variable en el medio plazo.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Esta estrategia de división de celdas produce una estructura jerárquica en el diseño del sistema celular, además permite una mayor reutilización de las frecuencias.

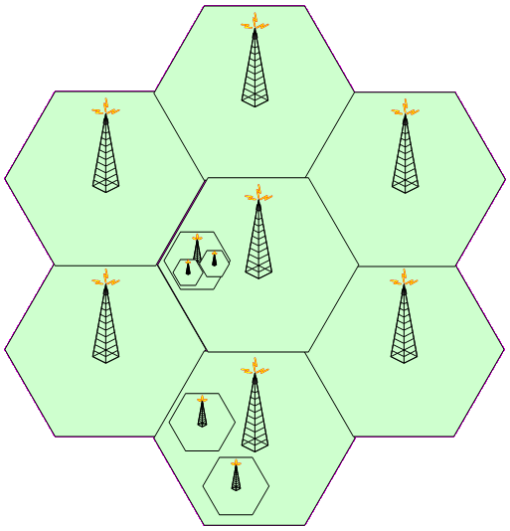


Figura 0.58: División de celdas

Suelen diferenciarse tres tamaños de células:

- Macroceldas: para zonas de cobertura grandes con usuarios de gran movilidad.
- Microceldas: para zonas urbanas reducidas (200-400 m) y usuarios de movilidad baja.
- Picoceldas: para cobertura de zonas interiores (70-80 m) y usuarios de movilidad reducida.

3.2.5 Traspasos (Handovers)

Finalmente, el último obstáculo en el despliegue de una red móvil celular tiene su origen en la posibilidad de que el usuario se mueva entre las células. Si sucede esto y no está utilizando su terminal para comunicarse, la red únicamente tendrá que llevar un registro de dónde se encuentra para poder localizarle en caso de recibir una llamada o mensaje.

Pero si el usuario está hablando la situación es bastante más complicada. En estos casos no sería aceptable que la comunicación se cortase cuando se cambiase de célula por lo que hay que proveer mecanismos para realizar el traspaso o *handover* de forma satisfactoria. El enfoque para solucionar este problema depende de la tecnología usada en la red móvil por lo que posponemos la explicación de las distintas soluciones a los correspondientes apartados en cada tecnología.

3.3 Propagación de señales

La señal de radio que viaja entre el terminal móvil y la estación base está expuesta a pérdidas cuando aumenta la distancia entre ambas, y se ve sometida en su camino a obstáculos y otras perturbaciones que van a provocar pérdidas aún mayores y desvanecimientos repentinos. A estos efectos, también hay que sumar las propias interferencias generadas por distintas señales que van a causar a partir de cierto nivel errores en la transmisión.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Se pueden distinguir cuatro fenómenos que afectan directamente a la propagación de señales radio en sistemas celulares.

3.3.1 Pérdidas debidas a la distancia

En la transmisión de ondas radio en el aire la atenuación debida a la distancia entre el transmisor y el receptor es proporcional al cuadrado de la distancia así como también al cuadrado de la frecuencia. Esto implica grandes diferencias de potencia de recepción según aumenta la distancia.

Pero en los sistemas celulares este condicionante no suele ser un problema grave debido a que cuando las pérdidas debidas a la distancia con la estación base son altas el terminal ya suele estar enganchado a otra estación base más cercana. De hecho, este fenómeno más que una desventaja es una facilidad para el diseño de las redes celulares puesto que provoca que las interferencias por otras fuentes de señal sean reducidas.

3.3.2 Obstáculos entre transmisor y receptor

Aparte de la distancia con la estación base, la señal se puede ver muy atenuada por obstáculos en el camino directo entre emisor y receptor. Este tipo de problemas suelen dar lugar a desvanecimientos lentos de la señal.

3.3.3 Multitrayecto

Cuando los obstáculos se encuentran cerca del receptor que lo suele suceder es que la señal se refleja en ellos llegando de diferentes maneras y en diferentes momentos al receptor. Esto provoca que el terminal móvil reciba una misma señal sumada varias veces pero con diferentes fases o retardos, causando que a veces el nivel de la señal presente mínimos (desvanecimientos) o máximos.

Pero este mecanismo también tiene su aspecto positivo puesto que permite recibir señales en zonas donde no hay visibilidad directa entre el terminal y la estación base (aunque también permite recibir interferencias que de otra forma no llegarían).

3.3.4 Desplazamiento Doppler

El efecto Doppler se produce por el desplazamiento del móvil respecto de la fuente de señal. De forma resumida se puede decir que produce desplazamientos rápidos y lentos por el cambio de fase y frecuencia que provoca.

Para solventar estos problemas y optimizar la transmisión radio del sistema se suelen emplear distintas técnicas que permiten reducir los problemas de la propagación de la señal radio, siendo los más comunes los siguientes:

3.3.5 Diversidad

Busca solventar el problema del multitrayecto mediante el envío de señales redundantes independientes. De esta forma la probabilidad de tener desvanecimientos en ambas es mucho más baja. Se suelen enviar señales separadas en el tiempo, frecuencia, espacio, polarización, etc.

3.3.6 Salto en frecuencias

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Ya vista en el apartado de reutilización de frecuencias, en este caso se utiliza para proporcionar robustez frente a desvanecimientos superiores a una trama. Se supone que en la siguiente trama la frecuencia va a ser otra y por lo tanto las condiciones de propagación muy diferentes.

3.3.7 Codificación

La codificación frente a errores consiste en mandar información redundante para reducir la probabilidad de error resultante. Evidentemente implican un menor ancho de banda para la información, pero permiten detectar y/o corregir determinados errores en las comunicaciones.

3.3.8 Entrelazado

El entrelazado se basa en transmitir los bits sin el orden original, es decir no consecutivamente. De esta forma, como los desvanecimientos suelen afectar a menudo a una cadena de bits consecutivos, se verán afectados sólo bits puntuales de diferentes sitios pudiendo la codificación de canal hacerse cargo de estos errores.

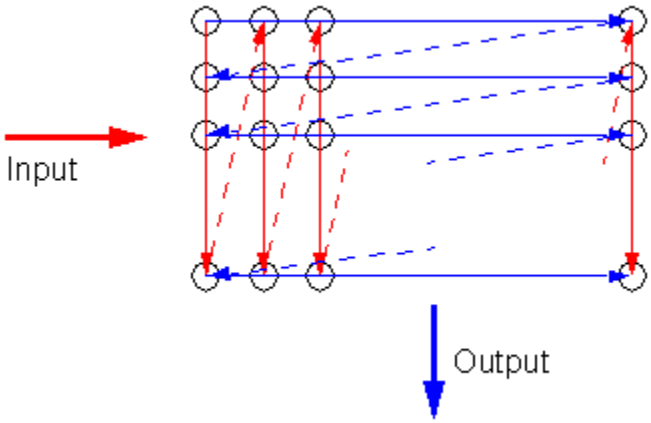


Figura 0.59: Diagrama del entrelazado de datos

3.4 Métodos de acceso múltiple al medio

Evidentemente, en las transmisiones radio tenemos un medio en el cual cualquiera puede transmitir y en el que se podrían producir colisiones continuas. En vez de utilizar algún sistema para gestionar estas colisiones, lo que se ha utilizado tradicionalmente son métodos de acceso múltiple que permiten dividir de alguna forma las comunicaciones para que las distintas transmisiones no choquen entre sí.

3.4.1 FDMA (Frequency Division Multiple Access)

El espectro disponible se divide en bandas (portadoras), cada una de las cuales se asigna a un enlace ascendente o descendente para cada usuario.

3.4.2 TDMA (Time Division Multiple Access)

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Varios usuarios escuchan en la misma frecuencia teniendo reservado para sus comunicaciones una determinada ranura (*slot*) temporal en la trama. Evidentemente, esta técnica sólo puede usarse en trasmisiones digitales.

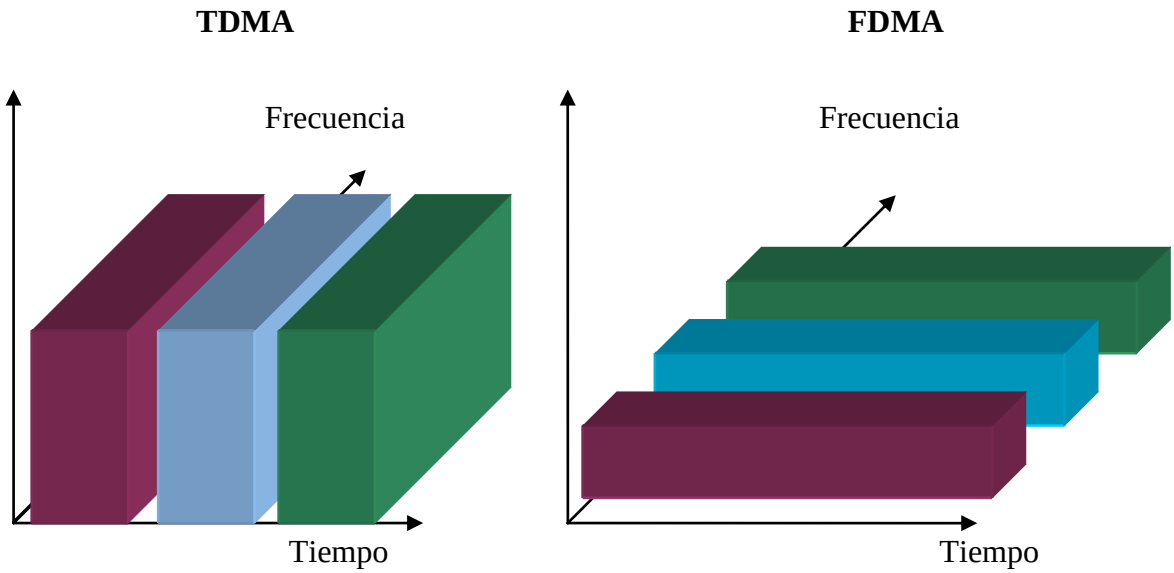


Figura 0.60: Comparación de métodos de acceso TDMA-FDMA

3.4.3 CDMA (Code Division Multiple Access)

Lo que se hace aquí para separar las comunicaciones y utilizar códigos para identificar cada una. Básicamente se multiplica la señal por una señal código y se transmite. En el receptor se utilizará ese mismo código para extraer la señal que le corresponde. La utilización de esta técnica produce unas consecuencias ciertamente interesantes que veremos con más detalle cuando analicemos el sistema UMTS.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

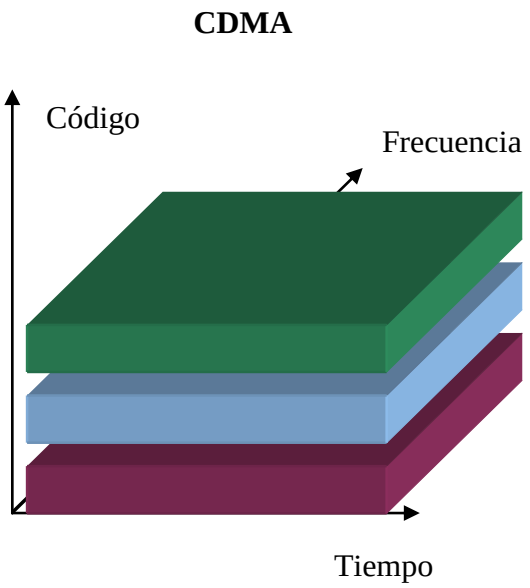


Figura 0.61: Método de acceso CDMA

En términos de complejidad, el más sencillo es el FDMA, siguiéndole el TDMA y finalmente el CDMA. Actualmente este último es en el que se están basando las redes de 3G en todo el mundo por su mayor eficiencia espectral.

3.5 Estándares de comunicaciones celulares

Una vez vistos los principios más generales de los sistemas de comunicaciones móviles celulares, es muy conveniente hacer un breve repaso por las distintas tecnologías que se han venido empleando para implantar el servicio de telefonía móvil. Aunque no vamos a entrar en detalle sobre ninguna de ellas, en sucesivos capítulos iremos viendo con más detalle las más importantes tecnologías que se han implantado en Europa (GSM/GPRS y UMTS)

3.5.1 Primera Generación (1G)

Los sistemas analógicos más importantes han sido sin lugar a dudas el AMPS norteamericano y los europeos NMT y TACS. Estos dos últimos sistemas han sido implantados en España aunque ambos están en desuso.

El sistema AMPS (*Advanced Mobile Phone Service*) nace en los Estados Unidos sobre el año 1974 y su utilización ha sido amplia en países como Australia, China, Canadá y varios países de Sudamérica. Se basa en la técnica de acceso al medio FDMA. Ha sido un sistema que ha estado en permanente evolución dando lugar ha diferentes versiones del mismo como *Narrowband* AMPS (NAMPS) que triplica la capacidad al reducir tres veces el ancho del canal utilizado, o el Digital AMPS (D-AMPS o ADC) que incluye TDMA dentro de los canales aumentando la capacidad en 6 veces.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

El sistema NMT surge en 1981 como un servicio normalizado en los países escandinavos (Suecia-Noruega-Dinamarca-Islandia). Puesto que es un sistema ideal para cubrir grandes extensiones de terreno con un coste moderado, aún se viene utilizando en ciertas regiones del norte de Europa como por ejemplo en Rusia. También se basa en FDMA y posee dos versiones, NMT 450, la más antigua y que opera en la banda de 450 MHz y NMT 900, más moderna y que trabajan entorno a los 900 MHz.

El sistema TACS 900 adoptado primeramente en Inglaterra en el año 1985 deriva del AMPS, lanzado comercialmente un año antes en Estados Unidos. Este sistema también utiliza FDMA pero la tecnología que usa es mucho más avanzada que la del NMT por lo que se obtiene una mejor calidad de audio así como una mejor gestión de los traspasos entre células. Una variante de este sistema, llamado ETACS, es el que se implantó en España con el nombre TMA 900 y con la marca MoviLine.

3.5.2 Segunda generación (2G/2.5G)

Sin lugar a dudas el sistema digital por excelencia de telefonía celular ha sido GSM. Se empieza a gestar en 1982 en el seno de la CEPT (*Conference Européenne des Postes et Telecommunications*) y por entonces sus siglas significaban *Groupe Special Mobile*. En 1991, la fase I del estándar se publica y a partir de hay su despliegue desborda todas las previsiones iniciales llegando a más de 160 países y cambiando su nombre por *Global System for Mobile Communications*. En España es el sistema que se ha venido usando mayoritariamente desde que se lanzó comercialmente en 1994.

En Estados Unidos en cambio no han tenido un único estándar digital, han ido pasando desde Interim Standard-54 (IS-54 o D-AMPS) hasta el IS-95 (CDMAone). Este último sistema fue una fuerte apuesta de la empresa californiana Qualcomm que introdujo en el mercado el primer sistema comercial con CDMA y que hacia competencia directa al europeo GSM. De hecho, hacia el año 2001 se estimaba que había llegado al 10% de los usuarios móviles mundiales.

Como puente entre la segunda generación y la tercera han ido apareciendo sistemas intermedios que incorporados sobre las redes digitales 2G proveen servicios de transmisión de datos a mayor velocidad y con conmutación de paquetes. Este último punto es especialmente importante puesto que permite compartir el canal de transmisión de datos y facturar exclusivamente el tráfico generado y no el tiempo de sesión. Entre estos sistemas 2,5G destacan sobre todo GPRS (*General Packet Radio Service*) y EDGE (*Enhanced Data rate for GSM Evolution*) [3].

3.5.3 Tercera generación (3G)

La tercera generación de móviles significa un salto considerable respecto de los sistemas actuales. Está pensada para transmisión de datos a alta velocidad a través de técnicas avanzadas de conmutación de paquetes y circuitos, soporta tecnología IP y posibilita el acceso a aplicaciones multimedia móviles entre otras cosas. El nuevo modelo de negocio es radicalmente distinto al actual y entran en juego nuevos agentes, como son los proveedores de contenidos y los proveedores de aplicaciones, entretenimiento, etc.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Todos los sistemas 3G se basan en CDMA. Aunque hay dos grandes familias, las que siguen las tecnologías WideBand CDMA (W-CDMA) y CDMA2000.

W-CDMA es la tecnología en la que se basan el estándar europeo UMTS (*Universal Mobile Telephony System*) y el japonés FOMA (*Freedom of Mobile Multimedia Access*). En cambio el estándar americano se basa en CDMA2000 y es una evolución de la tecnología 2G CDMAone.

3.5.4 Futura evolución (4G)

Mientras que la implementación de la tercera generación ha sufrido los problemas bien conocidos asociados a los efectos de la recesión y de una planificación inadecuada, a los que se añaden los altos precios pagados por las licencias y las tasas de utilización del espectro radioeléctrico, la 4G aparece ya como una alternativa razonablemente clara que se espera que se despliegue sobre el año 2010 con características tecnológicas superiores a la tercera generación.

Como requisitos más destacables aparece la interconexión con diferentes redes inalámbricas como WLAN, una red de satélites u otras redes celulares. También se esperan velocidades de hasta 100 Mbps y acceso con un único equipo y factura a diferentes servicios y aplicaciones.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

4 GSM

4.1 Historia de GSM

Durante el comienzo de la década de los 80, los sistemas celulares analógicos estaban experimentando un rápido crecimiento en Europa, especialmente en los países escandinavos y Inglaterra, aunque también en Francia y Alemania. Casi cada país había desarrollado su propio sistema que además era incompatible con el resto tanto en equipos como en procedimientos.

La situación distaba de ser ideal, no sólo porque los usuarios móviles estaban restringidos a utilizar sus equipos dentro de sus fronteras, sino que también porque los fabricantes de los sistemas no podían conseguir las deseadas economías de escala que ayudarían a reducir precios y aumentar eficiencia.

Europa se dio cuenta de la importancia de desarrollar un estándar común y ya en 1982, en la Conferencia Europea de Correo y Telecomunicaciones (CEPT *Conference Européenne des Postes et Telecommunications*) forma un grupo de estudio para analizar y desarrollar un sistema de telecomunicaciones móvil celular para toda Europa. El nombre de este grupo es el que originalmente se utilizó para formar las siglas GSM (*Group Spécial Mobile*). El estándar que se iba a desarrollar debía cumplir los siguientes criterios y calidades:

- Buena calidad subjetiva de sonido con la voz humana
- Costes de servicio y terminales bajos
- Soporte para *roaming* internacional (capacidad de los usuarios de conectarse con redes en países extranjeros)
- Eficiencia espectral
- Compatibilidad con RDSI
- Soporte para una gran gama de servicios y facilidades de red

En 1989, la responsabilidad del desarrollo del estándar GSM pasó a manos del *European Telecommunication Standard Institute* (ETSI), y finalmente en 1990 se salió a la luz la fase I de las especificaciones de GSM. El servicio comercial empezó a mediados de 1991 y a para el año 1993 ya existían 36 redes basadas en GSM en 22 países diferentes.

Aunque como hemos visto el proceso de estandarización fue realizado por instituciones europeas, GSM no se puede considerar únicamente como una tecnología específica de este continente. Más de 200 redes están actualmente en servicio en por lo menos 110 países alrededor del globo. A comienzos de 1994 ya existían más de 1,3 millones de usuarios de GSM en todo el mundo, número que creció hasta los 55 millones en sólo tres años. Con Norteamérica usando la versión derivada de GSM llamada PCS1900, se puede decir que la tecnología GSM está actualmente funcionando en todos los continentes.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Figura 0.62: Evolución de los usuarios GSM Fuente: EMC World Cellular Database

En España la tecnología GSM entra en 1995 de la mano de Telefónica (MoviStar) y Airtel. Posteriormente se les uniría Amena en 1999. La cobertura actual de estos sistemas es cercana al 100% respecto a la población, aunque geográficamente es algo más baja, puesto que hay zonas de muy difícil cobertura.

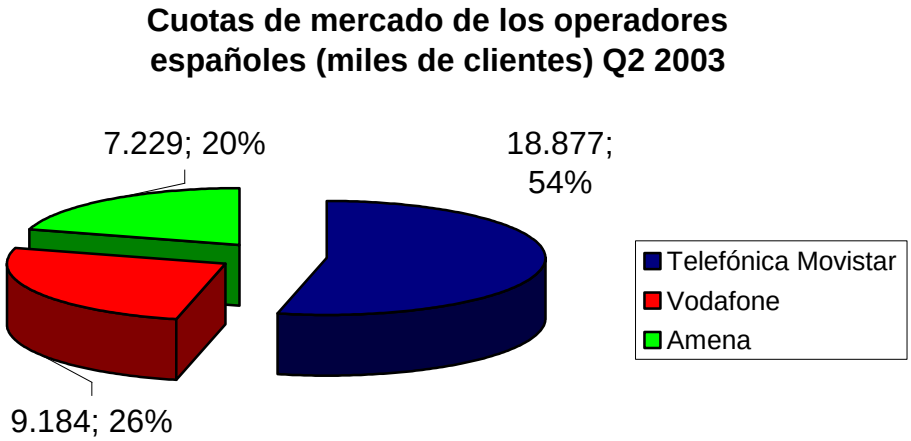


Figura 0.63: Cuotas de mercado de los operadores móviles en España Datos: Expansión

4.2 Servicios proporcionados por GSM

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Desde el principio, los arquitectos de GSM querían compatibilidad con el sistema digital de comunicaciones fijas, RDSI, en cuanto a los servicios ofrecidos y la señalización de red utilizada. Sin embargo, la limitaciones de la transmisión radio respecto a ancho de banda y coste de transmisión, no han permitido conseguir en la práctica la tasa de 64 kbps que tiene un canal B del estándar de RDSI.

Los servicios que GSM proporciona a los usuarios se clasifican de la misma forma que en RDSI, es decir usando las definiciones creadas por la ITU-T. Estos servicios se dividen en teleservicios, servicios portadores y servicios suplementarios.

4.2.1 Teleservicios

- Telefonía

Servicio modo circuito similar al de la red telefónica conmutada o la RDSI, que permite la conversación con abonados GSM o de otras redes telefónicas. La voz se digitaliza y comprime de modo que el flujo de información que se transmite sobre el interfaz radio es de 13 kbps o 6,5 kbps, según sea *full rate o half rate*.

- Llamadas de Emergencia

Este servicio permite efectuar llamadas de emergencia mediante la marcación de un número de tres cifras (Ej. 112). Se trata de un servicio prioritario, obligatorio para toda la red GSM y que agiliza el tratamiento de estas llamadas hacia el centro de atención adecuado (policía, bomberos, etc.)

- Servicio de Mensajes Cortos (SMS)

Servicio modo paquete que permite el intercambio de mensajes alfanuméricos de hasta 160 caracteres (codificados a 7 bits por carácter) entre terminales GSM o desde la red hacia los terminales.

- Servicio de fax

Permite el envío y recepción de documentos facsímile grupo 3. Si bien existen teléfonos GSM que incluyen facilidades de fax y adaptadores GSM para terminales facsímile G3, en el caso más habitual se emplea un PC con una tarjeta PCMCIA (módem-fax) a la que se conecta un teléfono GSM.

4.2.2 Servicios portadores

GSM ofrece servicios portadores para transmisión de datos hasta 9.600 bps.

4.2.3 Servicios suplementarios

GSM soporta servicios suplementarios similares a los de la RDSI. Por ejemplo:

- Desvíos de llamadas

Las llamadas pueden redirigirse a otro número o a un buzón de voz si el abonado está ocupado, no contesta o es inalcanzable (terminal desconectado o sin cobertura).

- Identificación de abonado llamante y de abonado conectado

La pantalla del móvil muestra respectivamente el número de abonado que llama o el número destino de la llamada.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- Llamada en espera

Durante la conversación se puede dar paso a una nueva llamada inhibiendo la actual. Por supuesto luego se puede retomar.

- Prohibición de llamadas

Puede aplicarse a llamadas entrantes o salientes (ej. llamadas internacionales o a servicios de valor añadido).

4.3 Arquitectura de la red

La red GSM está compuesta de diversos componentes cuyas funciones e interfaces están especificados en el estándar. En la siguiente figura podemos observar los principales elementos de la red.

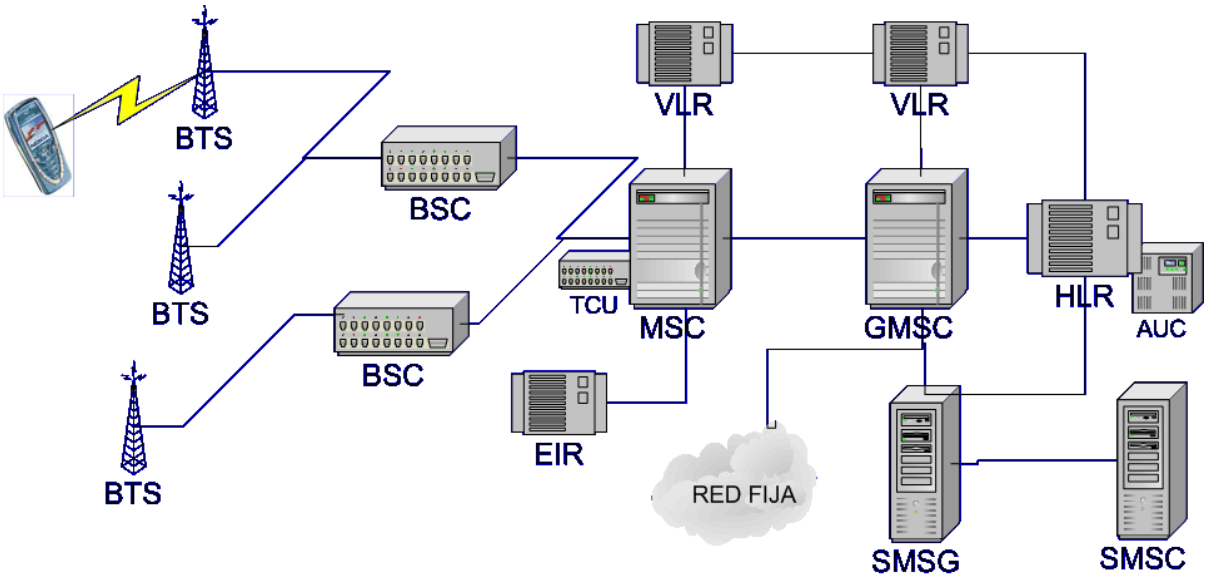


Figura 0.64: Arquitectura de la red GSM

Uno de los aspectos más importantes de la arquitectura de las redes GSM es que la red de acceso tiene una organización jerárquica, cada elemento controla un conjunto de elementos de nivel inferior y a su vez es gestionado por otro de nivel superior. Esto facilita bastante tanto el diseño y la planificación de la red, así como su gestión y ampliación.

En las siguientes páginas vamos a ir repasando uno por uno los principales elementos de la arquitectura de red, estudiando además los interfaces que los interrelacionan.

- Estaciones móviles

En GSM se entiende como estación móviles (MS, *Mobile Station*) el conjunto formado por el terminal móvil del usuario y la tarjeta SIM (*Subscriber Identity Module*). Sin la tarjeta, el terminal sólo permite realizar llamadas de emergencia. Cualquier acceso al resto de servicios que proporciona una red GSM requiere la inserción de una tarjeta SIM en el terminal.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Todo terminal GSM tiene un número de serie (IMEI, *International Mobile Equipment Identifier*) que lo identifica de forma internacional. El IMEI permite a los operadores de redes GSM controlar y gestionar el acceso de los terminales a sus redes. Podría impedir el acceso a terminales que no han sido homologados o no cumplen determinadas especificaciones técnicas. Además, y esto sí que se realiza actualmente, los operadores pueden bloquear y dejar inutilizados terminales móviles que hayan sido declarados como robados mediante su identificación con el IMEI.

La SIM es una tarjeta inteligente que tiene capacidad para almacenar, entre otras informaciones, un identificador universal del usuario GSM (IMSI, *Internacional Mobile Subscriber Identity*) así como una clave secreta para la autenticación con la red. El acceso a la información contenida en la tarjeta SIM está protegido mediante un código personal de acceso (PIN, *Personal Identification Number*) que se solicita al encender el móvil.

Las tarjetas están provistas de una memoria adicional que proporciona facilidades de agenda electrónica al usuario, por ejemplo permitiéndole almacenar números de teléfono. Además también suelen tener cierta memoria reservada para almacenar mensajes de texto del usuario. Según ha ido evolucionando GSM las tarjetas SIM han ido aumentando en capacidad y funcionalidad. No sólo poseen más memoria, sino que además las últimas tarjetas han incluido una funcionalidad llamada *SIM toolkit*, que permite a los operadores programar las tarjetas para que en los terminales que lo soportan aparezcan menús de acceso rápido personalizados. De esta forma se consigue que los usuarios tengan de forma rápida y sencilla un menú de acceso rápido con los servicios que el operador considera más interesantes.

Ambos identificadores, IMEI e IMSI, son independientes, por lo que permiten la movilidad de los usuarios respecto a los terminales que usan. De esta forma un terminal puede ser usado por diferentes usuarios sin que existan conflictos a la hora de identificar al cliente o tarificar las llamadas. Simplemente basta que el usuario inserte su tarjeta SIM en el terminal.

Las estaciones móviles se comunican con las estaciones base (BTS) mediante el interfaz radio *Um*. Este es el único interfaz de toda la arquitectura GSM que se realiza vía radio, siendo muy importante su comprensión para entender gran parte de las características que GSM posee.

- Interfaz Um

Los canales de comunicaciones radio en GSM tienen un ancho de banda espectral de 200 KHz. Cada uno de estos canales tiene una frecuencia central llamada portadora que se equiespacian en las bandas reservadas para el estándar GSM.

Una célula GSM puede tener asignados uno o más pares de frecuencias portadoras. Cada operador debe decidir el número de portadoras a utilizar en cada célula según el tráfico que ésta vaya a soportar y el número de portadoras que le hayan sido asignadas por el regulador competente.

Cada par de frecuencias está formado por una en la banda ascendente (de MS a BTS) y otra en la descendente (de BTS a MS) con el fin de hacer posible la comunicación bidireccional simultánea.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En general, siempre que se mencione un número de frecuencias se sobreentiende pares de portadoras. Por ejemplo, en la frase “esta célula tiene asignadas 3 frecuencias” se debe considerar que quiere decir que la célula tiene asignadas tres portadoras en la banda ascendente y tres en la banda descendente.

En la versión GSM 900, la interfaz radio tiene reservada una banda de frecuencias en torno a los 900 MHz. El ancho de banda asignado está estructurado en 125 portadoras. La mitad inferior de la banda (890-915 MHz) está destinada a enlaces ascendentes y la superior (935-960 MHz) está destinada a los descendentes.

Más recientemente ha sido asignada una nueva banda en torno a 1.8 GHz, para la variante de GSM conocida como DCS 1800 (*Digital Cellular System*). Al igual que GSM 900, se pueden diferenciar dos bandas, una ascendente (1710-1785 MHz) y otra descendente (1805-1880 MHz).

Como vimos en el capítulo de introducción a las tecnologías celulares, este tipo de acceso al medio compartido se denomina FDMA y consiste en la multiplexación de las telecomunicaciones mediante el uso de distintas frecuencias. En GSM no sólo se utiliza este tipo de acceso al medio. Además en cada frecuencia (o par de frecuencias) se transmiten digitalmente tramas TDMA con 8 intervalos de tiempo (*Time Slots*) que permite multiplexar en el tiempo 8 canales físicos por frecuencia. Sobre un par de frecuencias dado, el canal físico bidireccional i, con i=1, 2,.., 8, estará formado por el intervalo de tiempo i en cada una de las tramas TDMA, tanto en sentido ascendente como descendente.

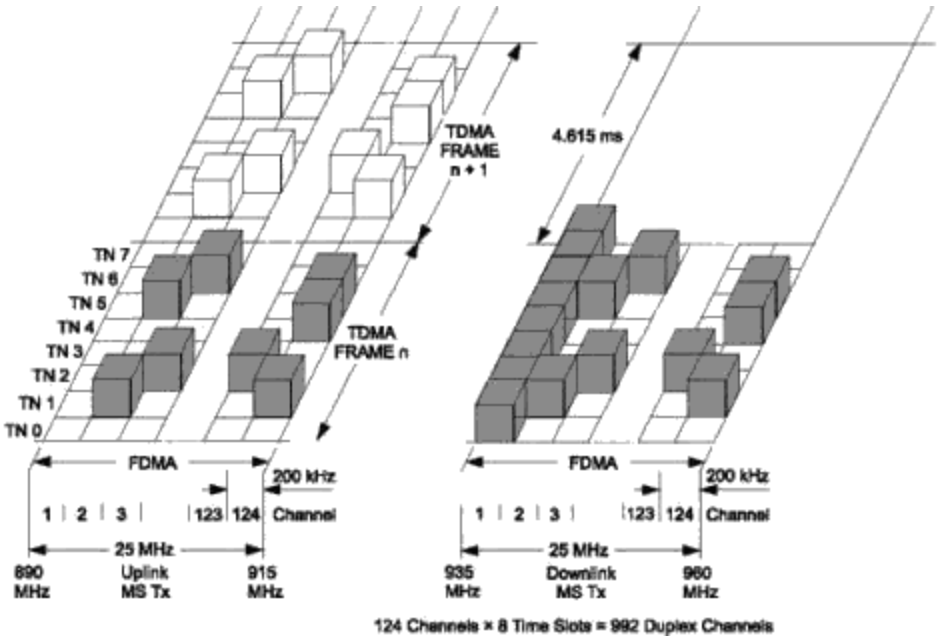


Figura 0.65: FDMA y TDMA en GSM

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Cada uno de estos canales físicos se puede utilizar para transmitir datos o voz o puede ser utilizado para transmitir señalización. Además hay que destacar que entre el intervalo de tiempo en sentido ascendente y el correspondiente intervalo en sentido descendente hay un desfase temporal para evitar que el terminal tenga que simultáneamente transmitir y recibir datos. La estructura de estos canales físicos es relativamente compleja y por lo tanto sólo vamos a estudiar por encima las capacidades de transmisión que soportan estos canales en las diferentes posibilidades que existen.

Cada intervalo de tiempo de la trama tiene una duración de 15/26 ms (~ 0,577ms). En consecuencia, una trama dura 120/26 ms (~ 4,615 ms), lo que equivale a decir que sobre una portadora se transmiten 26 tramas cada 120 ms.

Estos intervalos de tiempo en el que se divide una trama se denominan períodos de ráfaga (*burst periods*) debido a que dentro de cada intervalo se pueden transmitir una ráfaga de bits. El número de bits por ráfaga puede oscilar entre 88 y 148 según el tipo de información a transmitir (datos, señalización,…). De estos bits, una parte está destinada al transporte de información de usuario más redundancia para protección de errores, correspondiendo el resto a bits de guarda y sincronismo.

Las ráfagas normales, utilizadas para tráfico de usuario (voz/datos) y para buena parte del tráfico de señalización, son de 148 bits. De estos 148 bits, 114 son de información más protección de errores. Teniendo en cuenta este datos, un canal físico da una capacidad bruta de 114 bits/4,615 ms = 24,7 Kbps.

El empleo de un canal físico para una conversación de voz o para datos requiere la consideración de agrupaciones de 26 tramas, denominadas multitramas. Según lo visto, la duración de una multitrama de 26 tramas es de exactamente 120 ms. De los 26 intervalos de tiempo por multitrama que corresponden al canal físico que consideremos, sólo se puede enviar voz (o datos) en 24. Por lo tanto la capacidad anteriormente calculada se debe multiplicar por el factor 24/26.

La capacidad neta disponible es aún menor puesto que en los 114 bits están incluidos los bits de redundancia para la protección frente errores de canal (con códigos específicos según se trate de voz o datos que se salen del ámbito de esta documentación). En el caso de la voz, de cada 456 bits sólo 260 son de información y el resto es de redundancia. El resultado de descontar de la capacidad bruta antes calculada los dos intervalos no utilizados en la multitrama y la redundancia es:

$$24{,}7 \cdot \frac{24}{26} \cdot \frac{260}{456} = 13 Kbit/s$$

En el caso de los datos la proporción entre la información y la redundancia es similar, resultando que la máxima velocidad de transmisión que se puede alcanzar es 9,6 kbps.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

4.3.1 Subsistema de estaciones base

En la arquitectura GSM, se denomina subsistema de estaciones base (BSS, *Base Station Subsystem*) al conjunto constituido por las estaciones base (BTS, *Base Transceiver Station*) y sus controladores (BSC, *Base Station Controller*).

- BTS

Las BTSs son los elementos de red que contienen las antenas y los equipos necesarios para las comunicaciones radio con las estaciones móviles. Como se ha visto, una BTS puede tener asignada uno o varios pares de frecuencias portadoras, en función de las necesidades de tráfico que se estima que puede tener la célula.

En un área urbana, existe la posibilidad que sean necesarias un gran número de BTSs, por lo que sus requisitos son:

- Robustez
- Fiabilidad
- Portabilidad
- Mínimo coste

- BSC

Los BSCs son equipos que, como su nombre indica, sirven para controlar estaciones base. Un BSC puede gestionar una o varias BTSs (hasta varias decenas, según el fabricante). Sus principales funciones son la gestión de recursos radio (asignación de canales a MSs) y la gestión de traspasos (*handovers*) entre las BTSs que controla. Esta última facilidad es la que permite el mantenimiento de una comunicación cuando una MS cambia de célula durante el transcurso de una llamada.

Como ya hemos visto el interfaz que permite las comunicaciones radio entre MSs y BTSs es el interfaz Um mientras que las BTSs se comunican con las BSCs con el interfaz Abis.

- Interfaz Abis

Las comunicaciones entre BTS y BSC se realizan a través de sistemas digitales convencionales de 2 Mbps, en los que uno o más canales de 64 kbps se emplean para señalización y el resto para voz y datos.

Con el propósito de aprovechar mejor los 64 kbps disponibles en los canales de tráfico, éstos se dividen en 4 subcanales de 16 kbps, ya que esta capacidad es suficiente para el transporte del flujo de información intercambiado con una MS. De este modo, un mismo canal de 64 kbps puede soportar cuatro conversaciones de voz o datos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

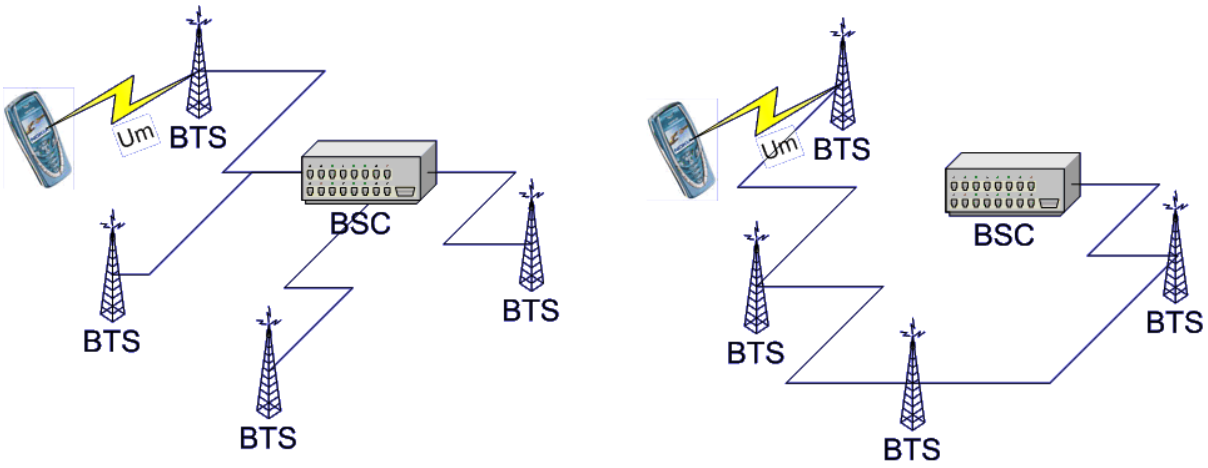


Figura 0.66: Comunicación en línea y en estrella entre BTSs y la BSC

La interconexión entre una BSC y las BTSs que controla puede efectuarse en estrella o en cadena tal y como se puede ver en la figura anterior. La ventaja de la configuración en cadena es que se pueden compartir canales de 64 kbps en las líneas de 2 Mbps. De esta forma se puede ahorrar en la interconexión de los elementos de red que típicamente es un factor de coste muy importante para un operador.

4.3.2 Subsistema de conmutación

Dentro de la arquitectura GSM, se denomina Subsistema de Conmutación (NSS, *Network Switching System*) al conjunto formado por las centrales de conmutación (MSC, *Mobile Switching Centres*) y los registros de información. Además, dentro de este subsistema se suelen incluir los Centros de Mensajes Cortos (SMSC) así como la pasarela SMSG que comunica los SMSC con el resto del sistema GSM.

- MSC

Las MSCs son centrales similares a las utilizadas en las redes telefónicas fijas, con facilidades adicionales para el soporte de funciones específicas de las redes GSM (soporte de movilidad, traspasos, autenticación, etc.). Cuando una MSC actúa de pasarela con la red fija, se dice que la MSC tiene funciones de GMSC (*Gateway MSC*).

Al igual que en la RTC o la RDSI, las MSCs se comunican entre sí mediante enlace SS7 (Sistema de Señalización nº 7) y circuitos telefónicos convencionales a 64 kbps. También se comunican, a través del interfaz A con los BSCs que dependen de ellas.

- Interfaz A

El interfaz A que comunica BSCs con MSCs es prácticamente igual que el interfaz Abis ya visto. La diferencia es que la conmutación en las MSCs se efectúa sobre circuitos convencionales de 64 kbps. La adaptación de velocidades (de 16 kbps a 64 kbps y viceversa) se efectúa en las denominadas unidades transcodificadoras (TCU, *Transcoder Units*), que normalmente se sitúan al lado de las MSCs.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- Registros de información

El tratamiento de llamadas en GSM requiere la consulta por parte de las MSCs de diferentes registros de información que no son más que bases de datos. Los principales registros son:

- Registro de Localización Base (HLR, *Home Location Register*)

  Contiene la información de tipo administrativo sobre los abonados de la red (su identidad, servicios contratados, etc.). También contiene el puntero que indica la localización de cada MS de la red, expresada como el VLR en el que se encuentra registrada la MS en un momento dado. Conceptualmente, existe un único HLR por la red, si bien físicamente puede realizarse como una base de datos distribuida.

  Vinculado al HLR aparece el Centro de Autentificación (AuC, *Authentication Center*) que gestiona de manera centralizada los parámetros relacionados con la seguridad y privacidad de las comunicaciones en la red GSM.

- Registro de Localización de Visitantes (VLR, *Visitor Location Register*)

  Almacena información temporal de las MSs que se encuentran dentro de un área cubierta por una (lo habitual) o más MSCs. Contiene la información de localización más precisa que el HLR, indicando un área de localización (LA, *Location Area*), esto es, un conjunto de células entre las que se encuentra una MS dada.

- Registro de Identidades de Equipos (EIR, *Equipment Identity Register*)

  Base de datos en la que el operador puede almacenar información relativa a terminales, identificándolos a través de sus IMEIs. El EIR pude contener, por ejemplo, una relación de IMEIs correspondientes a terminales robados, de manera que se prohiba su utilización en la red.

- Otros interfaces

Aparte de los interfaces ya estudiados existen otros también normalizados (B, C, etc.) entre los distintos elementos de la red GSM. Sobre estos interfaces se desarrollan los intercambios de señalización necesarios para el control de llamadas, la localización y traspaso de llamadas, la autenticación de usuarios, etc.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

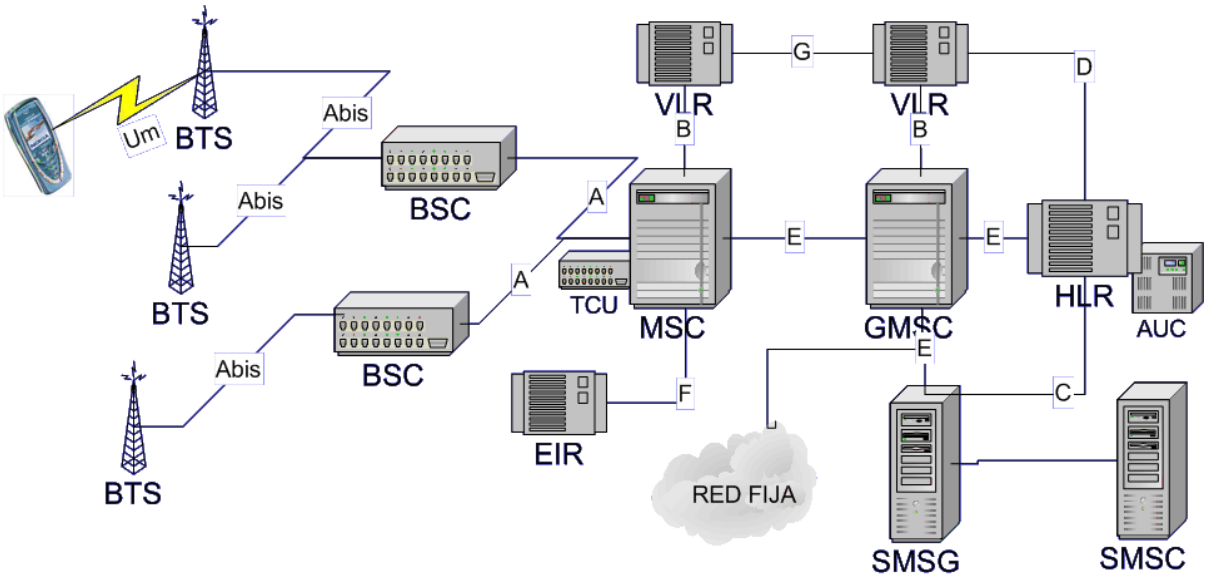


Figura 0.67: Arquitectura global de la red GSM con interfaces

4.4 Interfaz radio

Como ya hemos visto, la forma que tiene GSM de dar acceso a diferentes usuarios a un espectro compartido es utilizando simultáneamente TDMA y FDMA. Cada célula tiene distintos pares de frecuencias que a su vez se dividen en 8 canales físicos dividendo el tiempo de transmisión equitativamente entre cada uno de ellos [4].

Ese es el soporte físico de la transmisión GSM pero por encima aparecen un conjunto de mecanismos que forman el interfaz radio. Todo el diseño de este interfaz tuvo como objetivo la eficiencia espectral, a costa en algunos casos de cierta complejidad como en la estructura de canales lógicos. Debido al carácter introductorio de este capítulo no vamos a entrar en detalle a explicar el interfaz radio sino que vamos a dar nociones de cada uno de los mecanismos más importantes que lo componen.

4.4.1 Estructura de canales lógicos

Los canales lógicos se componen de una sucesión de tramas TDMA (formadas por 26 o 51 intervalos de tiempo). Se dividen en canales dedicados, que se asocian a una estación móvil en concreto, y en canales comunes que son usados por todas las estaciones móviles simultáneamente. Además, se dividen en canales de tráfico y de señalización.

El canal de tráfico por excelencia es el TCH (*Traffic Channel*), que se usa para transmitir voz y datos. Uno de los intervalos de este canal en cada trama se usa para transmitir señalización mediante el canal llamado SACCH (*Show Associated Control Channel*). Otros canales dedicados son el SDCCH (*Stand-alone Dedicated Control*) que se usa entre otras cosas para la señalización durante la fase inicial del establecimiento de llamada y para mensajes cortos y el FACCH (*Fast Associated Control*) que es el canal de señalización usado para completar el establecimiento y para los traspasos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Además de estos canales dedicados, hay otra colección más amplia de canales comunes dedicados a la señalización y entre los cuales destacan:

- BCCH (*Broadcast Control Channel*)
  Está continuamente emitiendo en el enlace descendente información sobre el identificador de la estación base, las frecuencias de la misma, y las secuencias del mecanismo de salto en frecuencia descrito más adelante.
- FCCH (*Frequency Correction Channel*) y SCH (*Synchronisation Channel*)
  Se usan para sincronizar la estación móvil a la estructura de los intervalos de tiempo definiendo sus límites y la numeración de los mismos.
- RACH (*Random Access Channel*)
  Canal con acceso instrumentado con Aloha ranurado que usan las estaciones móviles para el acceso a la red.
- PCH (*Paging Channel*)
  Lo usa la estación base para alertar a una estación móvil en concreto de que tiene una llamada entrante desde la red.
- AGCH (*Access Grant Channel*)
  Se usa para indicar la reserva de un SDCCH a una estación móvil para la señalización con el objetivo de obtener un canal dedicado. Su uso es consecutivo en el tiempo a una petición mediante un canal RACH.

4.4.2 Codificación de canal y modulación

Debido a las interferencias electromagnéticas naturales y generadas por el ser humano, la señal transmitida por el interfaz radio debe ser protegida contra los errores. En este apartado GSM utiliza códigos convolucionales y entrelazado de bloques para conseguir esta protección.

4.4.3 Ecualización

En el rango de los 900 MHz, las ondas de radio rebotan en casi todo, edificios, colinas, coches, etc.… Por lo tanto, multitud de señales reflejadas, cada una con un retardo determinado, pueden llegar a la antena. La ecualización se usa para extraer la señal deseada de todas las reflejadas que nos molestan para tener una recepción limpia.

El funcionamiento de la ecualización se basa en encontrar cómo el camino que recorre la señal modifica una transmisión conocida de ante mano y construir un filtro inverso que deshaga estos cambios. Esta señal ya conocida son 26 bits que se transmiten en cada trama TDMA y que se pueden considerar como una secuencia de entrenamiento para que la estación móvil pueda ecualizar la señal recibida optimizándola.

4.4.4 Salto en frecuencia

GSM utiliza el mecanismo de salto en frecuencia para minimizar el problema de interferencias y desvanecimientos en forma de ráfagas. Puesto que las características de propagación de una señal dependen de la frecuencia con la que se transmite, modificando ésta en cada trama TDMA obtenemos cierta protección contra desvanecimientos continuos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

La secuencia de cambio de las frecuencias utilizadas es continuamente retransmitida desde la estación base mediante el canal BCCH.

4.4.5 Transmisión discontinua

Minimizar la interferencia entre canales es un objetivo a optimizar en todos los sistemas celulares, puesto que permite dar un mejor servicio y aumentar la capacidad de todo el sistema.

La transmisión discontinua se aprovecha del hecho de que una persona habla menos del 40% del tiempo en una conversación normal y apaga la transmisión durante los periodos de silencio. Un beneficio añadido a esta funcionalidad es que permite un mejor aprovechamiento de la batería de los terminales móviles.

El componente más importante de la transmisión discontinua es, evidentemente, la detección de voz. Debe distinguir entre la voz y el ruido, una tarea en absoluto trivial a pesar de lo que pueda parecer. Hay que considerar que el ruido de fondo puede tener un potencia considerable. Si la señal de voz es malinterpretada como ruido, el trasmisor se apagará y se producirá un efecto muy desagradable al cortarse la comunicación. Además, si el ruido es interpretado como voz la eficiencia de la transmisión discontinua decrecerá rápidamente.

Otro factor a tener en cuenta es que cuando el terminal deja de trasmitir, el silencio que recibe el otro lado es absoluto debido a la naturaleza digital de GSM. Esta falta total de ruido no es nada recomendable puesto que suele generar la sensación de que la comunicación se ha cortado. Para evitar este problema, cuando no se transmite el receptor generará un ruido que tratará de suplir el ruido de fondo de la conversación.

4.4.6 Recepción discontinua

Otro método para conservar la potencia de una estación móvil es la recepción discontinua. El canal PCH, usado por la estación base para señalizar una llamada entrante, se estructura en subcanales. Cada estación móvil necesita escuchar solamente su propio subcanal. En el tiempo entre los sucesivos subcanales que no le afectan, el terminal puede entrar en un modo de espera, en el que apenas se gasta energía.

4.4.7 Control de potencia

Para minimizar la interferencia entre canales y optimizar la duración de las baterías, tanto las estaciones móviles como las estaciones base trabajan con la potencia de señal en la que pueden mantener una mínima calidad de servicio. Los cambios se realizan en saltos de 2 dB entre valores máximos y mínimos admitidos de potencia.

Las estaciones móviles miden la potencia de señal recibida y su calidad (a través del ratio de errores de *bits*) y transmiten esta información al controlador de la estación base que en última instancia es la que decide cuando se debe cambiar el nivel de potencia.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Este mecanismo de control de potencia debe ser usado con mucho cuidado pues existe la posibilidad de inestabilidad. Esto se produce cuando se aumentan alternativamente la potencia en dos móviles para contrarrestar aumentos alternativos de la interferencia mutua.

4.5 Señalización de la red

Asegurar la transmisión de voz o datos con una mínima calidad sobre el enlace radio es sólo una parte del conjunto de funciones de una red móvil GSM. Un móvil GSM debería poder moverse sin problemas nacional e internacionalmente, lo cual requiere funcionalidades tales como registro de usuarios, autentificación, enrutamiento de llamadas y actualización de posición. Además, el hecho de que la red se base en un sistema celular obliga a implementar el traspaso de llamadas entre células.

Todas estas funciones de señalización de red están descritas en el estándar GSM utilizando la capa MAP (*Mobile Application Part*) del sistema de señalización número 7 (SS7).

La señalización de la red en un sistema GSM está estructurada en tres subcapas:

- Gestión de recursos radio (RR)
  Controla el establecimiento, gestión, y liberación de los canales físicos así como los traspasos.
- Gestión de la movilidad (MM)
  Gestiona la actualización de la posición de las estaciones móviles así como los procedimientos para el registro y la autentificación.
- Gestión de la comunicación (CM)
  Maneja el flujo de una llamada, así como los servicios suplementarios y el servicio de mensajes cortos.

Figura 0.68: Capas de la señalización de GSM

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

4.5.1 Gestión de recursos radio

La capa de gestión de recursos radio supervisa el establecimiento del enlace entre la estación móvil y una MSC, tanto en el tramo móvil como fijo. Además también gestiona funcionalidades del interfaz radio como el control de potencia o la transmisión y recepción discontinuas.

- Traspasos (*Handover o Handoff*)

En una red celular, los enlaces móviles y fijos no siempre son los mismos durante la duración de una llamada. Los traspasos consisten en cambiar de canal durante la realización de una llamada. La gestión y las medidas necesarias para los traspasos son labores de la capa de gestión de recursos radio.

Hay cuatro tipos de traspasos en un sistema GSM, que consisten en transferir una llamada entre:

- Canales físicos de la misma celda
- Células (BTS) que estén bajo el control del mismo BSC.
- Células (BTS) que están bajo el control de diferentes BSCs pero que pertenecen a un mismo MSC.
- Células (BTS) que están en diferentes MSCs.

En los dos primeros tipos de traspaso, llamados traspasos internos, interviene solamente una BSC. Para ahorrar ancho de banda de señalización, son gestionados por la BSC sin involucrar la MSC correspondiente, excepto para notificar la finalización del traspaso.

Los dos últimos tipos de traspasos, llamados traspasos externos, son gestionados por las MSCs involucradas. Un aspecto importante de la red GSM es que, la primera MSC que gestionó la llamada, llamada MSC *ancla*, mantendrá la responsabilidad para la mayor parte de las funciones relacionadas con la llamada, con excepción de posteriores traspasos internos que se gestionaran en la MSC en la que se encuentre la estación móvil.

4.5.2 Gestión de la movilidad

La capa de gestión de la movilidad está implementada encima de la capa RR, y gestiona las funciones que surgen de la movilidad del usuario del sistema, así como aspectos de seguridad y autentificación. La actualización de posiciones hace referencia a los procedimientos que permiten al sistema conocer la posición actual de una estación base conectada a la red para poder pasarle una llamada o mensaje entrante.

- Actualización de posición

Una estación móvil conectada a la red es informada de una llamada interna mediante un aviso emitido a través del canal PCH de la celda. Una primera aproximación podría ser mandar el aviso a todas las celdas para asegurarse encontrar la estación móvil. Pero esto supondría un gasto de los recursos radio intolerable. Otra posible solución podría ser que la estación móvil mandase su nueva localización a la red cada vez que cambiase de célula. Pero esto también supondría una cantidad de mensajes de actualización de posición demasiada grande.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

La solución de compromiso que ha adoptado GSM se basa en agrupar las células en áreas de localización. Cuando un móvil cambia de área de localización debe mandar a la red un mensaje de actualización de posición y cuando la red quiere hacer llegar una llamada al móvil manda un aviso a cada una de las células del área de posición.

Los procedimientos de actualización y gestión de la posición engloban a las MSCs y a dos registros de la red, el HLR y el VLR. Cuando una estación móvil pasa a otra área de localización debe registrar su nueva posición en la red. En un caso general, un mensaje de actualización de posición se manda al nuevo MSC/VLR, quién almacena la información del área de localización y manda la información pertinente al HLR. Esta información mandada suele ser el identificador del VLR.

Otro procedimiento relacionado con la localización es la activación y desactivación de IMSI (IMSI *attack/detach*). Como vimos el IMSI es el identificador del usuario y cuando una estación base es apagada envía un mensaje de desactivación que permite a la red saber que no tiene que ir a buscar al móvil en caso de llamada. Evidentemente, cuando una estación móvil se enciende, se registra en la red y manda un mensaje de activación.

- Autentificación y cifrado

Puesto que el interfaz radio es accesible para cualquiera, la autentificación de usuarios debe probar que son quienes dicen ser, siendo esta una funcionalidad muy importante dentro del sistema GSM.

La autenticación involucra dos componentes de la red, la estación móvil y el centro de autenticación (AuC). Cada usuario de la red tiene una clave secreta, la cual está almacenada tanto en la SIM como en el AuC. Durante el proceso de autentificación, el AuC genera un número aleatorio que manda a la estación móvil. Este número se procesa junto con la clave secreta mediante el algoritmo A3 para generar una respuesta que es mandada al AuC. Si el número mandado por el móvil es el mismo que el calculado internamente en el AuC el usuario es autentificado.

El mismo número aleatorio y la clave secreta del usuario son también usados para calcular una clave mediante el algoritmo A8. Esta clave, junto con el número de la trama TDMA, usa el algoritmo A5 para generar una secuencia de 114 bits que a su vez es combinada con los datos de la trama mediante XOR.

No sólo sobre el usuario y las comunicaciones se realizan procedimientos de seguridad, también el terminal se ve involucrado en estas consideraciones. Como vimos anteriormente, cada terminal GSM posee un identificador único universal llamado IMEI. La lista de IMEIs se almacena en el registro EIR (*Equipment Identity Register*).

4.5.3 Gestión de las comunicaciones

La capa de gestión de las comunicaciones es responsable del control de las llamadas, la gestión de servicios suplementarios y la gestión del servicio de mensajes cortos. Cada una de estas funcionalidades puede considerarse como subcapas del nivel.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Como principal aspecto a destacar de este complejo nivel es el uso del *Mobile Subscriber ISDN* (MSISDN) como número de marcación para acceder a un teléfono.

No vamos a entrar en detalle de todos los procedimientos de esta capa pues su comprensión escapa a los objetivos de este análisis. En cualquier caso cabe resaltar que todos los procedimientos usados en esta gestión de las comunicaciones son muy parecidos a los correspondientes en RDSI.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

GPRS

4.6 Introducción a la conmutación de paquetes

Las redes móviles GSM ofrecen servicios de transmisión de datos desde la fase inicial de sus especificaciones (fase I). Sin embargo, se trata de servicios con modalidad de transferencia por conmutación de circuitos, es decir, donde la red, una vez establecida la conexión lógica entre dos usuarios, dedica todos los recursos hasta que no es solicitada expresamente su desconexión, independientemente de si los usuarios intercambian o no información y de la cantidad de la misma.

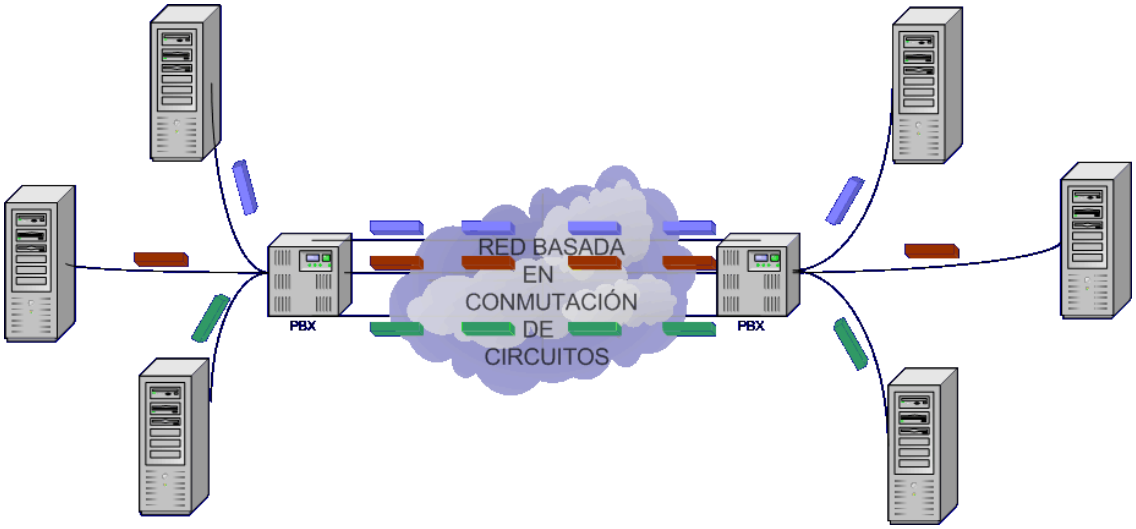


Figura 0.69: Transmisión de información mediante conmutación de circuitos

Este funcionamiento de las redes supone un desperdicio de recursos en la transmisión de datos y una factura más cara para el usuario puesto que paga por el tiempo de conexión, no por los datos realmente transmitidos y recibidos.

De hecho, la transmisión de datos con conmutación de circuitos sólo se puede considerar óptimas las transmisiones en las que se intercambien una cantidad significativa de datos como en las transferencias de grandes ficheros. El problema radica en que son precisamente muy ineficientes en las comunicaciones en las que se solicitan datos de forma interactiva y poco constante, como por ejemplo en la navegación a través de Internet. Este tipo de comunicaciones, que son las más habituales en nuestros días, suponen con conmutación de paquetes tener el canal reservado sin usarlo en gran parte del tiempo.

Es decir, de lo que se trata es de proveer a GSM de un mecanismo de conmutación de paquetes con el que los datos de los usuarios, junto con una indicación del remitente y destinatario, puedan ser transportados por la propia red sin necesidad de una estrecha asociación con un circuito físico.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

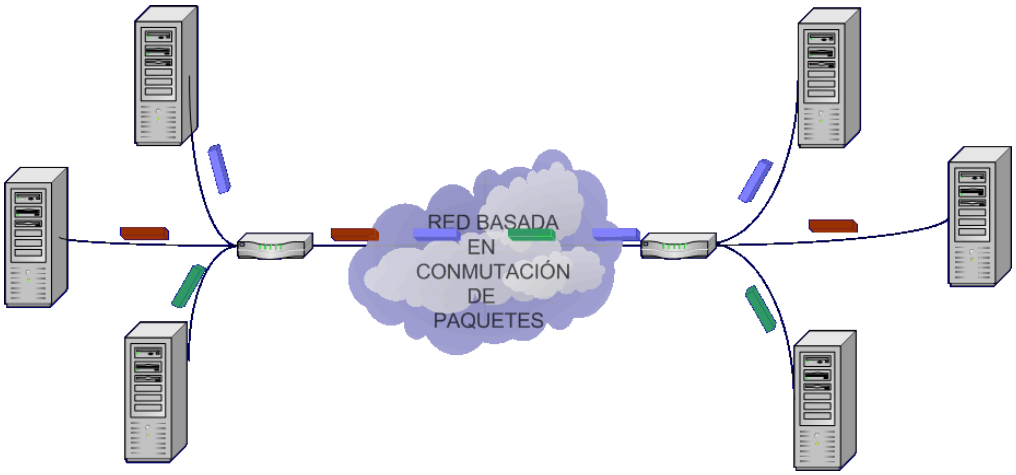


Figura 0.70: Transmisión de información mediante conmutación de paquetes

GPRS (*General Packet Radio Service*) viene a incorporar en GSM la transmisión de datos a mayor velocidad además de mediante conmutación de paquetes. Es un paso intermedio en la evolución de las redes GSM hacía las redes de tercera generación por lo que se le suele denominar como una tecnología 2,5G. Su implantación se hace sobre la red existente GSM por lo que su coste de despliegue es relativamente reducido. Esto, junto con la demora y el coste que tiene el despliegue de las redes 3G, ha hecho que su éxito comercial haya sido alto, siendo muchas las operadoras que los han implementado como puente hacia siguientes tecnologías.

4.7 Ventajas de GPRS

4.7.1 Velocidad

GPRS ofrece una gran gama de velocidades para la transmisión de datos. Como veremos, combinando diferentes configuraciones de red y terminales podremos obtener velocidades teóricas de hasta 172 Kbps. De todas formas las velocidades de las redes actuales suelen estar entre 20kbps - 56 Kbps. Aún así esto supone una gran mejora respecto a la velocidad de transmisión de datos que soporta GSM, que es 9.600 bps.

4.7.2 Conmutación basada en paquetes

Como hemos visto en el apartado anterior, la transmisión de datos con conmutación basada en paquetes aprovecha de mejor forma la capacidad de la red que la conmutación basada en circuitos. Además con este esquema de conmutación se obtiene una ventaja añadida, que no es otra que se facturará a los usuarios por tráfico real y no por tiempo de conexión.

4.7.3 Always on

Otra ventaja de GPRS, que además es consecuencia de la conmutación de paquetes, es que el terminal puede estar permanentemente conectado a la red. Esto significa que cuando el usuario quiera acceder a un servicio no tendrá que esperar a que se establezca la conexión reduciéndose considerablemente el tiempo de acceso. También permitirá mecanismos más sencillos de recepción de servicios *push*, es decir de servicios iniciados por las aplicaciones y no por el usuario.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

4.7.4 Bajos costes de despliegue

Los bajos costes que implica el despliegue de GPRS suponen una gran ventaja para las operadoras y ha permitido que su implementación haya sido en cierta manera masiva. Estos costes se deben a que GPRS se monta encima de la red GSM utilizando todos sus componentes. Además no tiene porque haber caídas o periodos de inactividad de la red debidas al despliegue de GPRS.

4.7.5 Nuevas y mejores aplicaciones

Gracias a la mayor velocidad de transferencia de datos, a su conmutación basada en paquetes y su característica de conexión permanente, GPRS posibilitará una serie de aplicaciones y servicios que hasta ahora no habían tenido cabida con GSM. Los usuarios podrán navegar por Internet con mayor velocidad, podrán jugar a juegos en red sin preocuparse que la conexión permanece abierta y facturando, y podrán recibir contenidos multimedia entre otras aplicaciones posibles.

4.8 Funcionamiento general de GPRS

Ya hemos visto que GPRS tiene como principales características que permite mayor velocidad de datos, utiliza conmutación de paquetes y permite conexiones permanentes. Vamos a analizar como funciona GPRS para conseguir estas ventajas mediante la red GSM.

En las redes GSM los recursos se gestionan según la modalidad *resource reservation*, es decir, se emplean en exclusiva desde el mismo momento en el que la petición de servicio se ha llevado a cabo. En GPRS, sin embargo, se adopta la técnica de *context reservation*, se decir, se tiende a reservar las informaciones necesarias para soportar, ya sea las peticiones de servicio de forma activa o las que se encuentran momentáneamente en espera. Por tanto, los recursos radio se ocupan sólo cuando hay necesidad de enviar o recibir datos y no en otros momentos. De esta forma los recursos de radio (canales lógicos de transmisión de datos) de una célula se comparten mediante aloha ranurado entre todas las estaciones móviles, aumentando notablemente la eficacia del sistema.

Por otro lado, el aumento de velocidad de las redes GPRS se consigue a base de dos factores. Por una parte se utilizan nuevos esquemas de codificación de los datos que permiten mayores velocidades a costa de menores cantidades de redundancia. Evidentemente para utilizar los esquemas de codificación más rápidos se debe tener una relación Señal/Ruido (S/N) muy elevada, es decir que la tasa de errores que se produce en la transmisión radio sea muy baja.

Puesto que los bits totales transmitidos por una trama TDMA son constantes, si enviamos menos bits de redundancia sin información podremos aumentar la cantidad de datos transmitida en cada trama. Como las tramas TDMA se transmiten de forma constante con el tiempo, utilizar esquemas de codificación más ligeros supone una forma sencilla de aumentar la velocidad a costa de reducir la robustez de las comunicaciones respecto a los errores.

En la siguiente tabla se recogen los cuatro esquemas de codificación especificados en GPRS con los bits de información transmitidos respecto a los 456 bits transmitidos en una trama. Además se incluye la velocidad que tendría una transmisión de datos a nivel de protocolos de enlace radio si el terminal usase todo el canal lógico. Esta velocidad teórica luego se verá reducida primero por el *overhead* de

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

los protocolos superiores de la pila de comunicaciones y segundo por la compartición que se realiza de los canales físicos en GPRS.

Esquema de codificación de canal	Bits de datos de cada radio-bloque de longitud fija 456 bits	Kbps por cada *Time Slot* en la capa radio
CS-1	181	9,05
CS-2	268	13,4
CS-3	312	15,6
CS-4	428	21,4

Tabla 0.2: Esquemas de codificación de canal en GPRS

En la práctica casi todas las operadoras han optado por utilizar los dos primeros esquemas de codificación, por dos razones prácticas. Primero por que en el CS-3 y en el CS-4 suponen una reducción alta de la redundancia poniendo en serias dificultades comunicaciones con tasas de errores altas.

La segunda razón surge a partir de la velocidad que proveen. Con estos esquemas de codificación superamos un límite en la red GSM. Este límite no es otro que el tamaño de los circuitos en el interfaz A-bis entre las BTSs y el BSC. Este ancho de banda (16 kbps) es menor que el ancho que el interfaz radio tendría con estos dos esquemas de codificación. Esto supone que para utilizar el CS-3 y el CS-4 se deben utilizar dos circuitos A-bis dificultando y encareciendo mucho la implementación. Las operadoras que querían obtener velocidades superiores a 100 Kbps han optado por otras tecnologías 2,5G como EDGE.

El segundo factor que permite conseguir mayor velocidad es la utilización de varios intervalos de tiempo (*Time Slots*) de forma combinada. Sencillamente la estación móvil puede utilizar tantos canales físicos simultáneos como tenga la célula o su tecnología lo permita. De esta forma las velocidades que hemos visto antes se multiplican por números enteros según el número de canales simultáneos que se utilicen.

Esquema de codificación de canal	Kbps por cada *Time Slot* en la capa radio	Velocidad máxima por portadora (usando 8 *Time Slots*)
CS-1	9,05	72,4
CS-2	13,4	107,2
CS-3	15,6	124,8
CS-4	21,4	171,2

Tabla 0.3: Esquemas de codificación en GPRS con velocidad máximas teóricas

4.9 Arquitectura

La tecnología GPRS se implementa sobre las actuales redes GSM. No hay por lo tanto que identificar la arquitectura de una red GPRS de forma separada y aislada de una GSM. Eso sí, puesto que se han

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

añadido ciertas interfaces para soportar la conmutación de paquetes, se debe dar una total compatibilidad entre las dos formas de conmutar, paquetes y circuitos.

Desde el punto de vista físico los recursos pueden ser reutilizados y existen algunos comunes en la señalización, así, en la misma portadora pueden coexistir a la vez tanto los intervalos de tiempo reservados a conmutación de circuitos como los intervalos de tiempo reservados al uso de GPRS.

Dentro de los canales físicos (intervalos de tiempo) disponibles para GPRS en las distintas portadoras de una celda se pueden encontrar tres tipos:

- Canales dedicados

  Su uso siempre es para canales lógicos de transmisión de datos mediante conmutación de paquetes (GPRS). El conjunto de estos canales en la célula establecen la calidad de servicio mínima.
- Canales conmutables

  Estos canales están dedicados a conmutación de paquetes (GPRS) por defecto, pero en caso de necesidad pueden pasar a transmitir tráfico GSM mediante conmutación de circuitos.
- Canales adicionales

  Estos canales están dedicados por defecto a conmutación de circuitos (GSM) aunque se pueden utilizar para GPRS.

Por lo tanto, la optimización en el empleo de los recursos se obtiene a través del reparto dinámico de los canales reservados a la conmutación de circuitos y de aquellos reservados a GPRS. En caso de necesitar un nuevo canal para una llamada de voz, hay tiempo suficiente para liberar parte de los recursos usados por GPRS, de tal manera que la llamada, con mayor prioridad, pueda ser atendida. Los operadores tienen en este hecho una ventaja muy importante, ya que pueden ajustar los recursos en función de la demanda y atender las llamadas de voz, ralentizando un poco las comunicaciones ya establecidas de datos sin llegar a cortarlas, lo que no es muy molesto para los usuarios, ya que no se llega a interrumpir el servicio.

Para conseguir que una red GSM permita todas las funcionalidades que incluye GPRS hay que realizar varias acciones sobre la misma. Por un lado hay que actualizar el software de cada uno de los elementos de la red para que soporten conmutación de paquetes. Esta actualización de software suele realizarse de forma remota por lo que su coste es bajo.

Además, hay que incluir nuevos elementos de hardware en la red. Estos elementos son los siguientes:

4.9.1 Unidad de Control de Paquetes

La Unidad de Control de Paquetes (PCU, *Packet Control Unit*), son elementos de red que se sitúan junto a las BSCs y se encargan de la asignación y gestión de los canales lógicos propios de GPRS.

4.9.2 Nodo servidor de GPRS (SGSN, Server GPRS Support Node)

Se encarga de las siguientes funciones:

- Cifrado, autenticación y comprobación de IMEI
- Gestión de movilidad
- Gestión del enlace lógico hacia el MS

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Datos de facturación
- Responsable de la distribución de paquetes a las estaciones móviles pertenecientes a su área de servicio

Este nodo estará conectado con el HLR, el MSC, BSC y SMSC.

4.9.3 Nodo pasarela de GPRS (GGSN, Gateway GPRS Support Node)

Hace de interfaz lógica entre la red GPRS y las redes de paquetes externas. Se encarga entre otras tareas de transmitir los paquetes a las redes externas correspondientes. Para saber a qué interfaz externa de cuál GGSN debe ir un paquete se usa un nombre lógico llamado APN (*Access Point Name*).

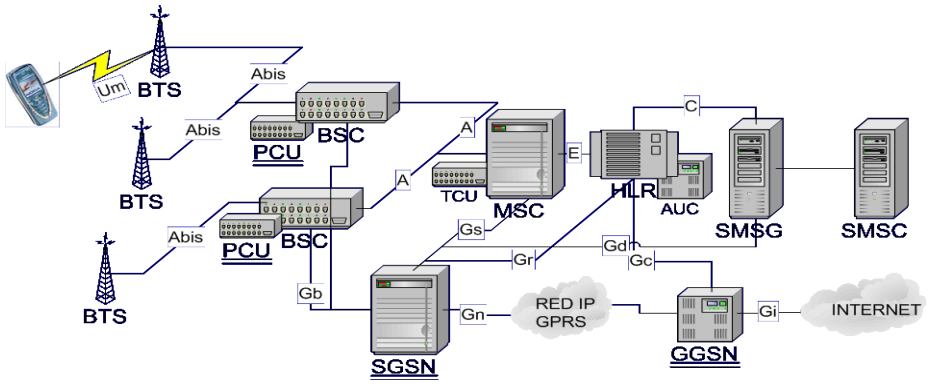


Figura 0.71: Arquitectura de la red GSM/GPRS

4.10 Terminales GPRS

Las estaciones móviles GPRS se basan en la misma arquitectura que las GSM. Constan de un terminal y una tarjeta inteligente SIM. Para GPRS sólo hace falta poseer un terminal que soporte el nuevo interfaz radio puesto que sigue siendo válida la tarjeta inteligente usada en GSM.

En función de la dualidad GSM/GPRS se pueden definir tres categorías de terminales GPRS:

4.10.1 Clase A

Son teléfonos totalmente duales GSM/GPRS con capacidad de manejar ambos tráficos simultáneamente. Esto significa que el terminal puede recibir y/o transmitir a la vez paquetes de GPRS y voz a través de circuitos de GSM.

4.10.2 Clase B

Son teléfonos duales GSM/GPRS pero que no pueden manejar simultáneamente ambos tráficos teniendo que conmutar automáticamente entre ambos modos de funcionamiento.

4.10.3 Clase C

En este tipo de terminales la conmutación entre el modo de funcionamiento GSM y GPRS se hace manualmente por parte del usuario.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

La implementación de terminales clase A es todavía muy compleja, siendo la mayoría de las terminales comercializados de clase B.

Figura 0.72: Terminales GPRS

Además de la clasificación según la dualidad GSM/GPRS, los terminales GPRS se suelen clasificar mediante el número de canales radio que pueden manejar tanto en el canal ascendente como en el descendente. En la siguiente tabla se especifica los diferentes tipos de terminales que puede haber. Evidentemente cuantos más canales puede manejar el terminal más compleja es su implementación y más batería gasta por lo que las configuraciones más avanzadas no se suelen llevar nunca a la práctica.

Clase de terminal *multislot*	**Máximo número de intervalos usados**		
	Recepción	Transmisión	Simultáneos
1	1	1	2
2	2	1	3
3	2	2	3
4	3	1	4
5	2	2	4
6	3	2	4
7	3	3	4
8	4	1	5
9	3	2	5
10	4	2	5
11	4	3	5
12	4	4	5

Tabla 0.4: Clases de terminales multislot

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

5 UMTS

5.1 Historia y evolución de UMTS

Es frecuente que se asuma que todos los sistemas de comunicaciones móviles celulares de tercera generación son UMTS (*Universal Mobile Telecommunications System*). Nada más lejos de la realidad. UMTS es una de las familias de este tipo de sistemas.

Nace en el contexto de programa europeo en el año 1988. Para esa época ya se empiezan a observar parte de los problemas que tenía la segunda generación y se plantea una nueva generación que suponga la solución para estos inconvenientes y que incorpore las mejoras tecnológicas que se van produciendo durante los años.

El planteamiento final que se obtiene de este proceso es un sistema compuesto por diversas familias que responden a intereses locales o regionales. El nombre de este conjunto de soluciones es IMT-2000. Durante este tema y el siguiente veremos las principales características de la mayoría.

ITU IMT-2000 interfaces		Standards organisations
IMT-DS	UMTS component paired WCDMA frequency bands (FDD mode)	3GPP
IMT-TC	Components of unpaired frequency bands: UMTS (TDD mode) TD-CDMA and radio interface proposed by China TD-SCDMA	3GPP CCSA (TD-SCDMA)
IMT-MC	CDMA2000: CDMA network evolution	3GPP2
IMT-SC	Evolution of IS-136 (TDMA) networks primarily deployed in US – UWC 136	3GPP
IMT-FT	DECT	ETSI

Figura 0.73: Familias IMT-2000 Fuente: UMTS-Forum

La estandarización de UMTS, se está diseñando principalmente en Europa por el 3GPP (*Third Generation Partnership Project*) aunque este organismo no tiene potestad para realizar normas, por lo que elabora los documentos técnicos que luego pasarán a ser normas gracias a los correspondientes organismos de estandarización.

Este proceso de normalización se ha ido realizando en fases, publicándose versiones del estándar cada pocos meses.

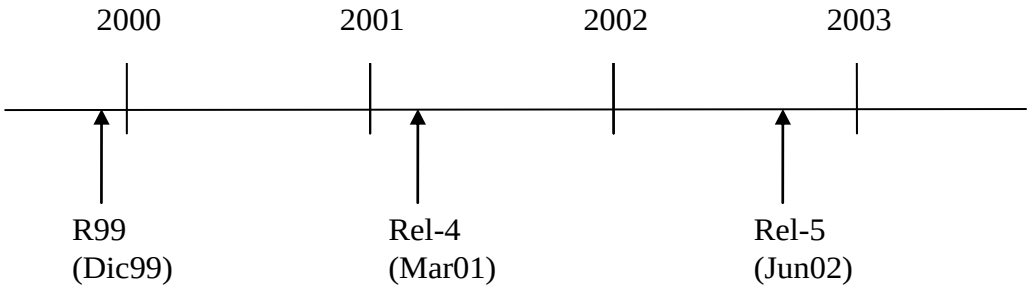


Figura 0.74: Calendario de las versiones de UMTS

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

UMTS es un conjunto de estándares que pretenden dar una solución global y más avanzada a las redes y servicios de telefonía móvil que la que había con la segunda generación. Es por esta razón que la complejidad y amplitud de este sistema sobrepasa en mucho a GSM con lo que en este curso sólo realizaremos una pequeña introducción a todos los componentes de UMTS haciendo hincapié en las principales diferencias con los anteriores sistemas. Además en este capítulo sólo trataremos la red UMTS dejando para más adelante los servicios 3G.

5.2 Servicios de UMTS

UMTS pretende soportar y mejorar los servicios que provee GSM. De forma resumida se pueden concretar en los siguientes:

- Servicios en modo paquete y en modo circuito
- Conexión de alta velocidad para transmisión de datos

Entorno	Velocidad de los móviles	Tasa de bit objetivo
Exterior rural (>1 Km)	Alta (<500 Km/h)	144 kpbs
Exterior urbano (200-400 m)	Media (<120 Km/h)	384 kpbs
Interior/Exterior de corto alcance (100m)	Baja (<10 Km/h)	2048 kpbs

Tabla 0.5: Servicios de transmisión de datos de UMTS

- Intercomunicación con otras redes.
- Soporte de servicios simétricos y asimétricos.
- Itinerancia (*roaming*) global. Movilidad de terminales, usuarios y *servicios*.
- Calidad de voz comparable a la de la telefonía fija.
- Integración de redes: fijas y móviles, voz y datos.

5.3 Arquitectura de UMTS

Tradicionalmente, las redes de telecomunicaciones se han diseñado con la intención de transmitir un solo tipo de contenidos: voz, datos, TV, etc., y por lo tanto su estructura venía determinada por este contenido. Estas redes eran totalmente independientes unas de otras y poseían sus propias redes de acceso, transporte y conmutación.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

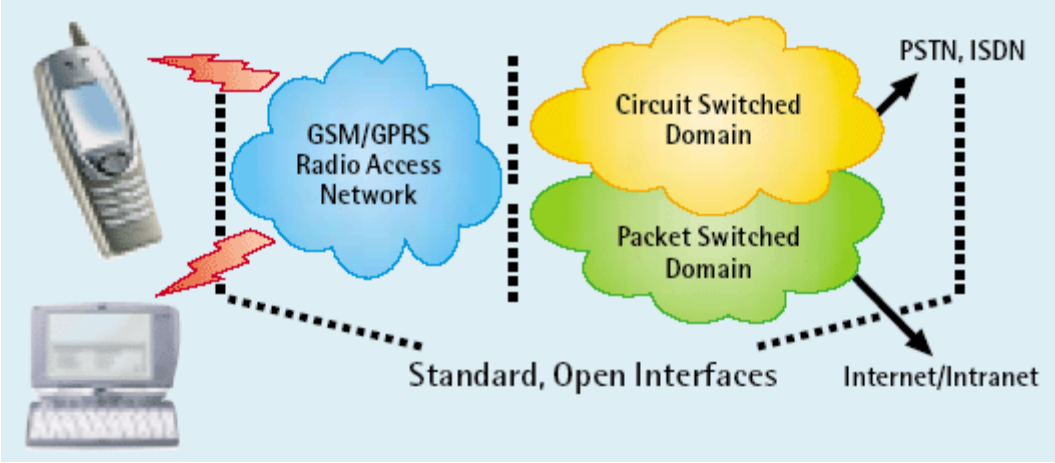


Figura 0.75: Estructura clásica de redes de acceso y troncales Fuente: UMTS-Forum

La tendencia actual es diseñar redes que reutilicen o tengan un diseño similar que otras ya construidas y además se procura que sean multicontenido y multiservicio, es decir que soporten los diferentes contenidos con las mismas infraestructuras de acceso y transporte. Todo esto se traduce en una importante reducción de costes, mayor eficacia en su funcionamiento y la facilidad de una gestión unificada.

Cuando se desarrollaron los estándares 2G se aplicaron de forma general a toda la red por lo que una red GSM es totalmente incompatible con otra TDMA, también digital, tanto en infraestructuras como en terminales. En el caso de la tercera generación el planteamiento es diferente. Existe por un lado un proceso de normalización para la red de acceso (*access network*) y otro diferente para la red central o troncal (*core network*). De esta forma se está desarrollando la red troncal, conmutación y transmisión, compatible con otras redes digitales actuales. No así la red de acceso radio, que es totalmente nueva. Esto significa que trataremos en mayor profundidad esta última así como su funcionamiento puesto que son conceptos nuevos y no vistos todavía.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

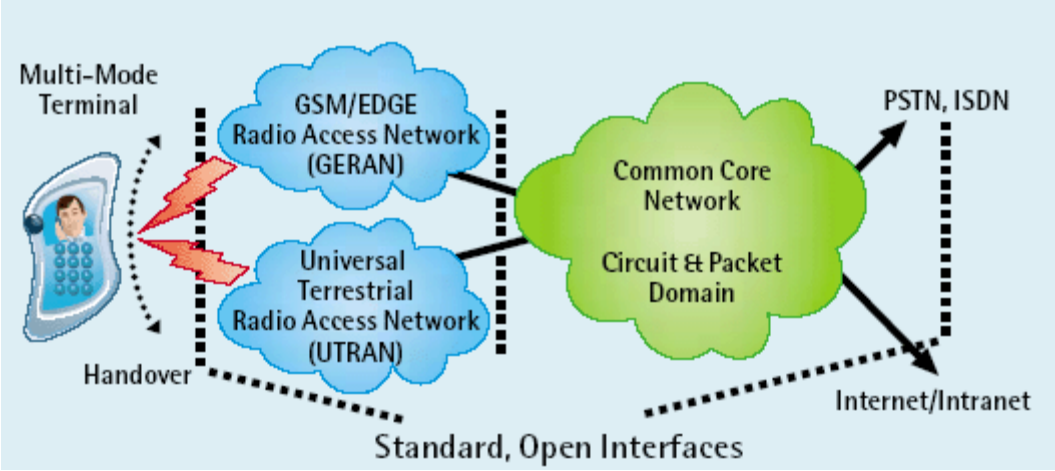


Figura 0.76: Estructura de redes de acceso con redes troncales comunes Fuente: UMTS-Forum

En la arquitectura UMTS se pueden distinguir claramente tres bloques, por un lado aparece la estación móvil o equipo de usuario, la red de acceso radio y la red troncal.

5.3.1 Estaciones Móviles

Las estaciones móviles o equipos de usuario (MS, *Mobile Station*), al igual que en GSM, se componen de un dispositivo o terminal móvil y un módulo de identificación de usuario. Este módulo de identificación es también una tarjeta inteligente que en el caso de UMTS se le denomina USIM.

Respecto a los terminales móviles, en la tercera generación de redes móviles la complejidad de estos dispositivos es mucho mayor que en épocas pasadas. Para manejar los complejos algoritmos de los protocolos, así como para soportar contenidos multimedia debe poseer una capacidad de proceso considerablemente mayor. Además tienen todo tipo de conectores (USB, IRDA, BlueTooth) para trabajar con equipos externos, soportan gran cantidad de periféricos y accesorios externos y embebidos como tarjetas de memoria, cámaras, etc.

También cambia el software que incluyen estos terminales 3G. No sólo incluyen en muchos casos verdaderos sistemas operativos sino que además soportan los nuevos servicios 3G. En este sentido se provee soporte para aplicaciones Java o decodificadores MPEG4 entre otras cosas.

Finalmente para aumentar la complejidad de los terminales y, puesto que la cobertura inicial y la capacidad de la red UMTS son limitadas, debe implementarse dispositivos duales con las redes 2G y 2,5G. Por lo tanto, lo normal en el inicio será encontrarnos con terminales híbridos GSM, GPRS y UMTS.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Figura 0.77: Terminales UTMS

No sólo los terminales han evolucionado, también la tarjeta inteligente que sirve para identificar al usuario lo ha hecho. Aparece USAT (*UMTS SIM Application Toolkit*) como evolución al *toolkit* que incluían las SIM de GSM. Actualmente estas tarjetas (USIM en UMTS) permiten al operador realizar actualizaciones de las aplicaciones incluidas en la tarjeta mediante mecanismos de transporte variados como SMS o GPRS. Además estas aplicaciones pueden realizar todo tipo de acciones en el teléfono por lo que el usuario tiene un menú personalizado en el que las entradas pueden realizar cosas como:

- Realizar llamadas
- Enviar mensajes cortos
- Conectarse a páginas WAP
- Etc.

5.3.2 Sistema de Red Radio

La red de acceso consiste en los elementos que controlan el acceso a la red y proporcionan a los usuarios un mecanismo para acceder a la red local.

En el caso de UMTS esta red de acceso se realiza mediante enlaces radio y se denomina UTRAN (*UMTS Terrestrial Radio Access Network*). En este parte es donde se encuentran las principales diferencias entre GSM y UMTS sustituyendo las BTSs y las BSCs por Nodos-B y RNC (*Radio Network Controller*) respectivamente. Además los interfaces que unen los diferentes elementos también cambian tal y como podemos ver en la siguiente figura.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

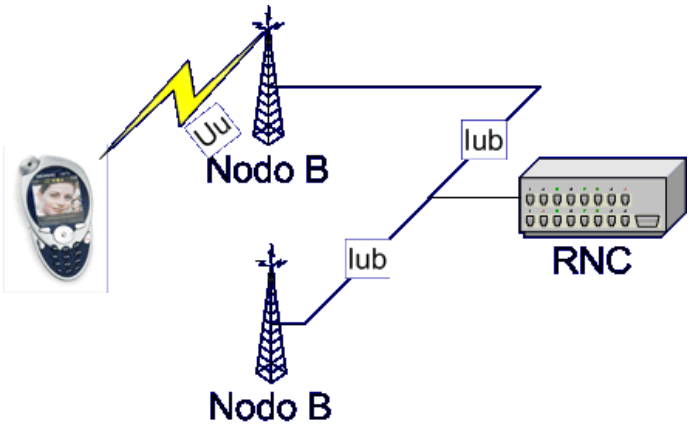


Figura 0.78: Sistema de red radio UMTS

Los principales bloques que forman el sistema radio se denominan RNS (*Radio Network System*) y se componen de un RNC y varios Nodos-B. Las funciones que realizan estos elementos son parecidas a las que se venían realizando en GSM en los elementos equivalentes, es decir, gestionar los recursos radio, decidir sobre los traspasos, etc.

Los Nodos-B, cada uno de los cuales puede controlar varias células, son los encargados de suministrar los recursos radio a las estaciones móviles. Existe una amplia variedad de nodos, tanto para interior como para exterior, micros, minis, y macros, escalables según la necesidad de capacidad.

Respecto al interfaz radio en sí, la complejidad comparado con el de GSM aumenta bastante. Se basa en tres capas que dan diferentes servicios y tiene dos modos de funcionamiento UTRA-FDD y UTRA-TDD ambos basados en WCDMA. Las principales características de estos modos así como otras características de este interfaz radio serán explicados en el siguiente apartado.

5.3.3 Red Troncal

Como hemos dicho la red troncal (*CN, Core Network*) es la parte que menos cambios ha sufrido respecto a las tecnologías 2G. En realidad esto es una verdad a medias, puesto que la primera versión del estándar (*release 99*) que se publicó efectivamente tenía una red troncal muy parecida a la GSM/GPRS. En las sucesivas versiones de UMTS se han ido definiendo redes troncales más cercanas a las redes de paquetes actuales, basándose prácticamente todas las comunicaciones en el protocolo IP.

En la siguiente figura podemos ver la arquitectura completa de la *release* 99 evolucionada a partir de una red GSM. Esta arquitectura es una aproximación bastante cercana a las redes que se han montado para lanzar UTMS al mercado.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

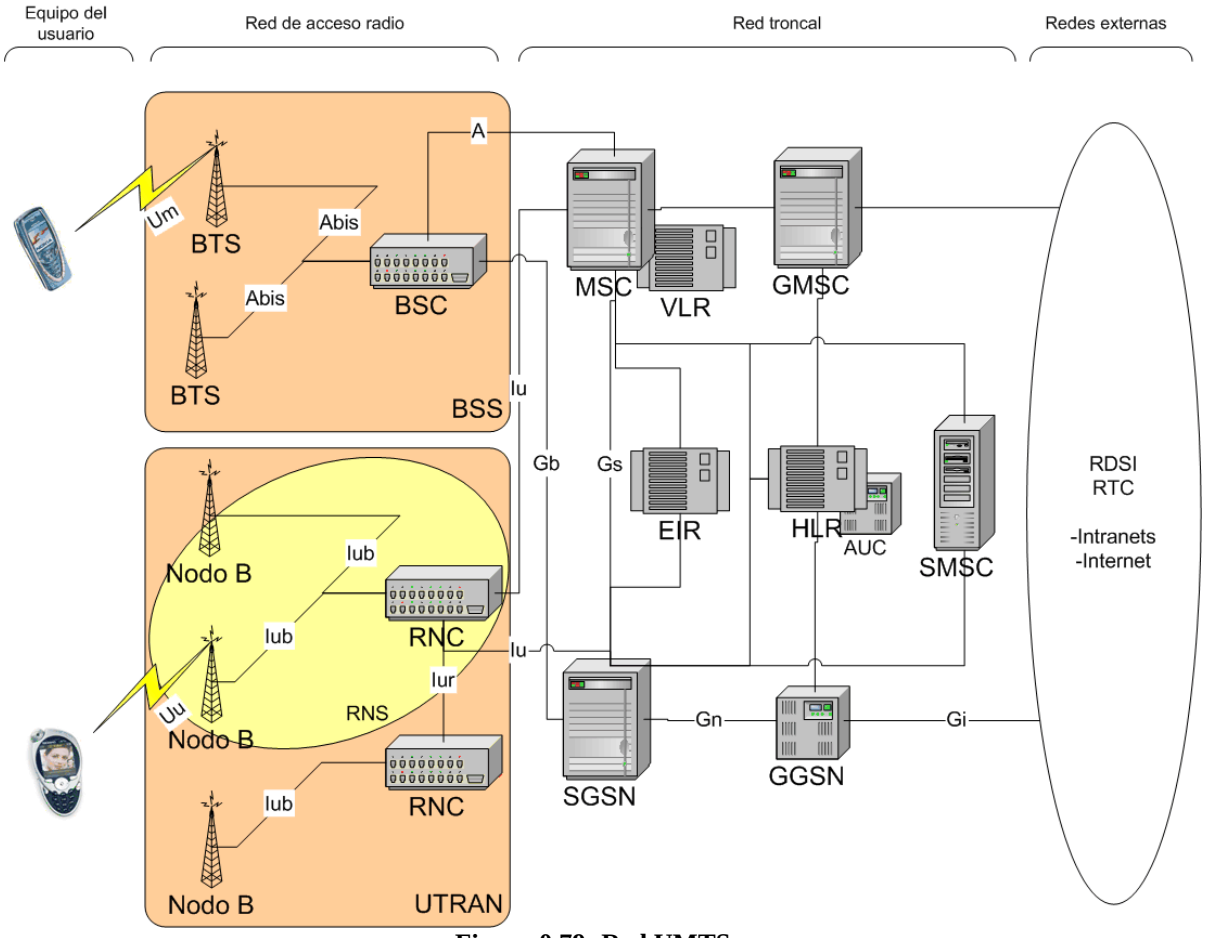


Figura 0.79: Red UMTS

5.4 Interfaz radio

La interfaz radio es la capa en la que UMTS supone mayor ruptura con los sistemas anteriores. Como vamos a ver a continuación, es la tecnología de acceso al medio que utiliza, CDMA, la que aporta las mayores diferencias y ventajas.

La interfaz radio de UTMS se diseñó pensando en una red que optimizase el uso del espectro disponible además de que permitiese una cierta escalabilidad en su despliegue. Según las medidas del *UMTS Forum* (www.umts-forum.org), si comparamos la capacidad que puede dar un Nodo-B de UMTS respecto a una estación base de GSM encontramos que es 10 veces mayor a un coste ligeramente superior.

En este apartado vamos a tratar de introducir los principales conceptos de la interfaz radio de UMTS, detallando las ventajas e inconvenientes de las principales novedades.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

5.4.1 Tecnología WCDMA

En cualquier tecnología inalámbrica se comparte el medio radio. En general, hay tres soluciones para que los usuarios puedan acceder de forma ordenada a ese medio. Ya vimos que estas soluciones eran TDMA, FDMA y CDMA. Todas las tecnologías de tercera generación usan esta última puesto que se ha destacado como la más eficiente en el uso del espectro (además de otras ventajas que luego veremos).

Cuando un usuario accede al medio radio usando CDMA, usa la misma frecuencia que el resto de usuarios al mismo tiempo. La forma de distinguir entre las transmisiones de distintos usuarios es mediante un código binario que se asigna unívocamente. El proceso teórico es el siguiente. El terminal del usuario multiplica la señal que quiere enviar por su código binario y el receptor calcula la correlación de la señal que le llega con ese código para obtener la señal transmitida. Si todos los códigos son ortogonales entre sí, el resto de señales de los demás usuarios se anularían en ese proceso.

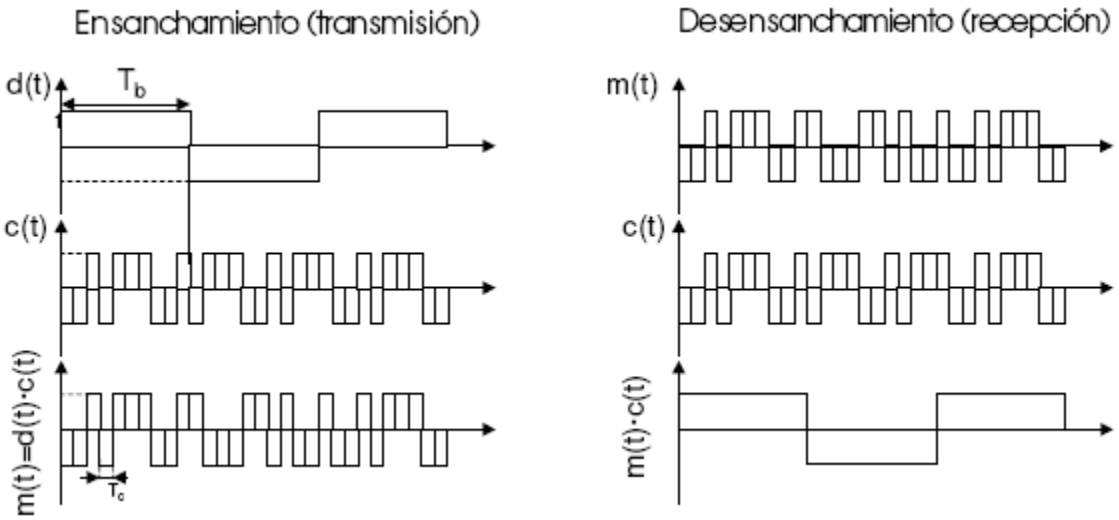


Figura 0.80: Multiplicación de señales en CDMA

Esta técnica se denomina espectro ensanchado porque al multiplicar la señal por un código de mayor velocidad binaria la señal resultante tiene un espectro de banda mucho más ancha que la primera. Cada *bit* del código binario se denomina *chip* y su duración tiene una relación inversa con el ancho del canal y con la tasa de transmisión de datos máxima.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

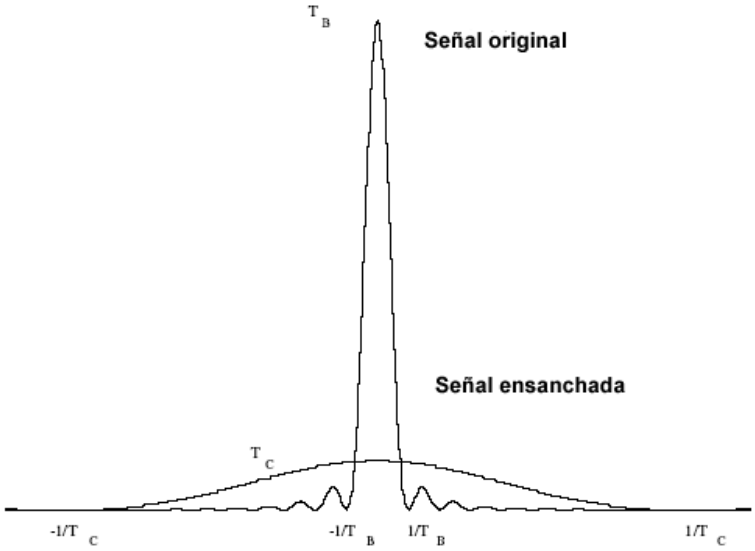


Figura 0.81: Espectro de la señal ensanchada

En el ensanchamiento se pasa de una señal de velocidad binaria Tb (bits/s) a otra señal de velocidad binaria Tc (chips/s) que es igual a la velocidad del código. A la relación Tb/Tc se la conoce como factor de ensanchado o ganancia de procesado.

En estos sistemas, puesto que el multitrayecto provoca que los códigos en recepción no sean totalmente ortogonales, las principales interferencias son las provenientes de las transmisiones de otros usuarios en la misma frecuencia pero con distintos códigos. Esto provoca que el control de potencia en CDMA sea especialmente importante para que la señal de todos los usuarios llegue con parecida potencia a la estación base.

Las técnicas de ensanchado del espectro tienen una serie de ventajas importantes:

- Estas técnicas dan una protección natural contra las interferencias de banda estrecha y de banda ancha. En el caso de la banda estrecha, estas señales se eliminan en el desensanchamiento puesto que al ser de banda estrecha lo que sucede es que se ensanchan.

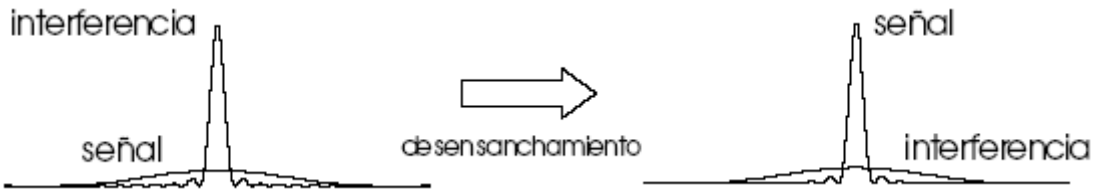


Figura 0.82: Protección contra interferencias de banda estrecha

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Si las interferencias son señales de banda ancha, cuando se produce el desensanchamiento la interferencia debería seguir ensanchada. Este rechazo depende mucho de la correlación cruzada con el código utilizado.

Figura 0.83: Protección contra interferencias de banda ancha

- Al ocupar más espectro, la señal no puede atenuarse totalmente en el canal dispersivo. Antes con señales de banda estrecha, si el canal sufría un desvanecimiento por multitrayecto en esas frecuencias perdíamos prácticamente la señal. Ahora tenemos la información distribuida entre muchas frecuencias con lo que la pérdidas son mucho menores, de alguna forma estamos utilizando diversidad en el espectro.
- Además, puesto que podemos utilizar las mismas frecuencias en celdas contiguas, un usuario puede estar conectado a la vez a dos o más estaciones base permitiendo traspasos suaves (*soft handover*) que hacen imperceptibles al usuario los cambios de estación.
- Otra ventaja del CDMA, a la hora de transmitir voz, es que se consigue aprovechar de forma natural el carácter discontinuo de la información hablada. Puesto que las interferencias las provocan los usuarios transmitiendo datos, cuando un usuario no habla y no se transmiten datos la capacidad del sistema se incrementa inmediatamente. Esto permite mejoras del orden del 50%.
- Además, en el futuro se podrán utilizar receptores basados en cancelación de interferencias o de detección conjunta. Estos receptores van suprimiendo, progresivamente, a todos los usuarios no deseados obteniendo una señal mucho más limpia para decodificar.

En UMTS se utiliza una versión del CDMA llamada *Wide CDMA* (WCDMA) con capacidad 8 veces mayor, y que emplea canales radio con una anchura de banda de 5 MHz frente a los 1, 25 MHz de CDMA.

En la práctica, no se pueden usar códigos ortogonales entre sí en toda la red puesto que el número de estos es limitado. Lo que se realiza es un ensanchamiento en dos fases.

En la primera fase, llamada de canalización, la señal de velocidad Tb se multiplica por un código de velocidad Tc (3,84 Mchip/s en UMTS). Estos primeros códigos son códigos cortos y totalmente ortogonales entre sí, pero por su escaso número sólo se usan para identificar a los usuarios dentro de una misma célula en el enlace descendente. Su longitud indica la ganancia de procesado de la transmisión.

Posteriormente se procede a la aleatorización, multiplicando la señal resultante por un código pseudoaleatorio de la misma velocidad por lo que no se produce mayor ensanchamiento. Estos códigos

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

son bastante complejos, presentan diferentes longitudes y aunque no son totalmente ortogonales su número es muy alto permitiendo identificar usuarios en distintas células.

5.4.2 Modos FDD y TDD

Como hemos visto, para UMTS se escogió CDMA para realizar el acceso al medio compartido para diferentes usuarios. Pero para diferenciar entre el enlace ascendente y el descendente se optó por dos soluciones diferentes, FDMA y TDMA. De esta forma hay dos modos de funcionamiento o dos interfaces radio distintas:

- Modo FDD (*Frecuency Division Duplex*)

  El acceso múltiple al medio se realiza por división en código (para diferentes usuarios) y en frecuencia (para distinguir entre el enlace ascendente y el descendente). Por lo tanto se utilizan dos portadoras como GSM con canales de 5 MHz de ancho de banda cada uno. Este modo de funcionamiento es usado tanto por UMTS como por FOMA, el estándar 3G japonés.

- Modo TDD (*Time Division Duplex*)

  En este modo el acceso múltiple se realiza por división en código (para diferentes usuarios) y en el tiempo (para distinguir entre el enlace ascendente y el descendente). Existe una única portadora y un único canal de 5 MHz e intervalos temporales de transmisión, que se reparten entre distintos usuarios y a su vez entre sentidos de transmisión. Una de las grandes ventajas de este modo de funcionamiento es que el número de intervalos de tiempo asignados al enlace descendente y al ascendente es configurable por lo que muy apropiado para soportar tráfico asimétrico como el generado en la navegación por Internet.

Estos dos modos de funcionamiento corresponden con el primer y el segundo integrantes de las familias IMT-2000.

5.4.3 Control de potencia

En un entorno WCDMA uno de los aspectos más importantes es el control de potencia, que, junto a la gestión de traspasos o el control de congestión forman parte de la Gestión de Recursos Radio (RRM, *Radio Resource Management*). Al contrario que los anteriores sistemas de telecomunicaciones móviles de 1ª y 2ª generación, en los que el elemento a planificar era la frecuencia, en los sistemas basados en un acceso CDMA el elemento a controlar es la potencia, ya que para obtener un funcionamiento adecuado todas las señales de los diferentes usuarios deben llegar al receptor con potencias similares.

En estos sistemas, la cobertura de la celda está directamente limitada a la potencia máxima que un terminal móvil puede radiar. Si hay muchos usuarios transmitiendo aumenta la interferencia de la celda y la necesidad de potencia es mayor, con lo que los usuarios más alejados no podrán suministrarla y quedarán descolgados de la celda. En la situación inversa, cuando los usuarios activos van disminuyendo, la interferencia se reduce junto con la potencia necesaria permitiendo que usuarios más alejados entren en la zona de cobertura.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Este fenómeno tan característicos de las redes CDMA se suele denominar “respiración celular” por las distintas compresiones y expansiones de la cobertura de una celda según la carga que tenga en cada momento.

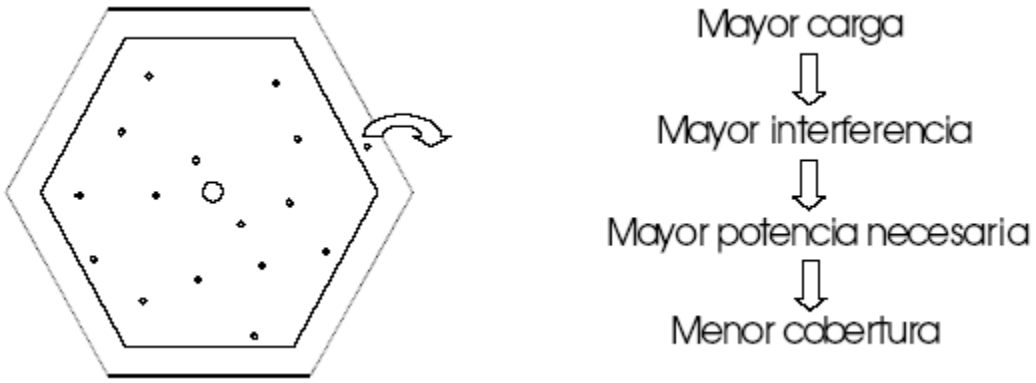


Figura 0.84: Respiración celular en CDMA

Además, según los usuarios transmitan a grandes velocidades bajará la capacidad de la celda. Si un usuario está transmitiendo a gran velocidad, utilizará un código muy corto para acercar la velocidad de *bit* lo más posible a la de *chip*. Esto provocará que la tasa de errores de la comunicación sea más alta por lo que seguramente tendrá que aumentar la potencia transmitida reduciendo la capacidad de la celda. No sólo eso, al usar un código corto reducirá el número de códigos ortogonales de forma considerable provocando en algunos casos que haya que no se puedan utilizar códigos totalmente ortogonales causando un aumento de interferencia que también reducirá la capacidad de la red.

Otra consecuencia directa del control de potencia sobre las transmisiones es la compartición automática de la carga de la red. Cuando una célula está poco cargada tiene menores interferencias y su nivel de potencia es también reducido. Esto provoca que las interferencias en las células vecinas también disminuyan provocando automáticamente que su capacidad aumente. Así, la carga de las células tiende a equilibrarse lográndose un uso más eficiente de los recursos radio. Este equilibrio de carga se logra de manera más natural que en los sistemas clásicos, en los que la compartición de carga exigía una asignación dinámica de los canales.

Para aprovechar mejor la potencia disponible en los terminales, WCDMA permite emplear redes adaptables de antenas en las estaciones base, que dirigen los haces hacia los usuarios para ofrecer un alcance máximo con un mínimo de interferencias.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

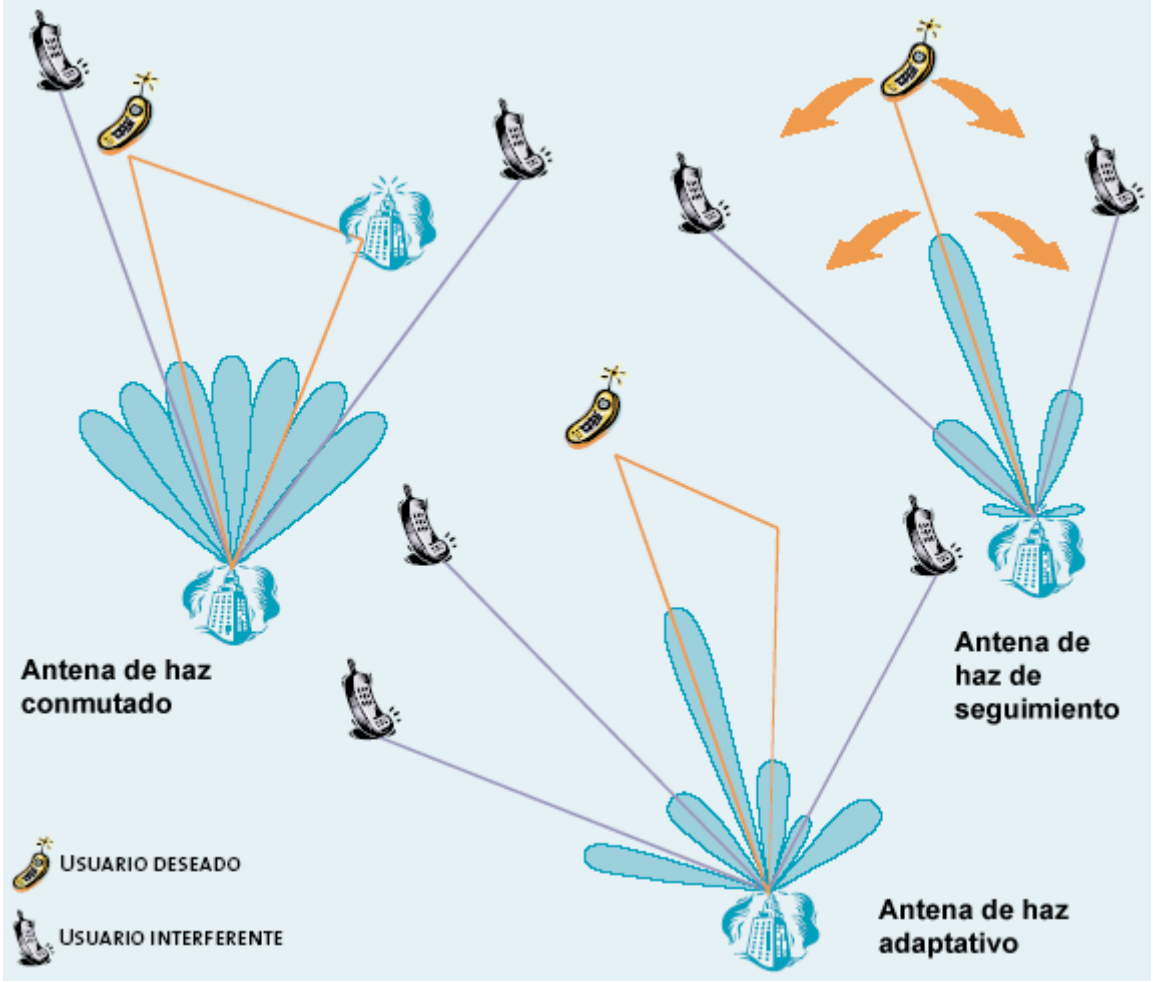


Figura 0.85: Antenas adaptables Fuente: Comunicaciones I+D nº21

Evidentemente las necesidades de control de potencia son diferentes en los dos enlaces:

- En el enlace ascendente los diferentes terminales pueden estar a diferentes distancias de la estación base por lo que sin un control de potencia muy estricto se podrían enmascarar las señales de orígenes más alejados (problema "*near-far*").
- En el enlace descendente habrá diferente nivel de señales debido al ruido térmico y a interferencias externas. De todas formas el problema aquí es mucho menor que en el enlace ascendente.

Los principales mecanismos de control de potencia en UMTS son tres:

- Bucle abierto: Se basa en estimar el nivel de atenuación de un enlace por el nivel de potencia recibido, y suponer que dicha estimación es válida para el enlace opuesto. Se suele utilizar en los momentos en los que no hay realimentación de la información de potencia como cuando se inician las comunicaciones o en la transmisión de paquetes cortos.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Bucle cerrado: Mediante un proceso de realimentación negativa, el receptor mide un parámetro de referencia (nivel de señal recibido o relación señal/interferencia), compara con el valor objetivo y ordena aumentar o reducir la potencia en el transmisor.
- Bucle externo: A partir de las condiciones de propagación y del número de usuarios ajusta el valor portador/interferencia para tener una calidad de servicio adecuada en la comunicación.

5.4.4 Traspasos en UTMS

Otra característica de UMTS diferenciadora con respecto a las tecnologías 1G y 2G, son los traspasos con continuidad o *soft handover*s. En este tipo de traspasos propios de las redes CDMA el terminal móvil puede estar comunicándose con dos estaciones base simultáneamente mientras se realiza el traspaso de las comunicaciones de una a otra.

En el enlace ascendente esto se traduce en dos recepciones de la señal en distintas base y a una selección de la mejor o incluso a una combinación de ambas para obtener una calidad óptima. En cambio, en el enlace descendente la transmisión es múltiple desde las estaciones base y la recepción se produce en el móvil combinando las señales mediante decodificadores apropiados.

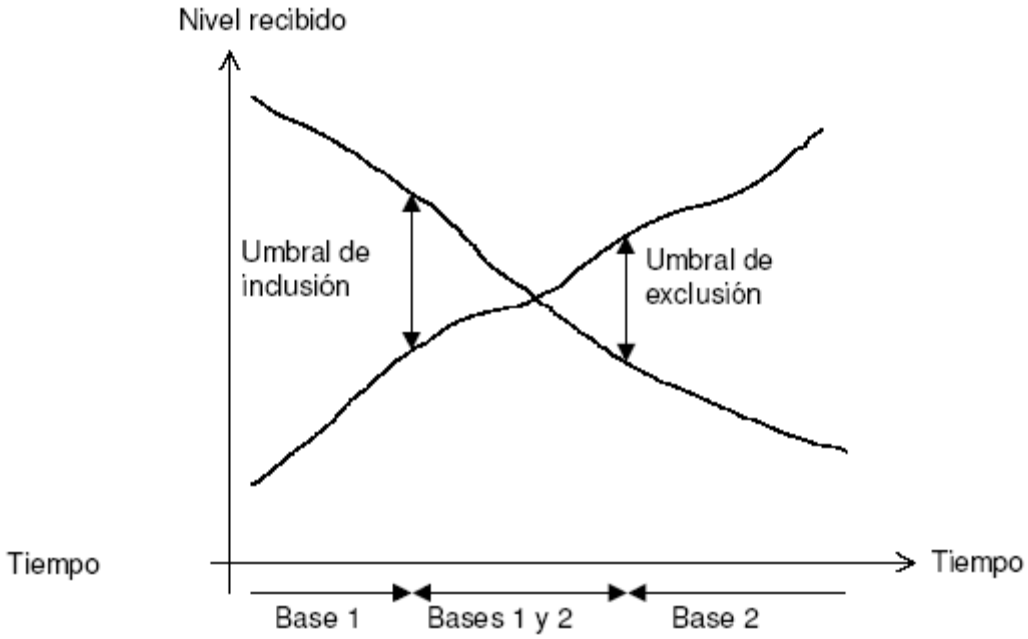


Figura 0.86: *Soft Handover* en CDMA

Este tipo de traspasos tienen una serie de ventajas importantes:

- Permiten una mayor continuidad en las llamadas
- Reducen la interferencia
- Proporcionan una mayor calidad puesto que tenemos dos enlaces y reducimos las pérdidas por desvanecimiento.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

6 OTRAS TECNOLOGÍAS 2,5G Y 3G

Ya hemos visto las principales tecnologías 2G, 2'5G y 3G que se han implantado en Europa y en gran parte del mundo. Pero hay otras tecnologías que se han desplegado o se van a desplegar en zonas muy influyentes del mundo y que merecen por lo menos una pequeña introducción.

6.1 EDGE

Si decíamos que GPRS era una tecnología 2,5G, EDGE (*Enhanced Data Rates for Global Evolution*) se podría decir que es una tecnología 2,7G. Al igual que GPRS aporta conmutación de paquetes, se monta sobre infraestructuras existentes y permite mayor velocidad. Y es en este último punto donde supera a otras tecnologías 2,5G. La velocidad máxima teórica que permite EDGE es comparable con las tecnologías 3G y alcanza los 384 Kbit/s.

EDGE se despliega evolucionando redes GSM/GPRS extendiendo los servicios de datos en modo paquete con un sistema de modulación/codificación adaptativo. De esta forma se consigue multiplicar por dos la eficiencia espectral de GPRS. Pero esta mejora tiene un coste, sobre todo de complejidad de decodificación en el receptor. Las necesidades de computación en el terminal se multiplican por 4 respecto a GPRS.

En cualquier caso, EDGE es una tecnología muy interesante puesto que supone un acercamiento a las capacidades que brinca las tecnologías 3G con un coste bastante menor. De hecho sus defensores abogan por redes mixtas EDGE-WCDMA en las que la tecnología EDGE se utilizaría para las zonas rurales consiguiendo reducciones de hasta el 50% en la inversión necesaria.

Ya existen operadores que han montado EDGE en sus redes como AT&T y Telefónica Chile entre otros. Por supuesto, ya están disponibles terminales GSM/GPRS/EDGE de fabricantes tan conocidos como Nokia o Motorola.

6.2 CDMA2000

Después de estudiar la familia de tecnologías IMT-2000, vimos que dos de ellas correspondían con UMTS y FOMA y otra tercera correspondía con la evolución de las redes americanas basadas en CDMA (IS-95) y que se conocía como CDMA2000.

Soporta 4 modos de operación de diferentes velocidades y número de portadoras. Aunque los modos que funcionan con multiportadora parece difícil que se implanten en el corto plazo. Al contrario que UMTS, que está siendo definido por el consorcio 3GPP, CDMA2000 se basa en el trabajo del grupo 3GPP2.

Esta tecnología ya ha sido desplegada comercialmente en países sudamericanos y asiáticos y por supuesto E.E.U.U.

6.3 TD-SCDMA

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Además de las cinco tecnologías que originariamente se incluyeron en el IMT-2000, posteriormente se han ampliado con dos más. La primera de estas tecnologías es TD-SCDMA, desarrollada conjuntamente por Siemens y la *China Academy of Telecommunications Technology* (CATT).

En Marzo del 2001 el 3GPP lo adoptó como parte de UMTS Release 4 como el modo de funcionamiento TDD LCR (*Low Chip Rate*).

En China ya se han reservado 55 MHz para este sistema, aunque hayan dejado más de 100 MHz adicionales que podrían compartirse con otras tecnologías.

6.4 1xEV-DO

La segunda tecnología que se incluyó a posteriori en la familia IMT-2000 fue 1xEV-DO. En realidad es el cuarto modo de funcionamiento de CDMA-2000, pero se diferencia del resto de tecnologías 3G en que está orientado únicamente a datos.

Teóricamente, soporta tasas binarias de hasta 2,4 Mbps en el enlace descendente. Su orientación a la transmisión de datos asimétricos hace que en el enlace ascendente se utilice CDMA y en el descendente se utilice TDMA, siendo únicamente un usuario a quien se transmita en cada momento.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

7 TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES MÓVILES

Durante el nacimiento y consolidación de las redes móviles, los servicios de voz son los que tradicionalmente más retorno económico han generado. Ya en los últimos años, esta tendencia ha ido cambiando produciéndose un crecimiento importante de tráfico de datos basado en SMS. El interés de todos los agentes del mercado, operadores, fabricantes y proveedores de contenidos, es que la parte de la factura del usuario móvil asociada al tráfico de datos sea cada vez mayor y de esa forma ir aumentando el mercado sin tener que mermar la rentabilidad de la voz con bajadas continuas de precios.

Vamos a dejar el análisis de la evolución y el futuro de los diferentes tipos de aplicaciones que existen en el mundo móvil para el siguiente capítulo, centrándonos en éste en las diferentes tecnologías que existen o se están estandarizando para el desarrollo de esas aplicaciones.

Una parte importante de la estandarización de la tecnologías y servicios para el desarrollo de aplicaciones móviles se está llevando a cabo en la organización *Open Mobile Alliance* (OMA). Durante este capítulo una parte importante de las tecnologías más recientes veremos que están siendo especificadas por OMA.

El consorcio OMA nace en Junio del 2002 de la mano de Vodafone, Openwave, Nokia y Cingular. Su misión se concretó en crecer el mercado de la industria móvil a través del desarrollo de estándares abiertos y de la interoperabilidad. A partir de su nacimiento, ha ido integrando varios de los foros que se estaban encargando de la especificación de estándares móviles como el WapForum o el foro de posicionamiento móvil *Location Interoperability Forum* (LIF), entre otros.

Pasamos ahora a entrar en cada una de las tecnologías, intentando dar una visión concreta y sencilla de cada una de ellas, con un objetivo principal que es cómo se pueden utilizar y para qué, y un objetivo secundario, en el que no siempre entraremos, que es cómo funcionan internamente. Para ello hemos dividido las tecnologías en cuatro conjuntos de servicios, la mensajería móvil, los servicios de localización, el acceso a Internet y las tecnologías nativas que permiten ejecutar aplicaciones en los terminales.

7.1 Mensajería móvil

Uno de los mayores éxitos de los últimos años en lo que a servicios móviles se refiere han sido los mensajes cortos de texto (SMS, *Short Message Service*). Su auge ha sido mucho mayor de lo esperado y constituyen hoy en día el mayor tráfico de datos móviles.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

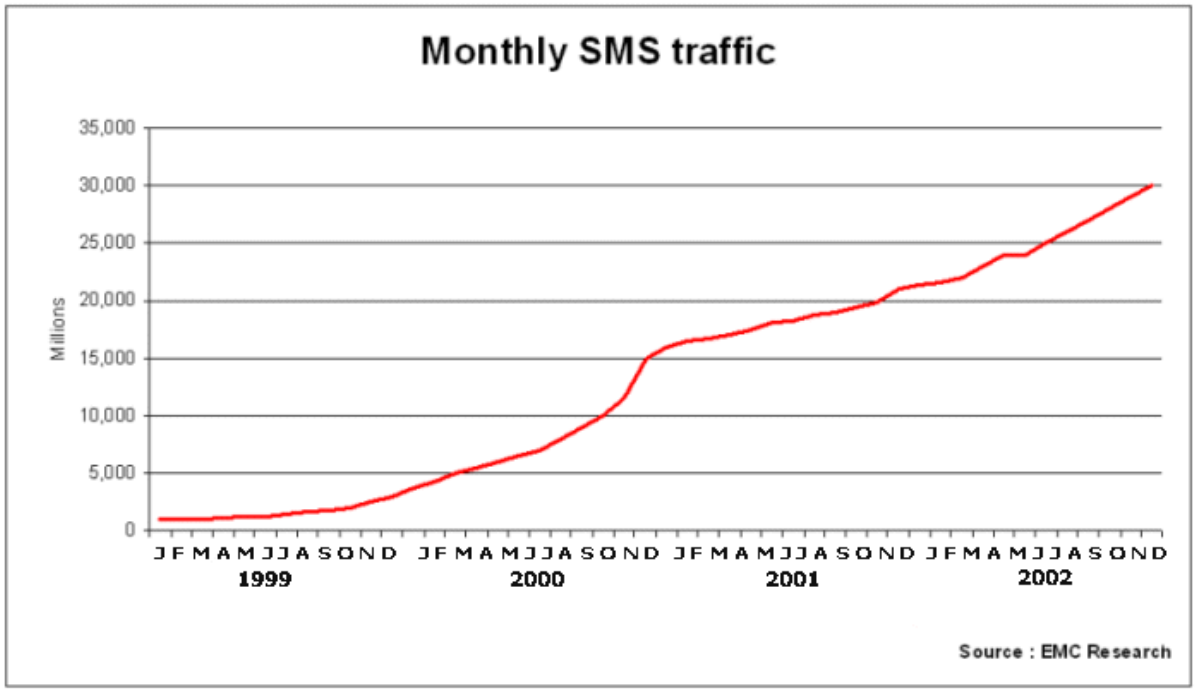


Figura 0.87: Evolución de los mensajes SMS mandados Fuente: EMC Research

A la sombra del éxito de los SMS han ido apareciendo todo tipo de tecnologías que intentaban transmitir más contenidos, más información sobre un formato de mensaje asíncrono con el objetivo de aumentar el tráfico de datos y la facturación media por cliente (ARPU, *Average Revenue Per User*). Casi todas ellas eran meros puentes hacia un futuro integrador basado en MMS, la mensajería multimedia a la que han convergido.

Como idea asociada al concepto de mensajería, casi todas las tecnologías van a proporcionar un entorno de comunicaciones asíncrono. Esto significa que cuando se realiza una comunicación basada en un mensaje el envío no tiene porqué coincidir con la recepción del mismo y además no tiene una respuesta asociada. Esto supone una arquitectura basada en centros de mensajes que se encargan de almacenarlos y reenviarlos cuando el receptor esté disponible (arquitectura *Store&Forward*).

Siguiendo esta arquitectura asíncrona se suelen dividir los mensajes en dos grupos:

- Mensajes MT (*Mobile Terminated*): Son los mensajes que van desde la red hasta el usuario final.
- Mensajes MO (*Mobile Originated*): Son los mensajes que envía el usuario y que llegan a la red.

Siguiendo esta terminología, un mensaje que un usuario manda a otro, estaría en realidad formado por dos mensajes, uno MO y otro MT. No es gratuita esta afirmación porque nos permite darnos cuenta que cuando un usuario manda un mensaje a otro realmente se está utilizando la red del operador dos veces, mientras que si desarrollamos una aplicación que envíe mensajes MT de publicidad o

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

reciba mensajes MO con órdenes, estos mensajes sólo utilizan la red una sola vez, con la consiguiente diferencia en coste asociada.

En los posteriores apartados vamos a ir analizando las diferentes tecnologías sobre las que se pueden desarrollar aplicaciones de mensajería móvil, siguiendo un orden cronológico de aparición, de las más antiguas originarias de GSM como SMS y USSD a las más modernas como PTT.

7.1.1 SMS

El servicio de mensajes cortos (SMS, *Short Message Service*) consiste básicamente en el envío de mensajes de texto en modo *Store&Forward* a través de un centro de servicio de mensajes cortos (SMSC). Este servicio permite mandar mensajes de hasta 140 bytes (160 caracteres de 7 bits) mediante la capa de señalización de la red. Esta limitación corresponde no a la red móvil, sino al tamaño máximo de los mensajes de la red de señalización número 7.

En GSM, los mensajes cortos se encaminan desde el terminal emisor hasta el SMSC donde se almacenan. Según la especificación GSM, aquí se acaba el envío de un mensaje y este mecanismo es independiente de cómo se haga llegar el mensaje a su destinatario. Cuando el SMSC decide enviar el mensaje contacta con el HLR y si el usuario está activo procede al envío del mensaje a través del MSC correspondiente.

En caso que el usuario no esté activo, el mensaje esperará en el SMSC a ser enviado sino caduca antes. Cuando el usuario se vuelva a conectar el HLR enviará una notificación al SMSC para comunicarle que el usuario ya está en línea. En todas las relaciones del SMSC con los elementos de red GSM (MSC y HLR) se utiliza otro elemento intermedio llamado SMSGateway que actúa de pasarela con el resto de la red.

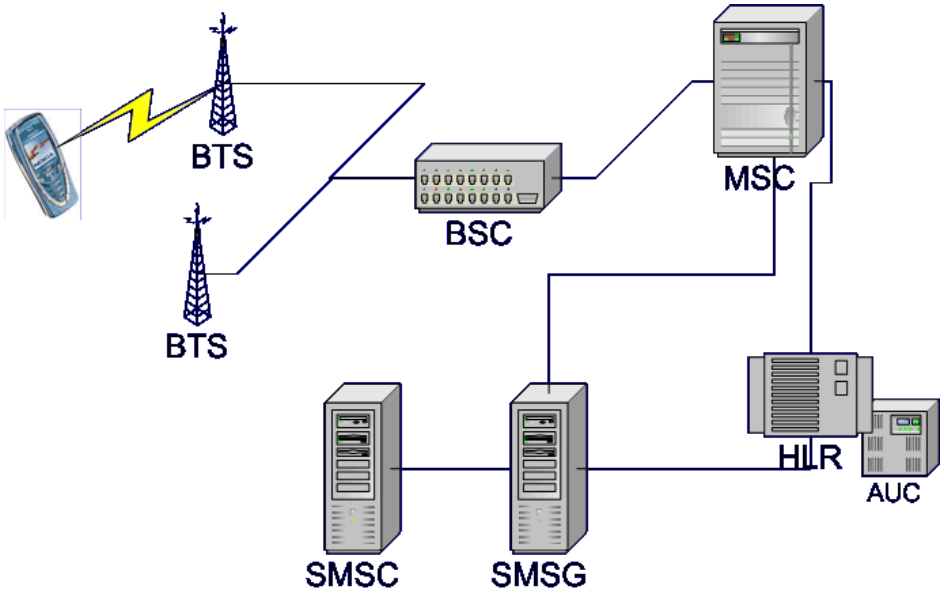


Figura 0.88: Arquitectura para el envío y recepción de SMSs

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

A la hora de comunicarse con los SMSC existen varios protocolos diferentes aunque los cuatro que han sido reconocidos por el estándar desarrollado por el ETSI (*European Telecommunication Standard Institute*) son los siguientes:

- El protocolo SMPP (*Short Message Peer to Peer*) fue creado originalmente por Logica para su SMSC. Actualmente su especificación se encuentra a cargo del SMS Forum que engloba a los principales fabricantes de SMSCs.
- CIDM (*Computer Interface to Message Distribution*) lo desarrolló Nokia para su SMSC y todavía lo mantiene y evoluciona.
- UCP (*Universal Computer Protocol*) es el protocolo asociado al SMSC de CMG y mantenido por esta misma empresa.
- Por último, OIS (*Open Interface Specification*) es el protocolo del Sema 2000 SMSC. Su mantenimiento corre actualmente a cargo de SchlumbergerSema.

Evidentemente esta multiplicidad de protocolos supone un problema a la hora de desarrollar aplicaciones compatibles con varios operadores que envíen mensajes de texto. Hay que tener en cuenta que estos protocolos tienen funcionalidades ligeramente diferentes, interfaces distintos y métodos de codificación de caracteres propios.

Las funcionalidades típicas estos protocolos son el envío de mensajes. Además, a la hora de mandar un mensaje, se pueden especificar una serie de características para el envío especialmente importantes a la hora de implementar aplicaciones sobre SMS:

- Se puede especificar la prioridad del mensaje. En caso de una prioridad alta se encolará de forma beneficiosa en el SMSC.
- También se puede indicar el momento del envío. Con esta funcionalidad se pueden programar envíos de forma anticipada.
- Otra posibilidad es incluir la caducidad del mensaje para el caso en el que el usuario no esté conectado a la red en el momento del envío.
- Para aplicaciones que envían varios mensajes al mismo usuario (por ejemplo juegos o alertas con información) se puede indicar que un mensaje sobrescribe al anterior en caso que éste no haya sido enviado.
- Por último, también se puede indicar qué tipo de mensaje se está realizando. Los terminales mostrarán al usuario el mensaje de distinta forma según el tipo del mismo de tal forma que a veces el usuario recibe notificaciones que no asocia con un SMS por el modo en el que se ven en el teléfono. A modo de ejemplo de las diferencias de visualización de los diferentes tipos de mensajes, muchas veces el terminal no deja contestar determinados tipos de mensajes. Entre los tipos posibles están, mensajes normales, mensajes de información de la red celular, mensajes de respuesta a envíos USSD (ver más adelante en el siguiente apartado) o notificación de mensajes de voz entre otros.

Después de haber analizado la arquitectura y el funcionamiento del servicio de mensajes de texto podemos extraer varios problemas del mismo:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- El SMS debe ser visto como un servicio de mensajería asíncrono y no confirmado. Los mensajes se envían pero no se sabe si se van a recibir en el instante o incluso si realmente le van a llegar al destinatario.
- Diferentes implementaciones e incluso conceptos entre los servicios de distintos países. En Europa por ejemplo suelen cobrarse a los usuarios que envían los mensajes mientras que en USA también se cobra a los receptores.
- Los terminales tienen distintas capacidades a la hora de gestionar los SMS. No sólo para soportar evoluciones de SMS cómo EMS o *Smart Messaging*, sino que además tienen diferentes tamaños de pantalla con diferentes números de letras por línea o líneas por pantalla.
- Las distintas tecnologías usadas tanto en los SMSC como en los SMSG obligan a los desarrolladores de aplicaciones a implementar diferentes protocolos con el coste que esto supone. Muchas veces esto se soluciona incorporando *brokers* de mensajes cuya labor es interconectarse con las operadoras de un país para publicar un interfaz común de envío y/o recepción de mensajes a través de números cortos facilitando la realización de aplicaciones SMS con soporte a varios operadores. Evidentemente estos *brokers* cobrar un pequeño margen de cada mensaje.

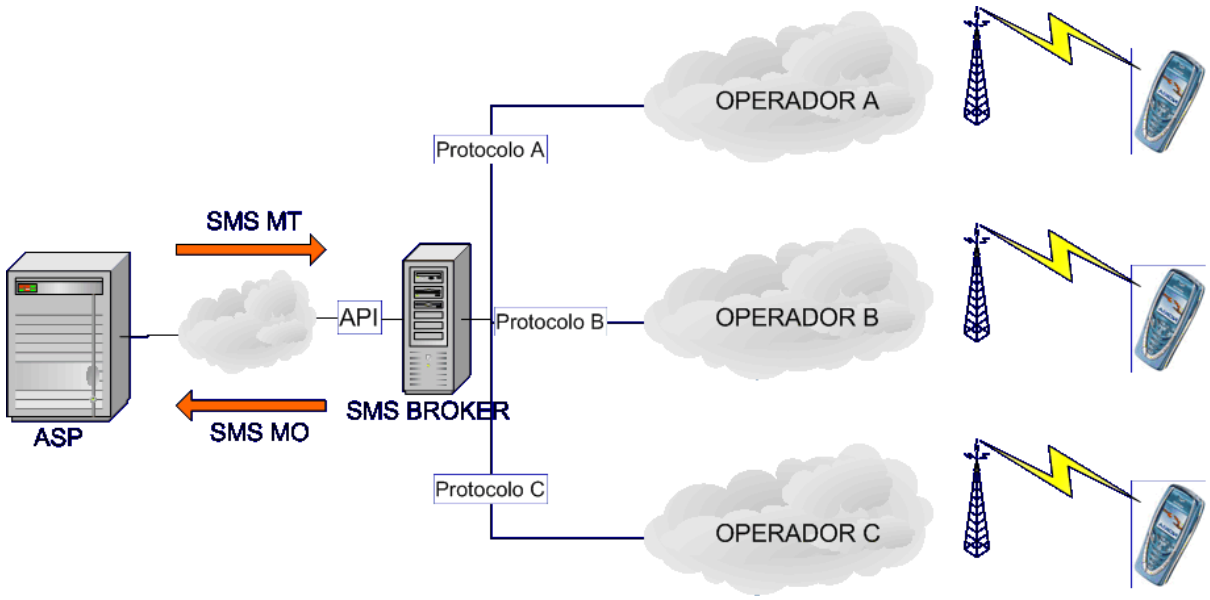


Figura 0.89: Funcionamiento de los *brokers* de SMS

El negocio de estos agentes es tratar con las operadoras para obtener buenos precios en el envío, conectarse con buenas conexiones a los SMSCs y enviar los mensajes de la forma más barata según el destino. Además publican API's basadas en HTTP y/o en Java que permiten enviar mensajes de la forma más sencilla y rápida posible.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El éxito sorprendente que han tenido los SMS entre los usuarios (sobre todo entre el público joven), ha motivado a muchas empresas a tratar de especificar y desarrollar tecnologías que permitiesen evolucionar y ampliar el servicio de SMS. Entre las diferentes propuestas que se han llegado a cabo destacan las siguientes:

- *Smart Messaging*

  El formato *Smart Messaging* fue publicado por Nokia en 1997 con el objetivo de extender los SMS mediante iconos, melodías y todo tipo de contenidos binarios como, por ejemplo, tarjetas de visita (*vCard*), notas de calendario (*vCal*), logos de operador o de grupo, etc.

  La información binaria enviada es descrita en el campo de datos del usuario del SMS. En este campo se incluye datos de la información enviada cómo el tipo de contenido mandada, su longitud o el número del puerto en el que la aplicación destino está escuchando. Además, también incluye información para segmentar el contenido en varios mensajes SMS.

- Mensajería EMS

  La tecnología EMS es un estándar abierto promocionado por el 3GPP. Permite el envío de contenidos embebidos en el mensaje SMS de formatos diversos. De esta forma en un mismo mensaje pueden llegar contenidos de diferente naturaleza dotando a esta tecnología de gran flexibilidad. Su ventana de oportunidad existió durante la transición entre los SMS y los mensajes multimedia MMS, pues es el eslabón intermedio entre las dos tecnologías.

  Su funcionamiento interno es parecido al del *Smart Messaging* utilizando la cabecera de datos del usuario para describir y mandar información relativa a los contenidos que se están incluyendo en el mensaje, información que luego el terminal tendrá que interpretar para poder decodificar los contenidos enviados.

7.1.2 USSD

La tecnología USSD (*Unstructured Supplementary Service Data*) aparece en los primeros estándares de GSM y se incluye en la práctica totalidad de todos los terminales del mercado. Aún así es una de los servicios menos usados de GSM y se puede considerar como el hermano pobre de SMS.

Su principal lacra es que no permite mensajes entre usuarios sino que sólo permite comunicarse al usuario con la red y además sólo se pueden enviar números en los mensajes MO. Junto a estas razones, su falta de uso se debe también a la falta de aplicaciones que le den una utilidad.

Básicamente, un mensaje USSD se estructura de la siguiente forma:

- Cada parámetro debe empezar por un asterisco
- Los parámetros sólo pueden contener dígitos.
- La petición debe acabar por una almohadilla

Por ejemplo, un usuario podría mandar un mensaje *23*55423# sólo escribiéndolo en el teclado y pulsando la tecla de llamada. Este mensaje llegaría al HLR que a su vez lo reenviaría a la pasarela USSD que se encargaría de mandarlo a la aplicación correspondiente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

A partir de ese momento la aplicación externa haría lo que correspondiese, pudiendo contestar al usuario con un mensaje corto SMS o mediante cualquier otro sistema.

En cualquier caso, la funcionalidad que aparentemente aporta un mensaje USSD es muy similar a un mensaje SMS MO aunque habría que distinguir ciertas ventajas:

- Un mensaje USSD es más rápido que un mensaje SMS MO. Los mensajes USSD no se basan en la arquitectura *Store&Forward* y no se almacenan, simplemente se encaminan a la aplicación.
- La arquitectura USSD es más sencilla y sus componentes de red más baratos que los de SMS. Esto significa que su coste para el operador es mucho más reducido. De hecho, esta es la principal ventaja de USSD frente a SMS MO, puesto que los operadores pueden montar servicios en los que no se cobre la navegación por los menús sin tener que soportar costes en tráfico significativos.

Aplicaciones que se adaptan perfectamente a USSD son menús de navegación (basados en listas numeradas) para seleccionar contenidos descargables como tonos y logos o una aplicación de comprobación de saldo. En estos casos el operador podría poner estos servicios de forma gratuita puesto que no supondrían gran coste en la red. Por supuesto, la descarga final del tono o del logo supondría un mensaje que se facturaría al usuario, pero éste habría navegado gratis mientras buscaba lo que quería.

7.1.3 WAPPush

Esta tecnología es parte de la especificación de WAP. Aunque podríamos haber incluido su estudio en el apartado de WAP, vamos a ver cómo funciona considerándola una tecnología de mensajería. Al final, no es más que una arquitectura para mandar mensajes MT que permitan al usuario entrar en páginas WAP. Además, su funcionamiento normal se basa en SMS y sobre WAPPush luego veremos que se monta la arquitectura MMS por lo que su interrelación con el resto de las tecnologías de mensajería es incluso más fuerte que con el propio WAP.

Al principio de WAP, su arquitectura se basaba sólo en relaciones *pull*, modelo básico de las aplicaciones cliente-servidor en donde el usuario a través del navegador del teléfono iniciaba una petición de información a un servidor reactivo que sólo contestaba si alguien le preguntaba. Es exactamente el mismo modelo que en la Web.

En contraste con este funcionamiento, existe la arquitectura *push*. En este modelo es el servidor el que inicia el envío de información sin que el cliente haya solicitado nada. De esta forma se pueden realizar multitud de aplicaciones como alertas o publicidad y fomentar el uso de WAP que en última instancia es lo que va a utilizar el usuario si la información le interesa [5-9].

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

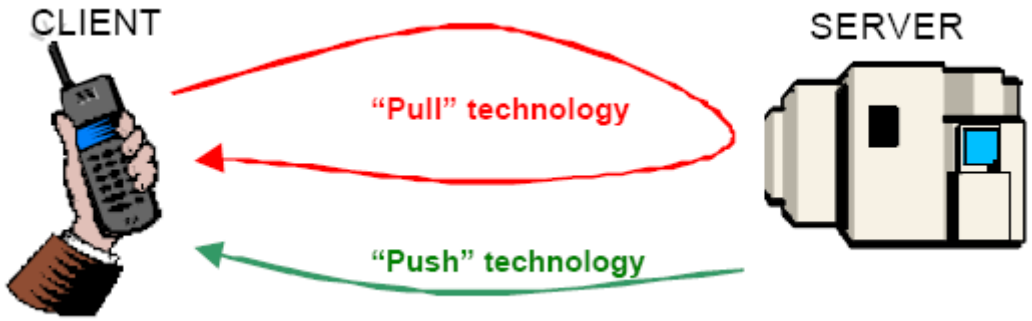


Figura 0.90: Arquitecturas *Push* y *Pull* Fuente: Open Mobile Alliance OMA

Los mensajes WAPPush se realizan enviando una información de entrega y un contenido a enviar al *Push Proxy Gateway* (PPG) que a su vez enviará ese contenido según la información especificada. Es bastante común que las funcionalidades del *WAP Gateway* (*pull*) y del *Push Proxy Gateway* (*Push*) se implementen en una misma pasarela que englobe todas las transacciones WAP.

Estos envíos se realizan a través de dos protocolos, uno entre el PPG y el servidor (*PAP, Push Access Protocol*) y otro entre el terminal y el PPG (OTA, *Push Over-The-Air Protocol*).

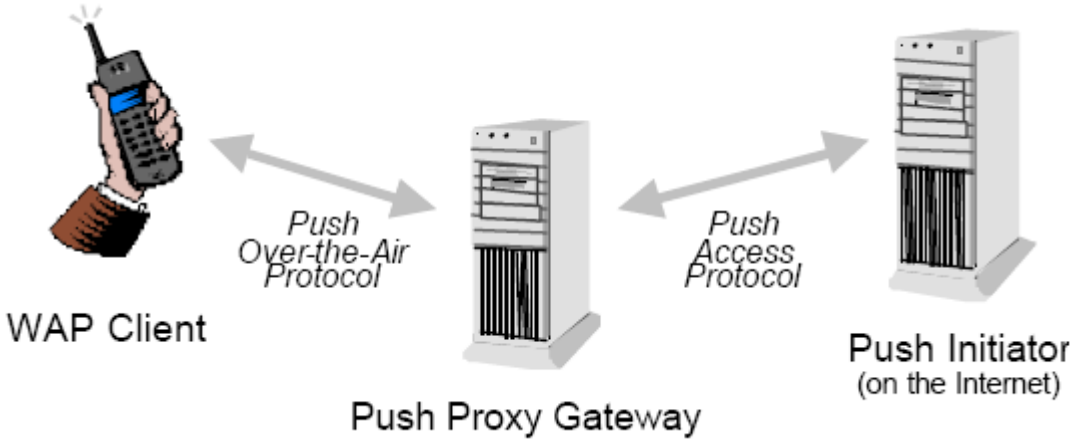


Figura 0.91: Arquitectura de envíos *WAPPush* Fuente: Open Mobile Alliance

Las tecnologías usadas para la implementación de estos protocolos se basan en estándares de Internet, HTTP y XML. El protocolo PAP permite al servidor realizar todo tipo de operaciones como:

- Envío de una notificación *Push*
- Resultado de una notificación (del PPG al servidor)
- Cancelación de una notificación
- Substitución de una notificación
- Pregunta del estado de una notificación

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Pregunta de las capacidades de un cliente

El protocolo OTA comprende varias posibilidades según el PPG conozca la IP del cliente (porque esté conectado), o si se requiere una comunicación orientada a conexión o no. No vamos a entrar a detallar todas las posibilidades, pero si vamos a explicar brevemente la más parecida a un envío *Push*. Cuando el usuario no está conectado hay que buscar la forma de que el terminal se conecte al servidor para descargarse el contenido. Se ha optado por los mensajes SMS que es una tecnología ampliamente utilizada tanto por terminales como por distintas redes. De esta forma cuando el PPG debe comunicarle algo al cliente y éste no está conectado le manda un SMS con la información de la notificación. En esta información puede indicarle donde está el contenido que se quiere mandar o sencillamente un mensaje con un pequeño texto y una URL para que el usuario la utilice si quiere. En ambos casos el usuario va a tener la última palabra para conectarse a la red o no y pagar.

Finalmente, respecto a los contenidos que se pueden enviar mediante WAPPush, lo más normal sería enviar una página WML. En la realidad otra solución que se está adoptando mayoritariamente es enviar una notificación que quepa en un SMS (esto limita bastante) con un breve mensaje de notificación y la URL de la aplicación o la página que se quiera mostrar. Por ejemplo, si se quiere realizar una alerta de goles se puede enviar la URL con la página WAP con los comentarios del partido y un mensaje indicando el goleador y el resultado actual del partido.

7.1.4 MMS

La evolución natural de los SMS son los mensajes multimedia o MMS (*MultiMedia Messaging*), mensajes con texto sin las limitaciones de SMS, con sonidos, imágenes y videos. La apuesta por este tipo de mensajería sobre redes de mayor ancho de banda como las 2,5G y 3G es clara no sólo por parte de los operadores sino también por parte de fabricantes de equipos y terminales. Los operadores buscan modelos de negocio más diversificados mediante los datos, así como de paso aumentar el ARPU (*Average Revenue Per User*) y los fabricantes mantener un ritmo de ventas que sin innovaciones no haría más que reducirse [10-12].

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Figura 0.92: Ejemplo de MMS Fuente: Nokia

Aunque cómo veremos las tecnologías implicadas en el envío de MMS son muy diferentes de las utilizadas para SMS, es muy importante que el servicio que percibe el usuario final se parezca lo más posible a la facilidad de uso de un SMS, sin perder de vista que se trata de funcionalidades más avanzadas y por lo tanto más complejas tanto a la hora de componer y escribir un MMS cómo a la hora de previsualizar o leer el mensaje. En cualquier caso el usuario debería poder escribir y enviar un MMS de forma parecida y accediendo de la misma forma que cuando lo hacía con SMS y algo equivalente se puede decir de la recepción que debería ser avisada y accesible de igual manera.

Otro punto importante es que los usuarios valoraban mucho de los SMS su coste fijo y totalmente predecible. Para el operador era sencillo puesto que los SMS tenían una longitud muy pequeña e iban a través de los canales de señalización de la red. Ahora con los MMS aparecen tamaños mucho mayores y además mucho más variables. Actualmente se está cobrando de forma proporcional mediante tramos al número de datos enviados, una aproximación muy simple al concepto de coste fijo que habrá que ver si se mantiene en el tiempo.

La tecnología MMS queda definida por una serie de características que vamos a ver por encima intentando dar una visión general del estándar para luego entrar un poco más en profundidad en la arquitectura de red necesaria para mandar MMS:

- MMS soporta distintos tipos de contenidos:
    - Imágenes: Formatos JPEG y GIF obligatorios.
    - Sonidos: AMR obligatorio. Otros posibles formatos son i-Melody, MIDI, WAV, MP3.
    - Texto plano.
    - Vídeos: Formato del archivo basado en el estándar 3GPP con codecs H263 (obligatorio) o MPEG4.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- El estándar de mensajería multimedia no sólo define los formatos de los contenidos a usar y la arquitectura de red para dar el servicio, también especifica un lenguaje de marcas basado en XML llamado SMIL que permite especificar la presentación y maquetación de los contenidos a partir de una secuencia de páginas.

```
<smil>
 <head>
  <meta name="title" content="vacation photos" />
  <meta name="author" content="Danny Wyatt" />
  <layout>
   <root-layout width="160" height="120"/>
   <region id="Image" width="100%" height="80" left="0" top="0" />
   <region id="Text" width="100%" height="40" left="0" top="80" />
  </layout>
 </head>
 <body>
  <par dur="8s">
   <img src="FirstImage.jpg" region="Image" />
   <text src="FirstText.txt" region="Text" />
   <audio src="FirstSound.amr"/>
  </par>
  <par dur="7s">
   <img src="SecondImage.jpg" region="Image" />
   <text src="SecondText.txt" region="Text" />
   <audio src="SecondSound.amr" />
  </par>
 </body>
</smil>
```

- Toda la especificación MMS incluye elementos para personalizar los contenidos al terminal de usuario. Esta personalización consistirá en recodificar los contenidos para adaptar el formato a los soportados por el terminal o bien a cambiar distintas propiedades del contenido, como dimensiones o colores para optimizar su previsualización según las capacidades multimedia del terminal.
- Al igual que SMS, los mensajes multimedia se basan en una arquitectura *Store&Forward*, en la que los mensajes son almacenados en un centro de mensajes hasta que se procede al envío. Eso sí, en los mensajes multimedia el envío no es una operación atómica como en los SMS, sino que se produce en diversos pasos que ahora veremos.

La arquitectura de red necesaria para enviar MMS gira en torno al centro de mensajes multimedia (MMSC). El estándar MMS ha sido definido por el grupo 3GPP sobre la sólida base de tecnologías ya probadas tanto en el mundo de Internet como en el móvil, como, por ejemplo, los protocolos HTTP y SMTP, los mensajes *multipart*, los tipos MIME, SMS o WAP. Esto permite una fácil integración de la mensajería móvil multimedia con otros tipos actuales de mensajería como por ejemplo un correo electrónico.

El MMSC se encarga de recibir y almacenar los mensajes para su envío posterior. Además, debe tener un subsistema de adaptación que controle el tipo y las características del terminal destino para adaptar los contenidos del mensaje a él.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

A la hora del envío, el MMSC manda una notificación WAPPush al teléfono para que éste mediante WAP se descargue el mensaje del MMSC. Es muy importante que aunque debajo del envío en realidad se encuentre una conexión al servidor, la experiencia del usuario sea lo más similar posible a la de recepción de SMS. Además si el operador no suele cobrar los SMS al receptor, tampoco sería coherente cobrar esa conexión WAP de descarga por lo que tendrá que incluir en sus sistemas de facturación este supuesto.

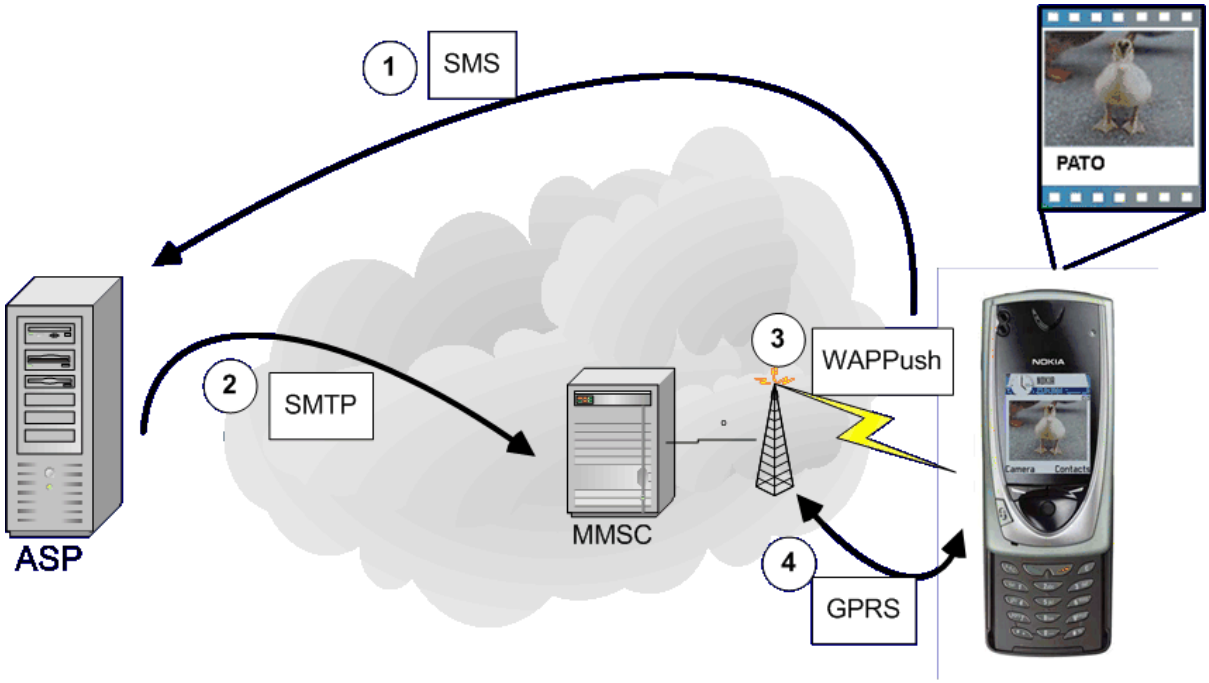


Figura 0.93: Ejemplo de provisión de un servicio MMS

Una de las problemáticas más importantes en el mundo MMS sobre todo en sus inicios son los distintos terminales MMS del mercado. Evidentemente el éxito de cualquier tecnología, especialmente de mensajería, está directamente ligado, no solamente al número de dispositivos que la soporten, sino también a la interoperabilidad entre ellos. A partir de experiencias pasadas con soluciones propietarias como el *Smart Messaging*, se ha visto que es fundamental una especificación a nivel global de la tecnología para poder hacer crecer el mercado de forma racional.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Figura 0.94: Primeros terminales MMS

Aun así, y suponiendo que todos los terminales lleguen a cumplir completamente algún día con el estándar, seguirá habiendo diferentes capacidades respecto a la memoria disponible, tamaño y calidad de pantalla, etc. Esta complejidad inherente a los diferentes terminales y a su implementación más o menos precisa del estándar debe ser resuelta de forma centralizada por el MMSC. La empresa se intuye complicada y de hecho ningún MMSC incluye un módulo de adaptación que abarque esta complejidad en su totalidad y la simplifique, aunque sí que se está avanzando en esa dirección con gran velocidad [13].

Finalmente, hay que abordar el problema de soportar el envío de MMS a terminales que no son multimedia. Evidentemente esta dificultad va a ir desapareciendo con el tiempo, pero los operadores desean tener un mecanismo para facilitar el acceso de la tecnología a todos los usuarios. Una primera aproximación a este problema es suministrar a todos los usuarios una cuenta de correo electrónico en el que almacenarán los mensajes recibidos. A partir de aquí las soluciones más complejas se basan en aplicaciones WAP, J2ME o WEB que accedan al almacén del MMSC para mostrar a los usuarios sus mensajes recibidos. Además, lo ideal sería poder contar también con funcionalidades de composición de mensajes, álbum multimedia, gestión de los mensajes o retoque de fotografías, entre otras cosas.

7.1.5 PTT

La tecnología PTT (*Push To Talk*) es seguramente la tecnología de mensajería para teléfonos celulares más moderna que casualmente se basa en uno de los conceptos de telecomunicaciones inalámbricas más antiguo. Su idea surge a partir de las comunicaciones personales sin hilos basadas en los famosos *walkie-talkies* populares y utilizados desde hace decenas de años. Básicamente su funcionamiento se reduce a enviar pequeños fragmentos de grabaciones de voz de forma asíncrona, es decir, como si de un mensaje de voz se tratara.

Los usuarios podrán mandar mensajes a otros sólo pulsando un botón y hablando, de ahí su nombre, *Push To Talk*, pulsar para hablar. El mensaje se transmite por redes con conmutación de paquetes y se recibe en el destino.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Esta funcionalidad es muy reciente en los teléfonos móviles. El operador que primero dio este servicio fue *Nextel*, operador Norteamericano que durante años incluyó en varios de sus teléfonos *Motorola* una funcionalidad que ellos llamaban *Direct Connect*. El funcionamiento era exactamente el explicado, basando su modelo en los *walkie-talkies*.

A partir de esa iniciativa, el segundo operador que se lanzó a la aventura del PTT fue *Verizon wireless*, operador de *Vodafone* y *Verizon Communications* que también da servicio en los Estados Unidos y que introdujo la tecnología PTT a mediados del año 2003. Por supuesto, también utilizó terminales suministrados por *Motorola*, como no podía ser de otra forma.

Pero no solo son estas operadoras y *Motorola* las interesadas en este tipo de mensajería. También *Ericsson*, *Nokia* y *Siemens Mobile*, han empezado a entrar en este mundo. Hasta ahora sólo había habido terminales PTT basados en CDMA pero por ejemplo Nokia ya ha lanzado el terminal 5410 con funcionamiento sobre GSM/GPRS y funciones PTT.

De hecho, tanto *Motorola*, como *Nokia*, *Ericsson* y *Siemens Mobile* encabezaron la especificación de la tecnología *Push To Talk over Cellular* (PoC), liberando su primera versión en Noviembre del 2003 y enviándola a el consorcio OMA para su integración en los grupos de trabajo del mismo.

7.2 Localización de estaciones móviles

La localización geográfica de usuarios en tiempo real es una funcionalidad intrínseca a cualquier red móvil celular, puesto que para establecer una comunicación con una estación móvil hay que saber en todos los casos en qué celda está para no tener que gastar recursos radio en todo el sistema.

Aun así, su uso está siendo menor de lo esperado puesto que las tecnologías que en teoría podrían permitir la localización de usuarios con bastante precisión no están siendo implementadas y desplegadas en las redes a la velocidad esperada.

Por ejemplo, hay que tener en cuenta que en la red GSM aunque la información del área de localización en la que está el usuario es una información existente en la red, no era accesible más que a los nodos de la red involucrados en el proceso de enrutamiento de llamadas. Además, la precisión que se puede encontrar situando al usuario en una área de localización de GSM es bastante pobre pues engloba a varias células.

Dado el gran valor añadido y el carácter diferenciador que aporta la información de localización a las aplicaciones móviles se han ido investigando y desarrollando diferentes tecnologías para facilitar la posición de los usuarios móviles con la mayor precisión posible. De esta forma, los operadores han incluido elementos de red propietarios para ir proveyendo esta información y poder servir aplicaciones basadas en ella.

Asimismo, la ETSI ha estandarizado para GSM los nodos de red y los mecanismos necesarios para poder obtener esta información y hacerla disponible tanto dentro como fuera de la red celular. Respecto a UMTS, en la red de acceso radio se han de incluir los mecanismos necesarios para obtener la

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

posición de cualquier usuario y en la red troncal se incluyen elementos de red necesarios para facilitar el tratamiento de la información de localización y así permitir crear servicios relativos a ella.

Aparte de los elementos de red necesarios para manejar la información, la localización se basa en diversas técnicas de posicionamiento de usuarios que en su mayoría se sitúan en la interfaz radio de los sistemas de telecomunicaciones móviles. Para localizar geográficamente uno a uno los usuarios de una red móvil se debe obtener información bien de los propios terminales o bien de los nodos de la red de acceso radio. Cómo inicialmente ninguno de los dos estándares proveyó tal eventualidad, se han de modificar o bien los terminales o bien los nodos de la red, o ambos. En cualquier caso, la solución no es barata ni sencilla por lo que la implantación de este tipo de tecnologías esta siendo lenta [14].

A continuación, vamos a detallar las diferentes técnicas que se pueden utilizar para localizar un usuario con cierta precisión.

7.2.1 Técnicas de localización basadas en modificación de terminales

En este apartado vamos a considerar las técnicas basadas en la incorporación de GPS en los terminales, o en la modificación de éstos para calcular la posición del usuario mediante los tiempos de llegada de la señal desde la red al terminal.

- GPS

  Incorporando un receptor GPS (*Global Positioning System*) o parte de él a un terminal se pueden utilizar la red de satélites de esta tecnología para calcular con gran precisión la posición del usuario. La mayor desventaja de este método es que los terminales aumentarían de tamaño y peso además de que tendrían menor autonomía por el mayor gasto de batería.

  De todas formas existen terminales comerciales que incorporan esta solución y una de las ramas donde puede tener mayor acogida son los dispositivos para automóviles donde la batería, el tamaño y el peso no tienen la misma importancia.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Figura 0.95: Terminal Benefon con GPS

- Tiempos de llegada (TOA) con terminales modificados

La técnica TOA (*Time of Arrival*) con terminales modificados consiste en calcular el tiempo que tarda la señal desde su salida de la red a la llegada al terminal. Si este cálculo se realiza con al menos tres estaciones cercanas se puede resolver la posición de forma bastante precisa.

Pero este método tiene dos problemas, el primero que de alguna forma el móvil debe saber cuándo salió la señal de la red, y el segundo y más complicado es que en caso de que la red mande esa información en la propia señal los relojes de los terminales deben estar sincronizados con un alto grado de precisión con los de la red. Conseguir esto último no es trivial y se traduciría hoy por hoy en un mayor tamaño y coste de los terminales.

- Tiempos relativos de llegada (TDOA) con terminales modificados

En la técnica TDOA (*Time Difference of Arrival*), también denominada E-OTD (*Enhance Observed Time Difference*), se necesita una red de estaciones base ficticias que actúan como terminales con posiciones fijas (se les suele denominar LMU, *Location Measurement Unit*). A intervalos periódicos tanto los terminales como a LMU más cercana calculan los tiempos de llegada de parte de la señal. La comparación de las diferencias de las medidas se utiliza para calcular la posición.

Este método soluciona el problema de sincronizar los terminales con la red, siempre y cuando las distintas estaciones base y las LMUs utilicen un reloj común o se conozcan los desfases en el reloj, puesto que se deben poder calcular las diferencias de tiempos reales o RTDs (*Real Time Difference*).

7.2.2 Técnicas de localización basadas únicamente en la red

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Se va a estudiar en este apartado las diferentes técnicas de localización que se pueden emplear sin tener que modificar los terminales, realizando mejoras en los nodos de la red únicamente.

- Técnicas de localización basadas en la identidad de la celda

La técnica más sencilla basada en la red y que ha sido ampliamente utilizada es aproximar la posición de un usuario por la posición de la estación base a la que está conectado.

Esta técnica es realmente muy fácil de implementar puesto que sólo había que sacar información ya disponible dentro de los nodos de red. El gran problema es la falta de precisión que supone, sobre todo en zonas rurales o menos pobladas donde el tamaño de las celdas es considerable.

Además de la celda, también se podría saber en que sector está el terminal limitando bastante la zona de localización.

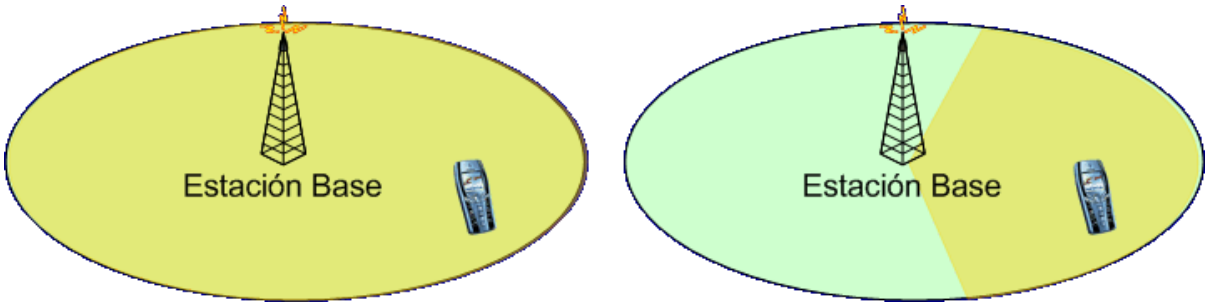


Figura 0.96: Localización basada en la identidad de la celda

- Ángulo de llegada (AOA)

Los métodos basados en ángulo de llegada también se suelen denominar Dirección de Llegada (DAO, *Direction of Arrival*). Se basan en antenas multiarray que pueden trazar una recta en la dirección de donde les llega la señal. Mediante la estimación de esta recta de dos estaciones se puede calcular la posición de un usuario aunque, si se pueden incluir más medidas, la precisión aumenta.

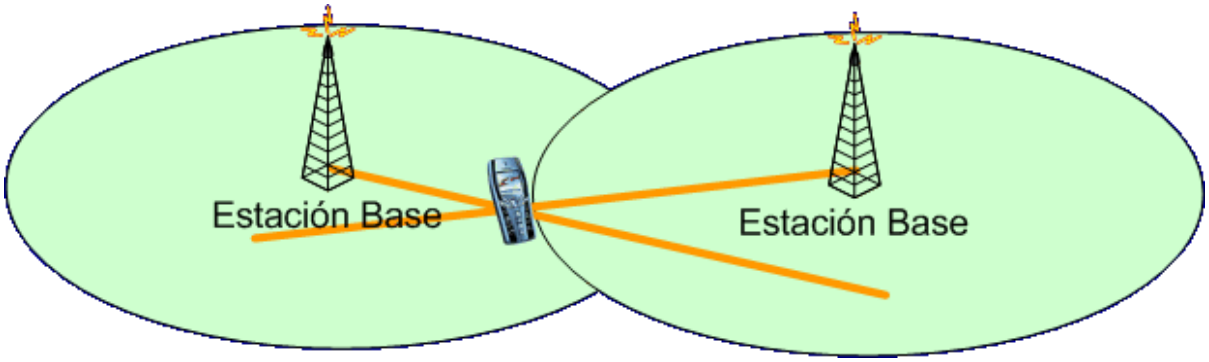


Figura 0.97: Localización mediante AOA

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Este método tiene el problema de los multitrayectos. Si un móvil no tiene visión directa de la antena la señal que recibe la antena es una reflexión de la original con lo que la recta que trazará no apuntará verdaderamente al usuario.

Otro problema es el movimiento de las antenas en tormentas o días con viento que producen pequeñas oscilaciones y reducen sensiblemente la precisión de esta técnica.

- Técnica TOA con terminales estándar

En este caso la técnica TOA se basa en calcular el tiempo que tarda la señal en ir desde la red hasta el terminal y vuelta. Las modificaciones sólo se realizan en el nodo de la red apropiado. Al contrario que su equivalente en bucle abierto (con modificaciones de terminales), no es necesario tener sincronizados los móviles con la red.

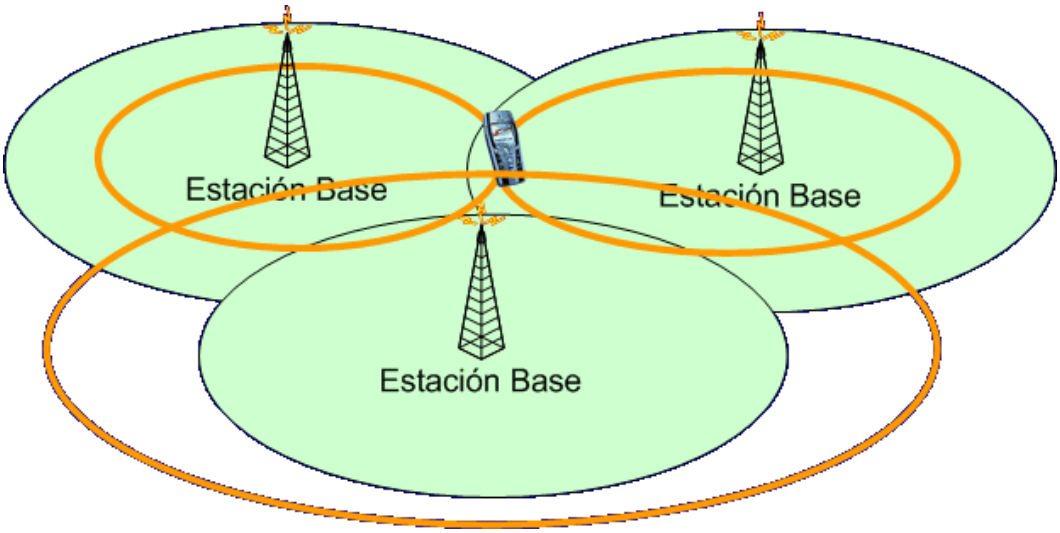


Figura 0.98: Localización mediante TOA

En este caso la dificultad aparece en el momento en el que hay que calcular cuánto tiempo tarda el terminal en contestar, puesto que este tiempo de procesamiento depende mucho del terminal, de la marca y de otros factores que provocan que su variación sea alta.

- Técnica TDOA con terminales estándar

Esta técnica consiste en calcular la correlación entre la señal recibida en dos estaciones base distintas desde un mismo terminal. A partir de esta medida se puede saber la diferencia de tiempos que ha tardado en llegar y se puede calcular el lugar geométrico (una hipérbola) en el que puede estar el móvil. Realizando esta medida con otros pares de estaciones base se puede calcular la posición del usuario.

7.2.3 Técnicas híbridas de localización

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

De forma inmediata se pueden empezar a pensar en combinar las técnicas vistas hasta ahora. Además, en general, se suelen mantener la complejidad y coste de las técnicas originales aumentando la precisión de la medida.

- Técnica TOA/AOA híbrida

  La técnica híbrida más interesante seguramente sea la combinación del cálculo de la dirección del usuario mediante AOA y el cálculo de la distancia al usuario mediante TOA con bucle cerrado.

  Esta técnica es la única de la que hemos visto que proporciona cierta precisión en el cálculo utilizando sólo una estación base. Además no tiene la necesidad de modificar el terminal.

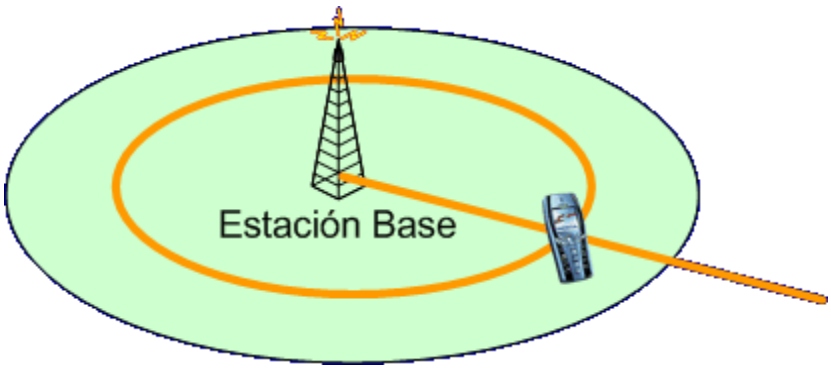


Figura 0.99: Localización híbrida mediante TOA/AOA

7.3 Acceso a Internet

Aunque el nombre de este apartado es bastante fácil de entender y de ahí su forma, quizás no sea el más correcto, ni técnicamente ni comercialmente. Como veremos en el apartado donde estudiaremos las aplicaciones móviles, un error inicial de algunas operadoras fue vender el acceso WAP como Internet en el móvil. La verdad es que el usuario enseguida pudo comprobar la distancia entre el Internet que conocía desde el PC de su casa y el acceso que obtenía desde el móvil. No son comparables por ahora y más adelante veremos las razones en más detalle.

Desde el punto de vista técnico, referirnos a WAP y I-mode como las tecnologías de acceso a Internet tampoco es lo más riguroso. En realidad cualquier tecnología que permite el acceso a un servidor publicado en Internet ya está facilitando un acceso a la información publicada en Internet. Por otra parte, realmente estas dos tecnologías sí son lo más parecido en el mundo móvil a la aplicación *World Wide Web*, puesto que tanto WAP como I-mode se basan en páginas parecidas a HTML para acceder a la información disponible en Internet (WML y cHTML). Este último punto es el que ha prevalecido para hacer la analogía entre el mundo de Internet tal y como lo conoce un usuario y el acceso desde el móvil. El error al realizar esta comparación no es muy grande pero sí significativo y hay que tenerlo en cuenta.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

En realidad, técnicamente quizás lo más correcto para definir las dos tecnologías que vamos a analizar a continuación sería describirlas como "tecnologías para aplicaciones servidoras" puesto que son precisamente en el servidor donde se realiza la mayor parte de la ejecución de la aplicación. Incluso esto tampoco es muy apropiado en I-mode puesto que, como veremos, I-mode engloba mucho más que una o dos tecnologías.

Como vemos, es difícil describir en pocas palabras a WAP e I-mode. En los siguientes apartados intentaremos dar más luz a este pequeño entramado de siglas, tecnologías y aplicaciones.

7.3.1 WAP

WAP (W*ireless Application Protocol*) es un entorno de aplicación y un conjunto de protocolos para dispositivos móviles que permiten a un acceso a contenidos de Internet y a servicios avanzados móviles.

WAP es mucho más que una pila de protocolos incluye todo un entorno para implementar aplicaciones móviles con funcionalidades desde librerías para realizar una llamada desde un enlace, hasta gestionar las capacidades del terminal para adaptar el contenido al mismo. Durante este apartado iremos estudiando la evolución tecnológica de WAP para acabar conociendo las principales características y posibilidades de esta tecnología.

WAP aparece en un momento en el que la explosión de la telefonía móvil está en su mayor auge y cuando se preveía que en pocos años el mayor número de terminales conectados a Internet iba a ser precisamente los teléfonos móviles. Significaba por aquella época unir las dos tecnologías con mayores crecimientos, la móvil y el acceso a Internet. El éxito parecía seguro.

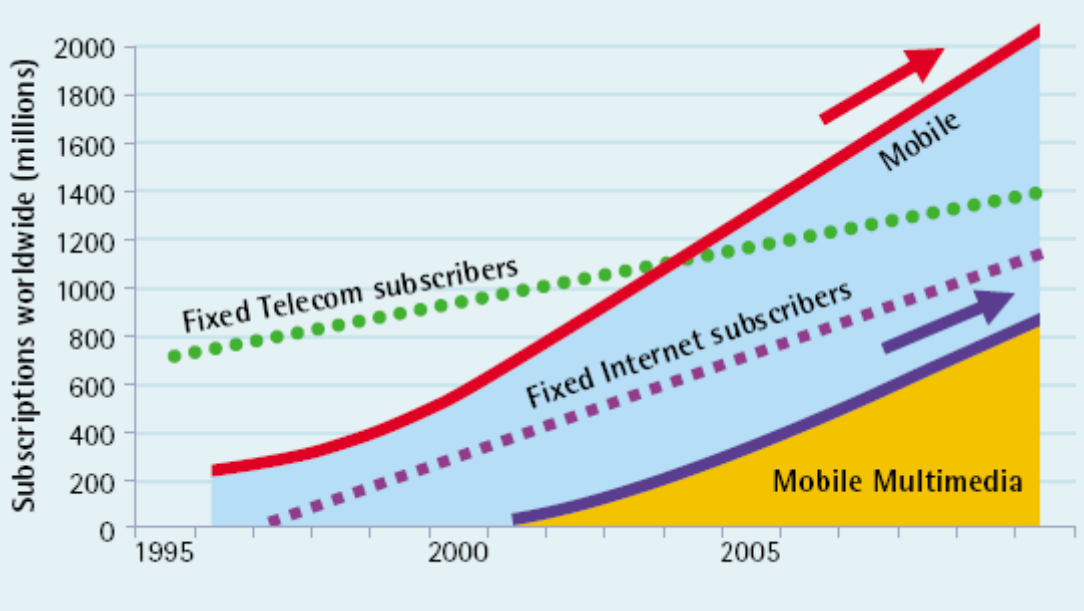


Figura 0.100: Evolución usuarios de telecomunicaciones Fuente: UMTS-Forum

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

WAP comienza a gestarse en 1997 mediante la creación del WAPForum por parte de *Ericsson*, *Motorola*, *Nokia* y *Unwared Planet*. No es un estándar propietario de una sólo compañía, sino que su desarrollo ha sido posible con la colaboración de distintos agentes tecnológicos (actualmente el número de miembros del foro de especificación para WAP supera el centenar). Los primeros terminales que soportaban la primera versión aparecen a finales de 1999, aunque realmente el desembarco de distintos modelos empieza en el año 2000. Como hemos dicho antes las expectativas eran grandes. Pero el desencanto llegó pronto. A finales del 2001 ya se consideraba a WAP como una tecnología caduca y poco exitosa. Las razones fueron varias:

- Aparecieron pocos terminales y la mayoría tenía graves problemas de implementación que provocaron que las páginas realizadas fueran muy simples para que se vieran en todos los dispositivos correctamente. Además, las pantallas eran ridículas y los gráficos tenían que ser en blanco y negro con lo que las páginas no resultaban nada vistosas.
- Las redes tampoco estaban preparadas, con GSM la velocidad obtenida era muy baja y al principio la navegación era muy lenta (>30 segundos para mostrar una página). Si a esto le sumamos que con conmutación de circuitos se paga por tiempo, el resultado es que WAP empezó a traducirse como *Wait And Pay*.
- Los contenidos tampoco estuvieron a la altura. Las operadoras no fueron capaces de involucrar a los proveedores de contenidos para que hubiera un conjunto de servicios y páginas interesante para el usuario.
- Finalmente, se intentó vender como acceso a Internet de forma móvil. Los usuarios veían gráficos minúsculos en blanco y negro, junto con listas de enlaces y algo de texto, y la comparación resultó poco afortunada creando unas expectativas que nunca se cumplieron.

Finalmente, a partir del año 2003, WAP está viviendo una segunda juventud. Las razones se deben a una mejora en la mayoría de los puntos anteriormente citados:

- Los terminales han mejorado mucho. La incorporación de pantallas más grandes y con color ha contribuido a dar mayor vistosidad a las páginas. Han empezado a implementar los nuevos estándares de WAP con inclusión de nuevas funcionalidades como *WAPPush* que ha permitido involucrar más al usuario en la navegación.
- Con la aparición de GPRS y la maduración de los elementos de red necesarios para WAP, los tiempos de espera se redujeron considerablemente (<3 segundos para mostrar una página). Además, se empezó a pagar por el tráfico consumido y no por el tiempo.
- De todas formas, los contenidos todavía son el campo de batalla de los operadores. Actualmente la tecnología se puede considerar madura con la especificación WAP 2.0 y existen desarrolladores y fabricantes de terminales con experiencia sobrada. Pero siguen faltando contenidos y servicios. Aún así este punto ha mejorado y de la mano de mejores y más abundantes contenidos vendrá la consolidación de WAP.
- La orientación comercial de WAP ha cambiado mucho. Ahora ya se habla de servicios móviles sin entrar en la tecnología ni en comparaciones con el mundo fijo.

La historia de WAP, como hemos visto es la historia de una tecnología que está tardando en arrancar. Seguramente no se ha retrasado más que la mayoría, pero las expectativas y el momento en el que surgió hizo que cualquier cosa que no fuera el mayor éxito fuera un rotundo fracaso.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Las especificaciones de WAP definen una pila de protocolos para las comunicaciones a nivel de aplicación, sesión, transacción, seguridad y transporte. Además, definen un entorno de aplicación (WAE, *Wireless Application Environment*) donde se definen los lenguajes con los que escribir las páginas con los contenidos, los formatos de los contenidos multimedia aceptados, y un conjunto de funcionalidades extra que luego veremos.

WAP se basa en el modelo cliente servidor de WEB. El cliente comienza todas las peticiones y el servidor le devuelve ficheros con la información. Es más, lo extiende con la funcionalidad de *Push* que permite al servidor indicar al usuario que quiere empezar una petición. Al igual que en la WEB, hace falta un programa residente en el dispositivo cliente que se encargue de gestionar las peticiones y que interprete todos los ficheros que le llegan para hacer con ellos lo que corresponda, normalmente mostrarlos al usuario.

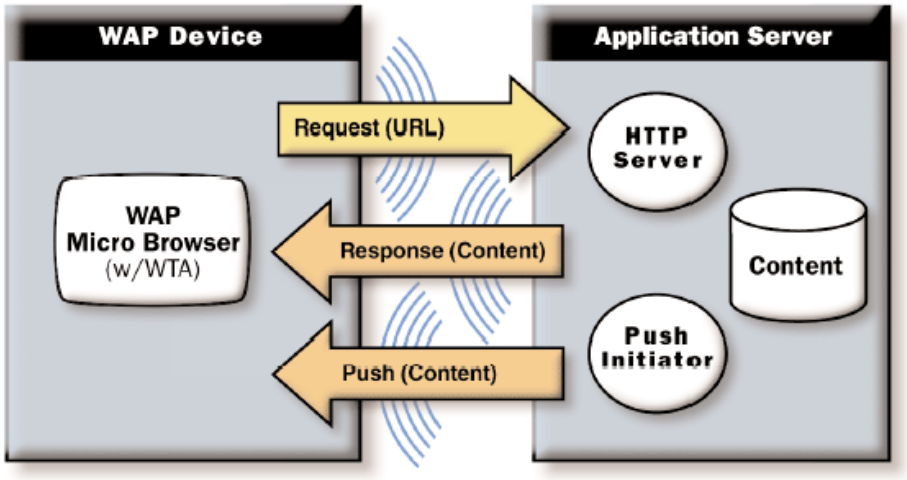


Figura 0.101: Modelo cliente-servidor con Push Fuente: Open Mobile Alliance

Actualmente hay tres navegadores o *microbrowsers* que predominan sobre los demás:

- El *microbrowser* de Nokia, que montan todos los terminales de esta marca finlandesa.
- El *microbrowser* de Openwave, que montan casi todos los fabricantes de terminales.
- El *microbrowser* de Ericsson, que montan los terminales de esta marca sueca.

Conocer el funcionamiento de estos navegadores es fundamental para conseguir una visualización óptima de las páginas que creemos. En general, una buena medida es probar todos los servicios en los tres navegadores para ver si todo se funciona correctamente. Además, también sería bueno probar diferentes versiones de los mismos para estar completamente seguros. Aunque en teoría la especificación de WAP está pensada para ser independiente de los terminales, la realidad es ciertamente distinta [15-19].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Otra característica que desde el principio se consideró en el diseño de WAP fue la independencia de la tecnología de transmisión. De hecho desde el comienzo de WAP, ya se han utilizado multitud de tecnologías para transmitir los datos. Desde SMS, pasando por conexiones de datos de GSM, CDMA hasta los más actuales GPRS, EDGE o UMTS.

La especificación de WAP ha evolucionado desde sus orígenes y ha dado tres versiones importantes con interesantes cambios, WAP 1.0, WAP 1.2 y la actual y esperada WAP 2.0.

WAP 1.0 fue la primera de las versiones de las especificaciones liberada. Una de sus principales aportaciones fue el diseño de los diferentes protocolos que permiten la conexión a Internet de los terminales.

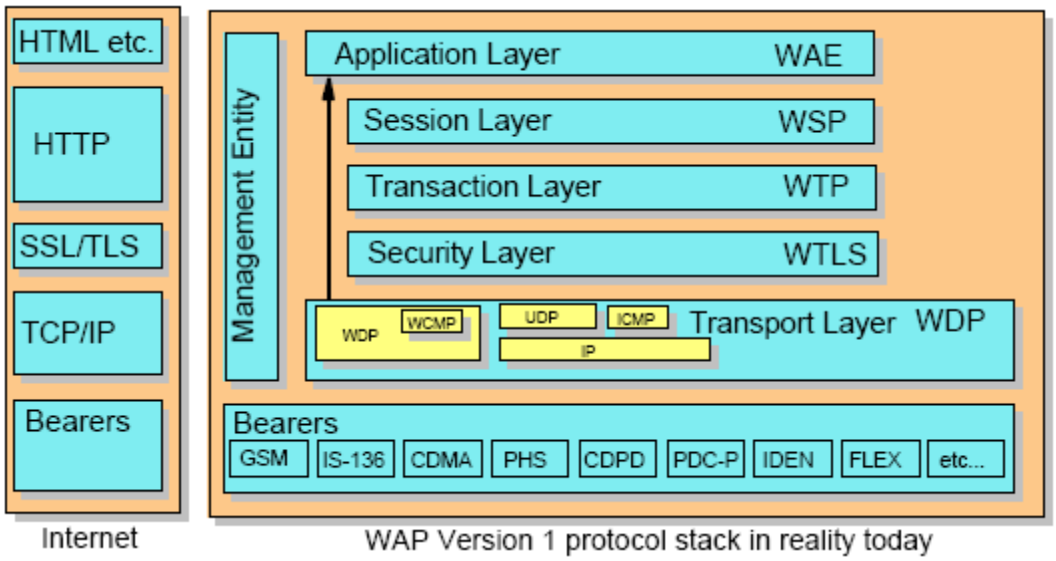


Figura 0.102: Pila de protocolos en WAP 1.0 Fuente: Open Mobile Alliance

Los principales protocolos que se especificaron son:

- WSP (*Wireless Session Protocol*), protocolo de sesión que permite intercambiar datos a las aplicaciones.
- WTP (*Wireless Transaction Protocol*), protocolo que se encarga que gestionar el modelo petición/respuesta.
- WTLS (*Wireless Transport Layer Security*), capa de seguridad encargada de la autenticación, no repudio, e integridad de los mensajes.
- WDP (*Wireless Datagram Protocol*), protocolo destinado a enviar los datos a través de la tecnología portadora correspondiente.

Inicialmente el modelo de WAP se basó en una pasarela WAP que actuaba de intermediario entre los servidores de Internet y los terminales.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

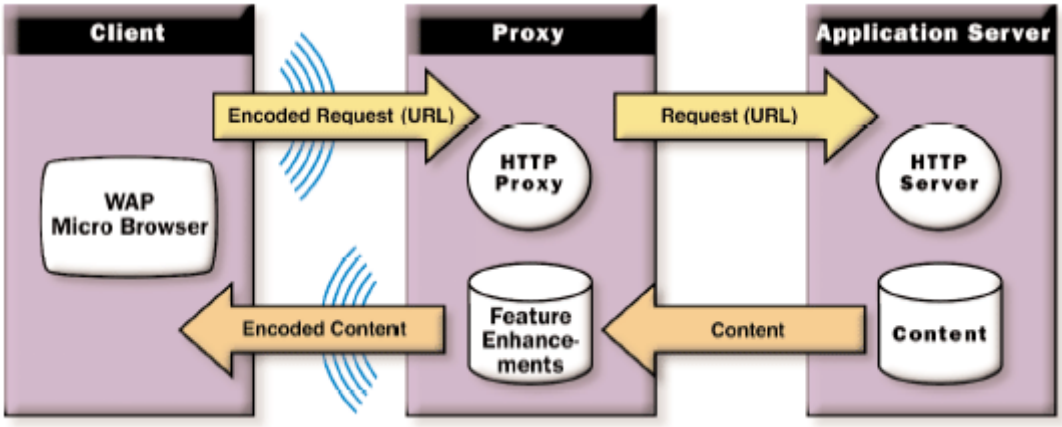


Figura 0.103: Arquitectura de WAP 1.0 con Gateway WAP Fuente: Open Mobile Alliance

El *WAP Gateway* o *WAP Proxy* es el centro de la arquitectura de red necesaria para montar WAP en un operador. Comercialmente, sirve para centralizar y controlar todas las conexiones desde los teléfonos del operador a Internet. De esa forma se puede sacar estadísticas, proveer contenidos exclusivos, facturar contenidos de pago, y en general controlar el acceso a Internet de los clientes de forma centralizada. Técnicamente, la pasarela WAP cumple las siguientes funciones:

- Actúa de traductor de protocolos entre la pila de protocolos WAP y la pila de protocolos de Internet.

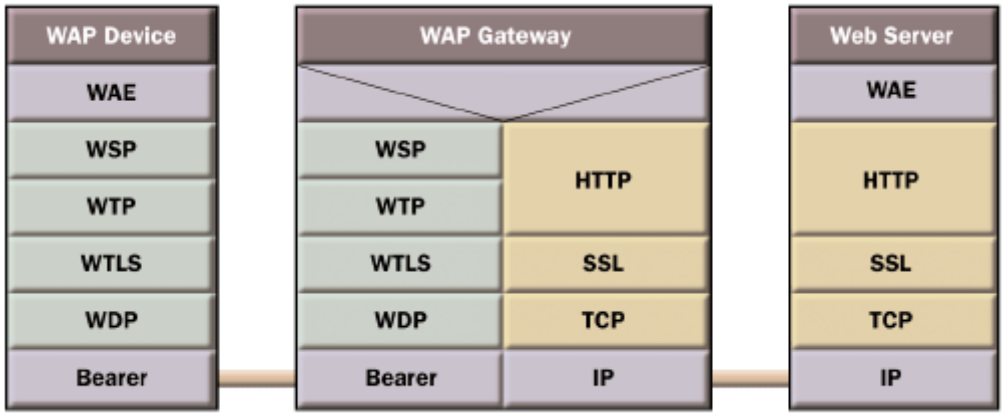


Figura 0.104: Traducción de protocolos en el Gateway WAP 1.0 Fuente: Open Mobile Alliance

- Se encarga de optimizar las comunicaciones codificando los contenidos. Los lenguajes estándares de WAP se pueden codificar convirtiendo los elementos del lenguaje en códigos hexagesimales que reducen el tamaño de las páginas considerablemente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Permite introducir y eliminar cabeceras de las peticiones y las respuestas. Interaccionando con otros elementos de la red móvil puede suministrar a los servidores que generan las páginas cabeceras con información sobre el número de teléfono o identificador del usuario, su posición, su saldo, etc.

Además de los protocolos y el funcionamiento de la pasarela, la especificación de WAP 1.0 incluía una serie de elementos que tenían sentido a nivel de la aplicación móvil y que conformaban el WAE (W*ireless Application Environment*).

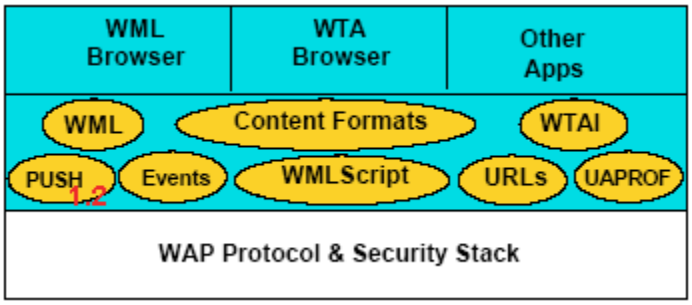


Figura 0.105: Entorno de aplicación WAE Fuente: Open Mobile Alliance

El principal componente del entorno de aplicación es el lenguaje WML. Es un lenguaje basado en XML y que aunque parecido al HTML es considerablemente más simple. Aún así, añade diversas funcionalidades no presentes anteriormente y que se deben en muchos casos a la necesidad de optimizar las comunicaciones por el escaso y caro ancho de banda móvil:

- Introducción de varias páginas en un mismo fichero. WML permite al desarrollador introducir en un fichero de WML (*deck*) varias páginas (*cards*) de forma que con una sola petición el navegador ya pueda mostrar varias pantallas. De esta forma se reduce las veces que el terminal tiene que comenzar a realizar una petición acelerando la navegación y reduciendo el *overhead* introducido por las peticiones y las respuestas.
- Variables en el navegador. WML introduce el concepto de variables en el navegador permitiendo al desarrollador guardar información en una pequeña memoria del terminal. Con estas variables, la aplicación se evita tener que estar pasándose valores en las peticiones y en las repu<estas optimizando la cantidad de datos transmitida.
- Botones. Debido a la especial forma de interactuar del usuario con el terminal, se incluyeron en WML la posibilidad de establecer enlaces y funciones sobre los botones de navegación del terminal.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<wml>
   <card id="menu">
     <p align="center">
-TÍTULO-
     </p>
     <p mode="nowrap" align="center">
Bienvenidos al portal WAP del futuro.
     </p>
   </card>
</wml>
```

Figura 0.106: Ejemplo de página WML

Además del lenguaje de marcado se incluye en la especificación un pequeño lenguaje de *script* llamado WMLScript. Este lenguaje se escribe en ficheros que luego son codificados por el *WAP Gateway* para su posterior descarga y ejecución en el dispositivo. Incluye una sintaxis muy sencilla parecida a otros lenguajes de *script* e incluye una serie de APIs que permiten realizar diversas funciones entre las que destacan:

- Manipulación de cadenas de caracteres y binarios.
- Gestión de la navegación como por ejemplo mostrar una página ya vista y almacenada en el historial, recarga de la página actual o descarga de una dirección.
- Gestión las variables WML almacenadas en el navegador. Mediante las funciones apropiadas se puede leer y modificar el contenido de cualquier variable.
- Etc.

```
/**
 * Calculate the Body Mass Index
 */
extern function calculate(height, weight) {
        var hm = height/100;
        var tmp = Float.pow(hm, 2);
        var result = weight/tmp;
        var classification;

        if(result <= 20) {
                classification = "underweight";
        } else if(result <= 24.9) {
                classification = "perfect";
        } else if(result <= 30) {
                classification = "slightly overweight";
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                } else if(result <= 35) {
                        classification = "overweight";
                } else if(result <= 40) {
                        classification = "very overweight";
                } else {
                        classification = "serious problem";
                }
                WMLBrowser.setVar("result", classification);
                WMLBrowser.refresh();
}
```

También se incluye dentro de la especificación del entorno WAE, la interfaz WTAI (*Wireless Telephony Application Interface*). Esta tecnología permite realizar llamadas dentro del código WML a funciones que permiten entre otras cosas:

- Realizar llamadas de teléfono a un número determinado.
- Mandar tonos DMTF.
- Gestionar la agenda con los números de teléfono del usuario.
- Mandar SMS

Las aplicaciones de estas funciones no han sido muy utilizadas aunque se pueden imaginar fácilmente usos como enlaces a teléfonos de consulta o enlaces para guardar números de teléfono en la agenda después de realizar una búsqueda en unas páginas amarillas. El problema de este tipo de funciones es que por un lado muchas veces son desconocidas para el desarrollador y por otro muchos terminales no las han implementado o las han realizado de forma propietaria.

Además de los lenguajes para incluir contenidos textuales, en el estándar WAP se especificó también los formatos de los contenidos gráficos que se aceptaban. Inicialmente sólo se propuso un nuevo formato gráfico llamado WBMP (*Wireless BitMaP*). Su especificación corrió a cargo del WAPForum y se simplificó de forma que sólo se podían representar imágenes en dos tonos, blanco y negro. Evidentemente el formato resultó ser muy sencillo de implementar y entender pero realmente poco potente.

Como veremos, tuvo cabida en terminales cuya pantalla sólo admitía píxeles apagados o encendidos, pero en cuanto empezaron a aparecer las pantallas a color, los formatos que prevalecieron fueron PNG, GIF o JPEG. De hecho, aunque inicialmente el formato se planteó para ser evolucionado, el WAPForum no ha mejorado el formato puesto que han decidido con buen criterio no volver a inventar la rueda.

Por último, podemos hablar de UAProf (*User Agent Profile*), una tecnología que aunque empezó su especificación con WAP 1.0 no ha sido hasta posteriores versiones donde ha empezado a utilizarse realmente con los terminales. Se basa en el estándar *Composite Capabilities / Preference Profiles (CC/PP)* del consorcio W3C, y permite a las aplicaciones conocer las capacidades del terminal que acceder y las preferencias configuradas por el usuario.

Básicamente el terminal manda en una cabecera HTTP una URL donde se encuentra el archivo con todas las capacidades del terminal. El siguiente xml muestra un ejemplo muy sencillo de este archivo.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
<?xml version=”1.0”?>
<!DOCTYPE rdf:RDF [
<!ENTITY ns-rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY ns-prf 'http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-
YYYYMMDD#'>
<!ENTITY prf-dt 'http://www.openmobilealliance.org/tech/profiles/UAPROF/xmlschema-
YYYYMMDD#'>
]>
<rdf:RDF xmlns:rdf=”&ns-rdf;”
xmlns:prf=”&ns-prf;”>
<rdf:Description rdf:ID=”MyDeviceProfile”>
  <prf:component>
    <rdf:Description rdf:ID=”HardwarePlatform”>
      <rdf:type rdf:resource=”&ns-prf;HardwarePlatform”/>
      <prf:ScreenSizeChar rdf:datatype=”&prf-dt;Dimension”>15x6</prf:ScreenSizeChar>
      <prf:BitsPerPixel rdf:datatype=”&prf-dt;Number”>2</prf:BitsPerPixel>
      <prf:ColorCapable rdf:datatype=”&prf-dt;Boolean”>No</prf:ColorCapable>
      <prf:TextInputCapable rdf:datatype=”&prf-dt;Boolean”>Yes</prf:TextInputCapable>
      <prf:ImageCapable rdf:datatype=”&prf-dt;Boolean”>Yes</prf:ImageCapable>
      <prf:Keyboard rdf:datatype=”&prf-dt;Literal”>PhoneKeypad</prf:Keyboard>
      <prf:NumberOfSoftKeys rdf:datatype=”&prf-dt;Number”>0</prf:NumberOfSoftKeys>
    </rdf:Description>
  </prf:component>
  <prf:component>
    <rdf:Description rdf:ID=”SoftwarePlatform”>
      <rdf:type rdf:resource=”&ns-prf;SoftwarePlatform”/>
      <prf:AcceptDownloadableSoftware rdf:datatype=”&prfdt;Boolean”>No</prf:AcceptDownloadableSoftware>
      <prf:CcppAccept-Charset>
        <rdf:Bag>
          <rdf:li rdf:datatype=”&prf-dt;Literal”>US-ASCII</rdf:li>
          <rdf:li rdf:datatype=”&prf-dt;Literal”>ISO-8859-1</rdf:li>
          <rdf:li rdf:datatype=”&prf-dt;Literal”>UTF-8</rdf:li>
          <rdf:li rdf:datatype=”&prf-dt;Literal”>ISO-10646-UCS-2</rdf:li>
        </rdf:Bag>
      </prf:CcppAccept-Charset>
    </rdf:Description>
  </prf:component>
</rdf:Description>
</rdf:RDF>
```

Como hemos dicho antes, WAP inicialmente fue un fracaso comercial. Las razones ya las hemos expuesto y técnicamente el WAPForum se propuso sacar rápidamente una nueva versión con algunas mejoras que hiciera de puente entre la 1.0 y la 2.0 y solventarán grandes agujeros que se vieron relevantes. Aún así, no sólo había problemas con la especificación, también había grandes problemas con las implementaciones realizadas por los terminales. Los primeros terminales tenían errores que provocaban que se “colgaran” en multitud de ocasiones. Pero lo más grave fue la limitación de varios dispositivos a un tamaño de fichero codificado menor de 1200 bytes y la nula implementación de las tablas. Esto limitó mucho tanto los contenidos textuales como los gráficos y provocó que se desarrollaran todas las aplicaciones para el caso peor, es decir para el terminal más pobre en capacidades. Las aplicaciones basadas en WAP, además de quedar relegadas al blanco y negro, se basaron en listas de enlaces simples y en texto plano. Esto sumado con la lentitud de la transmisión por GSM y el cobro por el tiempo usado provocaron una pérdida de popularidad de la que WAP todavía se resiente.

Los principales problemas que se resolvieron en la especificación WAP 1.2 fueron la ausencia de mecanismos *Push* (ya explicado en el apartado de mensajería) y la falta de *cookies* para mantener la

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

sesión con el servidor. Estas mejoras junto con la aparición de terminales a color y GPRS que permitía transmisión más rápida de la información y cobro por tráfico han permitido a WAP sobrevivir y tener ciertas esperanzas para un futuro cercano.

Con algo más de tiempo y algo más de experiencia, el WAPForum se lanzó a la especificación de WAP 2.0, la versión que actualmente implementan los terminales más novedosos. Intentaba dar solución a problemas que se habían destapado a partir de las especificaciones iniciales. Además tenía una clara orientación a dar un soporte tecnológico al acceso a Internet desde los terminales más potentes, con más memoria y mejores pantallas que estaban por aparecer en el mercado. Según el WAPForum la especificación WAP 2.0 seguía los siguientes objetivos:

- Convergencia con los estándares de Internet.
- Posibilitar nuevos servicios que puedan explotar las capacidades de los nuevos terminales y las nuevas tecnologías de red.
- Ampliar y mejorar los beneficios de las existentes tecnologías WAP 1.0.
- Gestionar la compatibilidad hacia atrás de los servicios basados en WAP 1.0 para proteger y conservar las inversiones ya realizadas.

Una de las principales novedades que aparecen en WAP 2.0 es los nuevos protocolos que aparecen en su pila. En su intento por converger en cierta forma con los estándares de Internet se aceptan los protocolos de la Red para comunicarse desde el terminal. Además se mejoran los protocolos de la antigua pila de WAP 1.0 para soportar nuevas funcionalidades relacionadas con Internet.

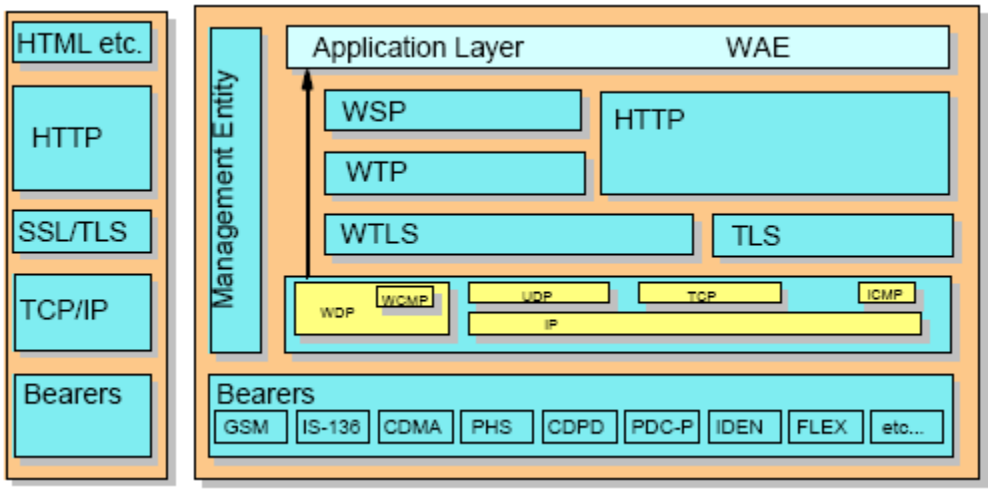


Figura 0.107: Pila de protocolos de WAP 2.0 Fuente: Open Mobile Alliance

Gracias a los nuevos protocolos añadidos, la pasarela WAP pasa a ser opcional. La labor obligatoria de traducir protocolos ha sido pasada a opcional desde el momento en el que los terminales empiezan a soportar HTTP y TCP/IP (estos últimos protocolos se adaptan al mundo *wireless* utilizando un perfil especial). Aún así, lo más lógico es que la vida de los WAP *Proxies* sea larga. Primero porque tendrán

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

que seguir dando soporte a los terminales antiguos. Segundo, porque aunque ya casi no hagan traducción de protocolos seguirán realizando labores de optimización de las comunicaciones, así como de interfaz con otros servicios móviles, como facturación, autenticación o localización. Y tercero, no hay que olvidar su componente estratégico como forma de controlar la conexión de forma centralizada por parte del operador.

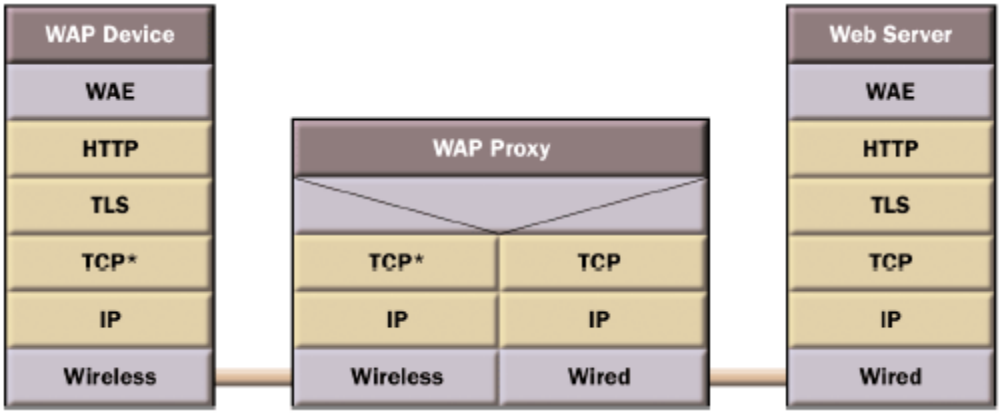


Figura 0.108: Transformación de protocolos en el Gateway WAP 2.0 Fuente: Open Mobile Alliance

Respecto a las nuevas funcionalidades del entorno de aplicación (WAE) destaca el cambio realizado sobre el lenguaje de programación WML. Siguiendo la política de la necesaria convergencia a Internet, se realiza un intento por hacer converger los dos lenguajes que predominaban en esos momentos, el HTML y el WML.

El WAPForum se coordinó con el consorcio W3C para utilizar el último estándar de éste. A partir de HTML 4.0 aparece el estándar XHTML que básicamente añade modularidad (cualidad indispensable para luego escoger los módulos que van a formar los distintos sublenguajes) y una mayor rigurosidad a la hora de la sintaxis de las páginas. El *XHTML Mobile Profile* (XHTMLMP) nace como un estricto subconjunto de XHTML mediante la conjunción del módulo inicial XHTML Basic y algunas extensiones como soporte para hojas de estilo y algunos elementos de presentación extras.

Además de este lenguaje aparecen un subconjunto del lenguaje CSS (WAP CSS) para realizar hojas de estilo y tratar de separar los contenidos de la presentación. Desaparecen algunos elementos del lenguaje no apropiados para las pantallas de los terminales móviles y se añaden algunos pequeños detalles que facilitan el desarrollo WAP, como por ejemplo el formato de los campos *input*, estilos para texto con *scroll* horizontal automático y teclas especiales para lanzar acciones.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

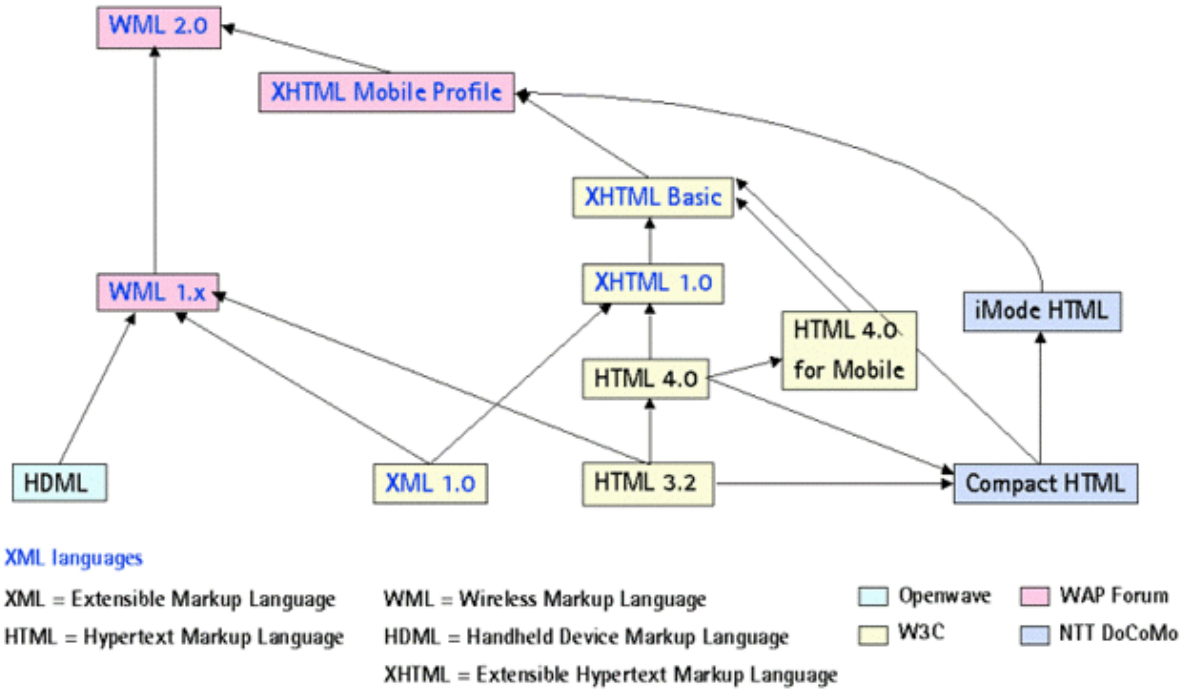


Figura 0.109: Evolución de los lenguajes de marcado para móviles Fuente: Nokia

Puesto que XHTML no surge a partir de WML sino de HTML 4.0, los elementos que éste no disponía y que sí que tenía WML 1.x desaparecen en XHTML. Todos estos elementos se incorporan en WML 2.0 extendiendo XHTML mediante una serie de atributos y elementos que comienzan con el *namespace* wml. Hay que tener en cuenta que la especificación obliga a los navegadores a seguir siendo compatibles con WML, por lo que interpretar este tipo de elementos no debería ser muy complejo. En la siguiente tabla se pueden encontrar los elementos que así se añaden:

Elemento nuevo		Elemento XHTML	Atributo nuevo
wml:acces		body	wml:onenterforward, wml:onenterbackward, wml:ontimer, wml:new-context
wml:anchor			
wml:card		html	wml:onenterforward, wml:onenterbackward, wml:ontimer, wml:user-xml-frag-ments
wml:do			
wml:getvar		img	wml:localsrc

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

wml:go		input	wml:emptyok, wml:format, wml:name
wml:noop		meta	wml:forua
wml:onevent		option	wml:onpick
wml:postfield		p	wml:mode
wml:prev		select	wml:value, wml:name, wml:ivalue, wml:iname
wml:refresh			
wml:setvar		textarea	wml:emptyok, wml:format, wml:name
wml:timer			

Tabla 0.6: Elementos WML añadidos a XHTML

Otras funcionalidades del WAE como el WAPPush, el WMLScript o WTAI mejorar discretamente o se mantienen de la misma forma que en anteriores versiones.

Además de la pila de protocolos y el entorno de aplicación, aparecen nuevos componentes en la especificación de WAP 2.0 que permiten mayor potencia en el desarrollo de aplicaciones. Entre estas funcionalidades destacan un API para almacenar datos en el teléfono (*Persistence Storage Interface*), funciones para configurar los teléfonos automáticamente (*Provisioning External Functionality*) o capacidades para sincronizar datos desde los terminales mediante *SincML*.

A principios del año 2002 ya estaba liberada la versión inicial de WAP 2.0. Y a mediados de ese mismo año desaparece el WAPForum. En realidad, lo que sucede es que se integra en la iniciativa OMA (*Open Mobile Alliance*) que nacía en esos mismos momentos. A partir de esa época, las actividades del WAPForum se distribuyen entre los grupos de trabajo del OMA, ampliando mucho la variedad de ámbitos trabajados y estudiados.

7.3.2 I-mode

El operador móvil dominante en Japón, NTT-DoCoMo, desarrolló una serie de tecnologías que unidas con un exitoso modelo de negocio ha convertido a i-mode en un servicio estrella desde su lanzamiento en 1999.

La primera consideración que hay que hacer sobre i-mode es que va más allá de una simple especificación técnica. Cuando se habla de i-mode se suele hacer referencia, no sólo a un conjunto de tecnologías, sino también a la forma de aplicarlas al mercado japonés. Por lo tanto seguramente no sea lo más correcto encuadrarlo en este apartado, aún así no sería lógico tratar las tecnologías de acceso a Internet mediante móvil sin hablar de la única que ha conseguido ser usada masivamente.

I-mode nace en 1999 de la mano de NTT-DoCoMo con la idea de sacar un servicio de datos con acceso a páginas con contenidos y aplicaciones de forma rápida y sencilla. Las primeras especificaciones técnicas no distaban mucho de las capacidades de WAP 1.0, si bien tenían varias diferencias

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

notables, como que utilizaban cHTML en vez de WML y que la red sobre la que se transmitía funcionaba mediante conmutación de paquetes.

Pero seguramente lo que más influyó en el éxito de i-mode fue la estrategia y la actitud de NTT-DoCoMo con el servicio. En el funcionamiento de i-mode intervienen varios actores, pero todos están controlados y coordinados por el operador. Éste comercializa con su marca todos los terminales, entablando intensas relaciones con los fabricantes para obtener los terminales según sus especificaciones. Además, también mantienen importantes conexiones con los proveedores de contenido filtrando y puliendo todas las páginas oficiales. Finalmente, también impulsa el servicio desde los canales directos e indirectos de venta de terminales y contratos.

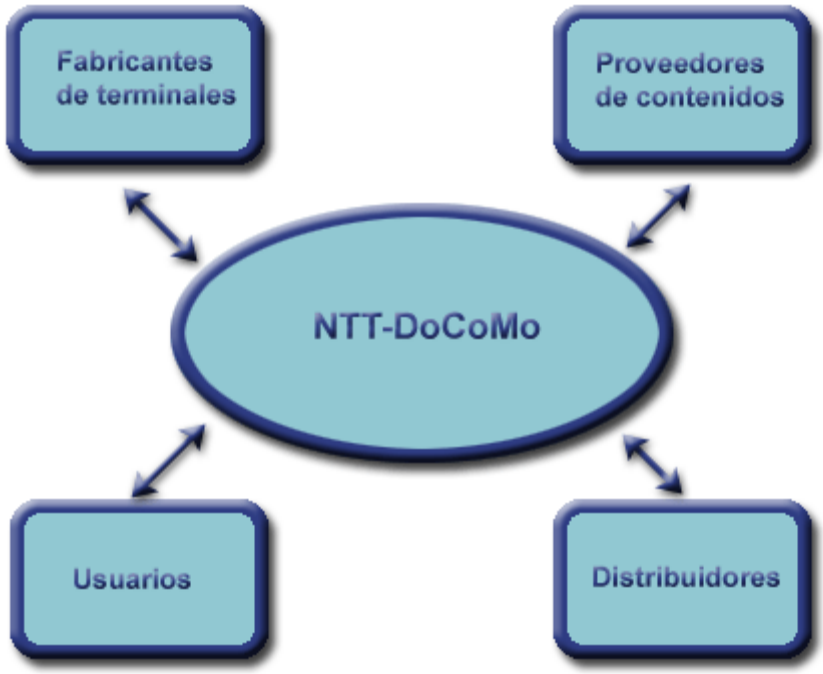


Figura 0.110: Agentes de i-mode

El éxito de i-mode ha sido total. Desde que nació en 1999 sólo necesitó 2 años para alcanzar 15 millones de usuarios y actualmente ya tiene más de 40. Números muy superiores a cualquier servicio montado sobre otras tecnologías móviles exceptuando los SMS.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

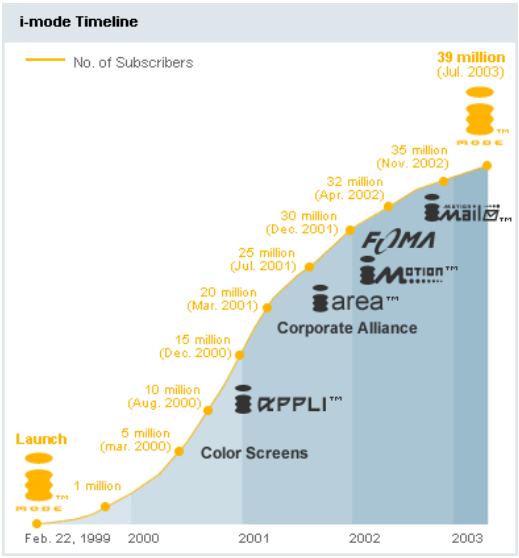


Figura 0.111: Evolución de usuarios y servicios de i-mode Fuente: NTT-DoCoMo

Técnicamente i-mode es especificado y desarrollado totalmente por NTT-DoCoMo. Incluso la tecnología de red es propietaria del operador. Eso le permitió sacar un servicio rápidamente y de forma muy completa y coordinada en poco tiempo, así como poder evolucionarlo con mayor velocidad. Inicialmente se basaba en una red de conmutación de paquetes a 9600 bps. Posteriormente han ido mejorando considerablemente las velocidades hasta las más recientes conseguidas con redes y terminales 3G.

En la parte de las aplicaciones, se creaban las páginas mediante páginas cHTML (una versión reducida de HTML) y con imágenes GIF en blanco y negro. Esto ha ido evolucionando rápido, incluyendo imágenes a color, y todo tipo de servicios más avanzados como descarga de aplicaciones i-appli (tecnología que veremos en el siguiente apartado), descarga de vídeos, localización, etc.

7.4 Aplicaciones nativas en los terminales

Un conjunto muy importante de aplicaciones móviles se está desarrollando utilizando tecnologías que permiten que la aplicación se ejecute en el terminal directamente. Muchas de ellas soportan APIs para acceder a recursos del teléfono como enviar SMS, escribir en la pantalla o acceder a Internet por lo que las posibilidades que ofrecen son realmente muy variadas.

Este tipo de aplicaciones tienen tres grandes ventajas respecto a las aplicaciones basadas en tecnologías que se ejecutan principalmente en el servidor como WAP:

- Puesto que normalmente se dispone de APIs gráficas la espectacularidad visual es más fácil de conseguir que en otras tecnologías.
- Otra gran ventaja de tener la aplicación ejecutándose directamente sobre el terminal es que permite una mayor interactividad con el usuario.
- La tercera ventaja hace referencia la utilización óptima del ancho de banda disponible en las conexiones a Internet. Mientras que las aplicaciones basadas únicamente en arquitecturas

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

cliente/servidor utilizan el ancho de banda para transmitir todos los menús así como el código de marcado, en las aplicaciones nativas se puede ir navegando por los menús sin mandar ni un solo byte. Esto permite seleccionar la información a descargarse sin gastar ancho de banda. Además, en el momento de descargar la información requerida se van a transmitir los datos necesarios, sin incluir ningún tipo de lenguaje de marcado.

Quizás la desventaja más evidente de las tecnologías nativas respecto a las aplicaciones que se ejecutan en el servidor es que una vez instaladas las aplicaciones en los terminales del usuario es muy complicado cambiar mensajes, configuraciones o solucionar errores. Por eso es muy importante apoyarse en un servidor si hay configuraciones o mensajes muy importantes y que posiblemente cambien (como por ejemplo mensajes sobre el precio o el premio a dar). También es extremadamente importante comprobar que no hay errores en la aplicación cliente puesto que en caso de detectar alguno será muy complicado de cambiar una vez publicado para los usuarios.

Un punto importante en este tipo de tecnologías son los terminales que las soportan. Primero porque es relevante saber qué número de dispositivos hay disponibles en el mercado y cuántos fabricantes tienen previsto sacar modelos con soporte. Además, no hay que olvidar la calidad de las implementaciones de la tecnología en cuestión. Si se produce una gran variedad de implementaciones con diferentes características el desarrollo de aplicaciones se va a ver condicionado a crear diferentes versiones para acomodarse correctamente a todos los dispositivos.

A continuación, vamos a ver las principales tecnologías nativas para terminales móviles. Como veremos gran parte de estas tecnologías son utilizadas para el desarrollo de juegos y aplicaciones de entretenimiento. Principalmente porque son las tecnologías que facilitan la interactividad con el usuario y la vistosidad gráfica.

7.4.1 J2ME

A mediados de los años 90 aparece el lenguaje de programación Java de la mano de Sun Microsystems. Un lenguaje de alto nivel, basado en la sintaxis de C++, multiplataforma con una seguridad basada en el modelo *sandbox*, necesario pues inicialmente estaba orientado a comunicar pequeños electrodomésticos y al mundo de Internet.

Las plataformas de desarrollo Java se basan tanto en un lenguaje de programación como en un entorno de ejecución llamado máquina virtual. A partir del Java original han ido apareciendo diferentes ediciones que hacían frente a necesidades particulares de los diferentes mercados.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

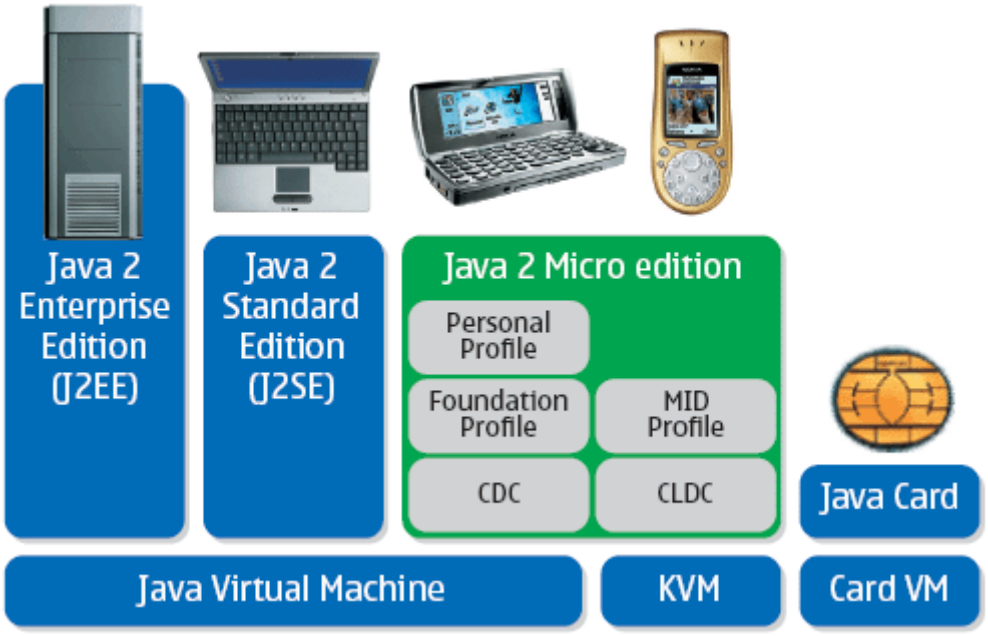


Figura 0.112: Situación de J2ME en el mundo Java Fuente: Sun Microsystems

J2ME (*Java 2 Micro Edition*) se encuadra dentro de las ediciones de Java como la versión desarrollada para terminales pequeños e inalámbricos tales como PDAs o teléfonos móviles. Su objetivo es proveer un entorno desde el que se pueda descargar aplicaciones para luego instalarlas y ejecutarlas en el dispositivo móvil. Gracias al consenso y coordinación de todos los agentes del mercado móvil J2ME se ha convertido en la tecnología de referencia para realizar todo tipo de aplicaciones descargables y ejecutables en el dispositivo.

A finales del año 2003, el número de modelos disponibles con J2ME rondó los 200 y el número de terminales vendidos superó los 100 millones ampliamente. Con este amplio despliegue de terminales y unido con las cantidades ingentes de desarrolladores de Java el éxito de esta tecnología estaba asegurado desde sus inicios [20-23].

La tecnología J2ME se fundamenta en tres pilares fundamentales sobre los que giran las diferentes versiones disponibles del entorno de ejecución de los dispositivos:

- Máquina virtual

  La maquina virtual de Java (JVM, *Java Virtual Machina*) es el programa que se encarga de traducir el *bytecode* que tiene el código Java compilado a instrucciones válidas en código máquina. Además también mantiene todas las cuestiones de seguridad tales como impedir a los programas acceder partes del sistema no permitidas (por ejemplo la agenda con los números de teléfono).

  Su principal función es dar independencia al código escrito respecto al terminal donde se va a ejecutar. Actualmente hay dos JVM incluidas en J2ME. Por un lado está la CVM, máquina virtual con todas las características de Java y que debe ejecutarse en dispositivos con unos requisitos de

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

memoria y potencia bastante altos. Además, existe la KVM, mucho más pequeña y ligera pero restringida en su funcionalidad. Por poner varios ejemplos, no dispone de cálculo en coma flotante, no posee reflexión de clases y no tiene soporte para código nativo.

- Configuración

Una configuración es un conjunto de APIs básicas que definen un entorno general de ejecución. Intentan agrupar terminales por características muy generales como por ejemplo sus capacidades computacionales o si tienen conectividad o no. Cuestiones más cercanas al terminal o más particulares, como las librerías gráficas nunca se meten en la configuración.

Actualmente hay definidas dos configuraciones:

- *Connected Device Configuration* (CDC), para dispositivos dotados de conectividad y con (relativamente) alta capacidad computacional.
- *Connected Limited Device Configuration* (CLDC), para dispositivos con capacidades (proceso y memoria) limitadas dotados de conectividad.

- Perfiles

Un perfil es un conjunto de APIs para una configuración dada y un entorno de aplicación en particular. Los perfiles intentan agrupar los dispositivos donde van a correr según las funcionalidades de éstos y según el tipo de aplicación que se va a ejecutar en ellos.

Un punto importante en los perfiles es la librería gráfica. Habrá perfiles que sólo incluirán en pantallas de texto, otros serán pantallas gráficas y táctiles, etc. Incluso los sistemas embebidos pueden que no tengan interfaz gráfica alguna.

Actualmente hay definidos o están en proceso cuatro perfiles, entre los que destaca por su mayor aplicación al mundo móvil el último:

- *Foundation Profile* (FP), consiste en un conjunto de APIs básicas para la configuración CDC que no incluye interfaz gráfica. Debido a su simplicidad está pensado para ser extendido por otros perfiles.
- *Personal Profile* (PP), incluye una librería gráfica con capacidades WEB y *applets* y tiene sentido en el contexto FP/CDC.
- *PDA Profile* (PDAP), perfil para la configuración CLDC pensado para PDA de gama baja, con bajas capacidades de memoria y procesamiento, pero con una pantalla táctil de al menos 20.000 píxeles. Su orientación está claramente influenciada por las agendas Palm.
- *Mobile Information Device Profile* (MIDP), se basa sobre la configuración CLDC y tiene las características necesarias para funcionar con los terminales móviles, comunicaciones limitadas, capacidad gráfica baja, entrada de datos simple, memoria y potencia bajas. Incluye APIs para el almacenamiento de datos en el terminal, conectividad basada en http 1.1, temporizadores, entrada de datos del usuario y un entorno de ejecución básico basado en *midlets*, aplicaciones reducidas que se ejecutan en la máquina virtual.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

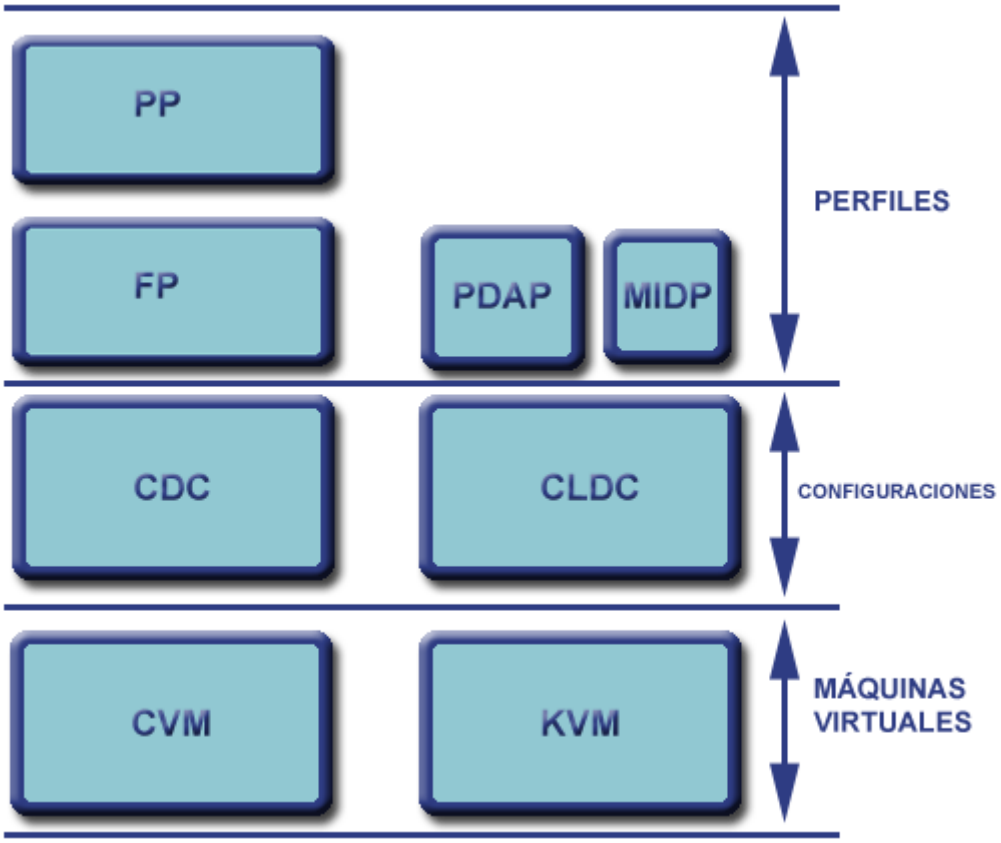


Figura 0.113: Arquitectura de J2ME

Casualmente el gran número de modelos que han surgido en torno a esta tecnología ha sido su mayor aliado pero a la vez ha supuesto una importante barrera para el desarrollo de aplicaciones. La problemática surge en torno a la dificultad de hacer aplicaciones que aprovechen toda la funcionalidad del terminal y que funcionen en más de dos modelos diferentes. Cierto es que se puede desarrollar cualquier aplicación para que funcione en todos los modelos pero en ese caso surgen dos problemas interrelacionados:

- Por un lado no se realizará una aplicación que aproveche todas las ventajas del terminal y todas las capacidades gráficas y de interfaz de usuario.
- Por otro, según queramos abarcar más terminales de forma que aprovechemos todas las capacidades la aplicación irá teniendo cada vez más tamaño.

Al final la solución a la que han llegado todos los proveedores de aplicaciones ha sido agrupar los modelos por familias de 2 ó 3 miembros como mucho, con características similares (sobre todo la

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

pantalla) y desarrollar diferentes versiones de la aplicación a partir de código común para cada familia. Esto permite conseguir aplicaciones muy vistosas gráficamente y con tamaños muy ajustados a costa de aumentar el coste de desarrollo y mantenimiento considerablemente.

Las aplicaciones que se han ido desarrollando sobre J2ME han sido de lo más variadas. Desde el entretenimiento puro como los juegos, hasta aplicaciones de información del tiempo, de cotizaciones, etc.. En realidad las únicas aplicaciones que han brillado por su ausencia han sido las orientadas al *m-commerce*. J2ME no ha tenido mucho éxito en aplicaciones como compra de entradas, gestión de bancos o aplicaciones corporativas. Principalmente por la falta de mecanismos de seguridad en las primeras especificaciones y implementaciones. Este problema se ha corregido en las últimas versiones de las especificaciones y pronto veremos un nuevo conjunto de aplicaciones J2ME.

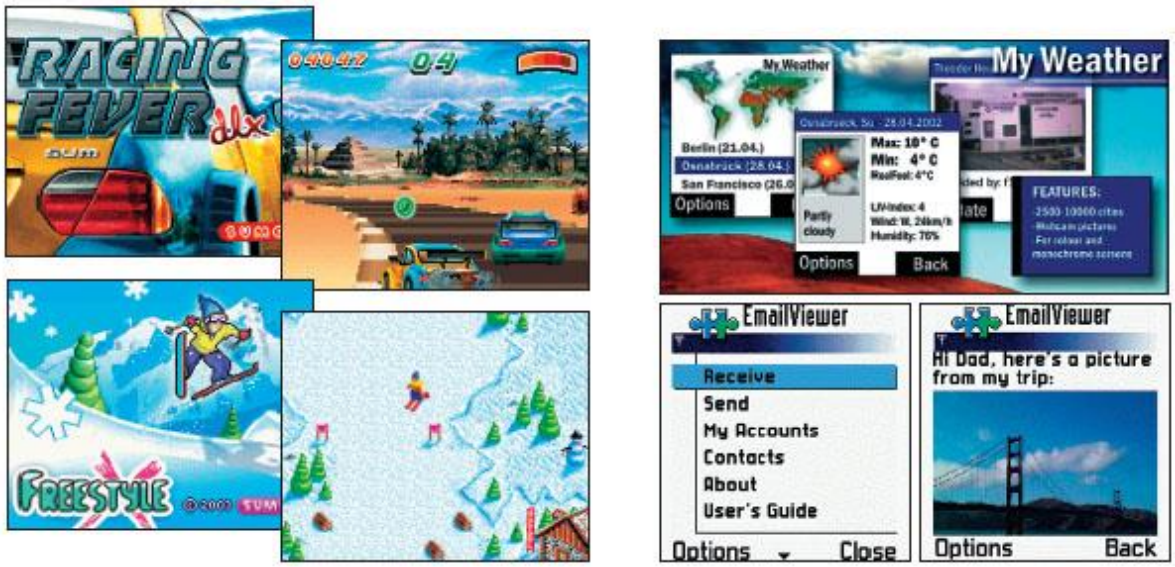


Figura 0.114: Ejemplos de aplicaciones J2ME

A finales del 2003 ya apareció el primer terminal móvil (Nokia 6600) con la segunda versión del perfil MIDP, y a partir de ahí casi todos los teléfonos ya incluían esta nueva versión. Esta especificación ha incluido grandes mejoras que van a permitir mejores comunicaciones, contenidos más multimedia y mayor seguridad. Entre las principales novedades que vamos a encontrar están las siguientes:

- Mejoras en la presentación y gestión de los gráficos y animaciones
- Inclusión de contenidos multimedia
- Mayor seguridad y más completa para las aplicaciones.
- Mejores comunicaciones incluyendo TCP/IP e iniciación de aplicaciones mediante mensajes *push*.

7.4.2 i-appli

I-appli es la tecnología para el desarrollo de aplicaciones que se ejecuten en el terminal del entorno *i-mode*. Esta tecnología se abre al público a inicios del año 2001 a partir del éxito del servicio basado

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

en aplicaciones servidoras de *i-mode*. La empresa propietaria del servicio y de la tecnología es la operadora japonesa de telefonía móvil NTT-DoCoMo.

Figura 0.115: Logotipos de NTT-DoCoMo y i-appli

I-appli surge a partir de la colaboración entre Sun Microsystems y NTT-DoCoMo. Se basa en J2ME, en la arquitectura MIDP/CLDC/KVM e incluye una serie de librerías y funcionalidades extra. En este sentido se ha notado considerablemente el conocimiento y experiencia de NTT-DoCoMo para desarrollar aplicaciones móviles y su capacidad para predecir el comportamiento y las necesidades tanto de los clientes finales como de los proveedores de contenidos.

Desde sus comienzos, se han desarrollado y publicado multitud de aplicaciones con gran variedad en su temática, como juegos, aplicaciones de información bursátil, etc..

Figura 0.116: Ejemplos de aplicaciones i-appli

De hecho la variedad de aplicaciones desarrolladas ha sido incluso mayor que en J2ME debido a que una de las mejoras introducidas en i-appli hace referencia a la seguridad. Además de la seguridad que ya se incluye en J2ME, I-appli ha introducido una serie de mecanismos de seguridad que han permitido crear todo tipo de aplicaciones *m-commerce*, cuestión todavía pendiente en J2ME. Entre estas medidas extra de seguridad destacan:

- Comunicaciones cifradas mediante SSL. I-appli dispone de mecanismos para transmitir la información cifrada punto a punto mediante claves de 40 bits o 128 bits.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Además, las aplicaciones I-appli sólo pueden conectarse al servidor desde las que fueron descargadas. Cualquier comunicación a otro servidor debe ser realizada desde el servidor original y luego reenviada al terminal.

Figura 0.117: Ejemplo de aplicaciones empresariales con i-appli

7.4.3 Brew

Brew es una tecnología desarrollada por Qualcomm. Esta empresa californiana es la responsable de la mayor parte de los chips CDMA del mercado. De hecho Brew nace como un soporte para aplicaciones nativas en esos chips a mediados del año 2001. Su mayor punto fuerte es la velocidad y tamaño de sus implementaciones al estar muy integrada directamente en el hardware del dispositivo.

Figura 0.118: Logotipos de Qualcomm y Brew

Al contrario que otras tecnologías nativas, Brew está originalmente pensado para aplicaciones escritas con C/C++. Qualcomm pone a disposición de los desarrolladores un SDK que facilita el desarrollo de aplicaciones para Brew. Además también suministra una plataforma para operadores (BDS, *BREW Distribution System*) que permite gestionar la descarga de aplicaciones de forma segura y la facturación de la misma.

A finales del año 2003, J2ME estaba distanciándose de Brew de forma evidente. Brew había crecido únicamente en los operadores con CDMA mientras que J2ME ha mantenido una estrategia más global. Los 200 modelos disponibles para Java y 120 millones de terminales vendidos no son comparables con los 100 modelos de Brew y 63 millones de terminales. Además las tendencias eran a la divergencia por lo que Qualcomm, en una decisión importante y complicada, optó por colaborar con IBM para desarrollar una máquina virtual J2ME que funcionase encima de Brew. De esa forma entrará también en el mercado de aplicaciones J2ME. Es una decisión arriesgada debida sobre todo a la percepción de estar perdiendo el tren de las aplicaciones nativas. Además ha exportado la tecnología de tal forma que no esté ligada a chips CDMA, situación que le restringía a determinadas zonas geográficas.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Respecto a los contenidos, Brew está pensado para todo tipo de aplicaciones tanto para entretenimiento, como para información o gestión empresarial.

Figura 0.119: Ejemplos de aplicaciones con Brew

Uno de los grandes puntos fuertes de Brew, ha sido la capacidad de Qualcomm de buscar alianzas estratégicas con fabricantes de terminales, operadores y proveedores de contenidos. Ha conseguido un gran número de terminales compatibles gracias a los acuerdos firmados con más de 20 fabricantes como Nokia, Motorola, LG, Sharp, Panasonic, etc. También ha conseguido implicar en el proyecto a grandes desarrolladores y proveedores de juegos como Bandai, Walt Disney, Namco, Sega Mobile, etc.

Finalmente es de destacar la estrategia que ha optado para introducirse en los diversos operadores con los que ha trabajado. En muchos de ellos ha optado por no mostrar la marca de Brew y cambiarla por el nombre del servicio de descargas que se haya querido dar en el operador en cuestión. Actualmente ya está disponible en más de 14 operadores de todo el mundo, aunque ha tenido especial penetración en los regiones con redes CDMA como América o Asia [23-25].

7.4.4 WGE

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

WGE (*Wireless Graphics Engine*) es un conjunto de APIs gráficas embebidas en las terminales diseñadas para desarrollar juegos móviles propiedad de la empresa *9 dots*. Está muy orientado a la creación de juegos para dispositivos móviles intentando parecerse lo más posible a los videojuegos de las consolas. Para facilitar el desarrollo y la migración de juegos han optado por el lenguaje de programación C/C++.

Figura 0.120: Logotipos de 9-dots y WGE

En el desarrollo de entorno de programación han hecho mucho hincapié en la espectacularidad gráfica. También han procurado soportar todas las formas de comunicación móvil para las comunicaciones con servidores, incorporando SMS, MMS, WAP o TCP/IP.

Otro punto fuerte es la seguridad que han utilizado para realizar los mecanismos de descarga e instalación de juegos en los terminales. Las aplicaciones se descargan codificadas y se decodifican en el cliente, para posteriormente borrar la aplicación descargada. De esa forma se evita poder pasar juegos y contenidos de un terminal a otro. Pero este sistema tiene un problema, que no es otro que necesita el doble del tamaño de la aplicación para poder instalarla puesto que en un momento dado va a tener la aplicación codificada y descodificada en memoria a la vez.

Si bien *9 dots* ha conseguido unos proveedores de juegos bastante importantes (E*lite*, *Digital Bridges*, etc.), el mayor problema de esta tecnología es que sólo hay un terminal disponible en el mercado (*Innostream i1000*), por lo que todavía no han conseguido introducirse de forma relevante en el mercado de las aplicaciones móviles.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Figura 0.121: Ejemplos de aplicaciones WGE

7.4.5 Mophun

Mophun es una tecnología de Synergenix para el entretenimiento móvil. Se basa en cuatro pilares fundamentales que integran el entorno Mophun:

- Para facilitar el desarrollo de juegos para dispositivos móviles incluyen un conjunto de APIs en C (Mophun API).
- También han creado un entorno de programación que incluye estas APIs y que facilita el trabajo a los desarrolladores (Mophun SDK).
- Por otro lado, a los fabricantes de terminales se les provee de una pequeña máquina virtual preparada para funcionar con los juegos (Mophun RTE, *Run Time Engine*).
- Finalmente, a los distribuidores de juegos se les habilita una herramienta para la firma y protección de juegos asociándolos a un determinado terminal. (Mophun VST, *Vendor Signing Tool*)

Figura 0.122: Logotipo de Synergenix y Mophun

Synergenix ha desarrollado su modelo de negocio con la intención de centralizar el desarrollo de juegos. En este sentido obliga a todos los desarrolladores a hacer pasar por ellos los juegos realizados para certificarlos. Esto es una práctica común en la creación de juegos para determinados entornos y

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

plataformas. Pero Synergenix va un paso más allá, incluyendo código que hace que el juego funcione en los terminales, por lo que es imposible sin su certificación instalar el juego en el dispositivo.

En el siguiente gráfico podemos ver todo el ciclo de vida de la creación y publicación de un juego.

Figura 0.123: Certificación de aplicaciones Mophun Fuente: Synergenix

Como podemos ver es obligatoria la participación de tanto Synergenix como del distribuidor de juegos para poder llevar a cabo una publicación exitosa de un juego. El único paso opcional es el primero, es decir, el envío de una idea inicial a modo de propuesta. A partir de ahí los pasos hay que seguirlos en orden. Como se puede ver Synergenix consigue su objetivo de centralizar la publicación de juegos y de paso controlar el flujo del dinero de forma directa.

El gran problema actual de Mophun es la escasa gama de terminales que lo soportan. De hecho Ericsson es el único fabricante que apoya el desarrollo de esta plataforma, siendo sus terminales T300, T310 y T610 los que soportan de forma nativa Mophun. Curiosamente, esta falta de terminales se ve suplida por un extenso catálogo de juegos de todo tipo y condición.

Figura 0.124: Ejemplo de aplicaciones Mophun

7.4.6 ExEn

ExEn (*Execution Engine*) es una tecnología desarrollada y mantenida por la empresa de entretenimiento móvil IN-FUSIO. Se basa en una máquina virtual realmente muy pequeña (menos de 100 Kbytes de ROM) y ligera (32 Kbytes de RAM). Sus principales puntos fuertes son las APIs gráficas,

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

y los mecanismos de envío de puntos y facturación por descargas de juegos y fases, por lo que como se puede ver está muy orientado al desarrollo de juegos. Estas últimas funcionalidades se basan en una plataforma que actúa como servidor. Quizás el punto más débil de este entorno de desarrollo es la capacidad de realizar juegos multijugador.

Figura 0.125: Logotipo de In-Fusio

Realmente ExEn se basa en la especificación J2ME. Su forma de programar es realmente similar y se basa en una serie de APIs que se montan encima de la máquina virtual de los teléfonos. De esta forma se aprovecha los conocimientos de los desarrolladores, facilitando la entrada de nuevos proveedores y mantiene las ventajas del modelo de máquina virtual respecto a la seguridad.

Su catálogo de juegos es relativamente variado y posee en la actualidad 23 juegos para teléfonos en blanco y negro y 5 juegos disponibles en color.

Figura 0.126: Ejemplos de aplicaciones ExEn

ExEn ya está siendo utilizada en diversos operadores móviles de toda Europa como D2 Vodafone, Orange, Telefónica Móviles, Omnitel Vodafone, etc.. También ha llegado a acuerdos con varios fabricantes de terminales para conseguir suministrar al mercado un conjunto de dispositivos que hagan llegar sus juegos a todo el mundo:

- Siemens SL42 and M50,
- Sagem 30XX range (for example 3026), My X-3, My X-5 and My G-5
- Philips Xenium 9@9, Azalys 288, Fisio 311, Fisio 620 and Fisio 825

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Trium Mars, Neptune, 110, Eclipse and 320
- Panasonic GD 67 and GD 87
- Alcatel OT 256 and OT 531
- Vitelcom TSM4
- Bird SC03

7.4.7 Sistemas operativos

Finalmente la plataforma más evidente para desarrollar aplicaciones móviles son los propios sistemas operativos de los dispositivos. Actualmente sólo los dispositivos más avanzados disponen de verdaderos sistemas operativos con APIs de programador. Pero esta situación cada vez va a ser más común.

La mayoría de sistemas operativos contienen APIs de desarrollo basadas en C/C++ que incluyen librerías gráficas y de comunicaciones entre otras. La documentación y la variedad de estas librerías depende mucho del proveedor pero no siempre es fácil conseguir realizar lo que uno desea. En cualquier caso, el desarrollo de aplicaciones sobre lo sistemas operativos siempre da más posibilidades que la creación de aplicaciones con módulos incluidos en los terminales, sobre todo en cuanto nos salimos fuera de los juegos.

Los más importantes sistemas operativos que están peleando por dominar el mercado móvil son los siguientes:

- Symbian

  Este sistema operativo es la evolución del EPOC creado por Psion. En 1998 se crea la *Joint Venture* Symbian con la participación de Ericsson, Nokia, Motorola y Psion. Posteriormente se uniría Panasonic y recientemente se ha separado Motorola puesto que ha apostado por los sistemas operativos de Microsoft.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

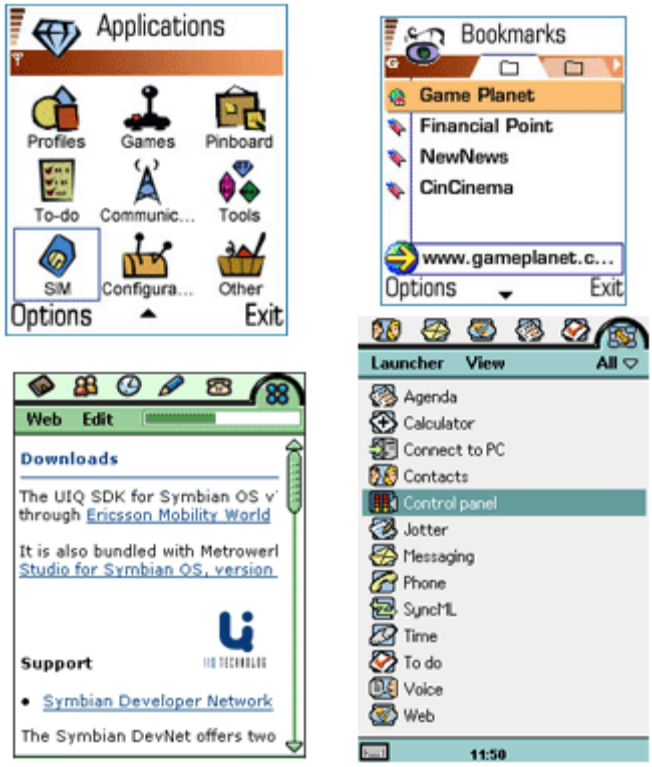


Figura 0.127: Pantallas del sistema operativo Symbian

- SmartPhone 2002

Sistema operativo de Microsoft basado en Windows CE 3.0. Sus principales apoyos han sido HTC, Samsung y Compal, aunque recientemente se ha unido Motorola. Incluye interfaces muy similares a los Windows más potentes e incluye versiones reducidas del Outlook, Explorer, Messenger y Windows Media Player.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

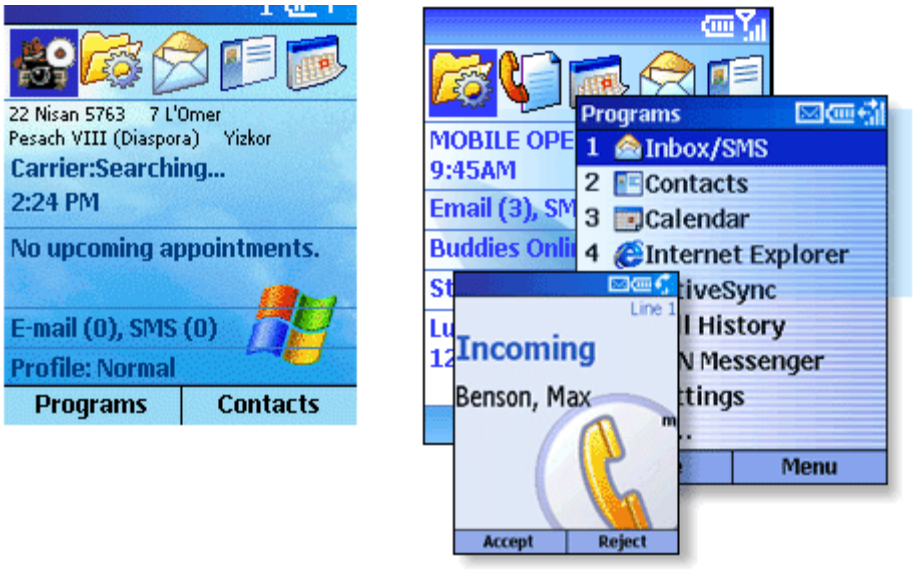


Figura 0.128: Pantallas del sistema operativo Smart Phone 2002

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

8 APLICACIONES MÓVILES

Durante las páginas anteriores hemos hecho un breve repaso de las principales tecnologías móviles para el desarrollo de aplicaciones. Evidentemente, todas esas tecnologías se deben combinar entre sí y con otras tecnologías de redes y desarrollo *software* para implementar nuevas aplicaciones orientadas a los usuarios de terminales móviles.

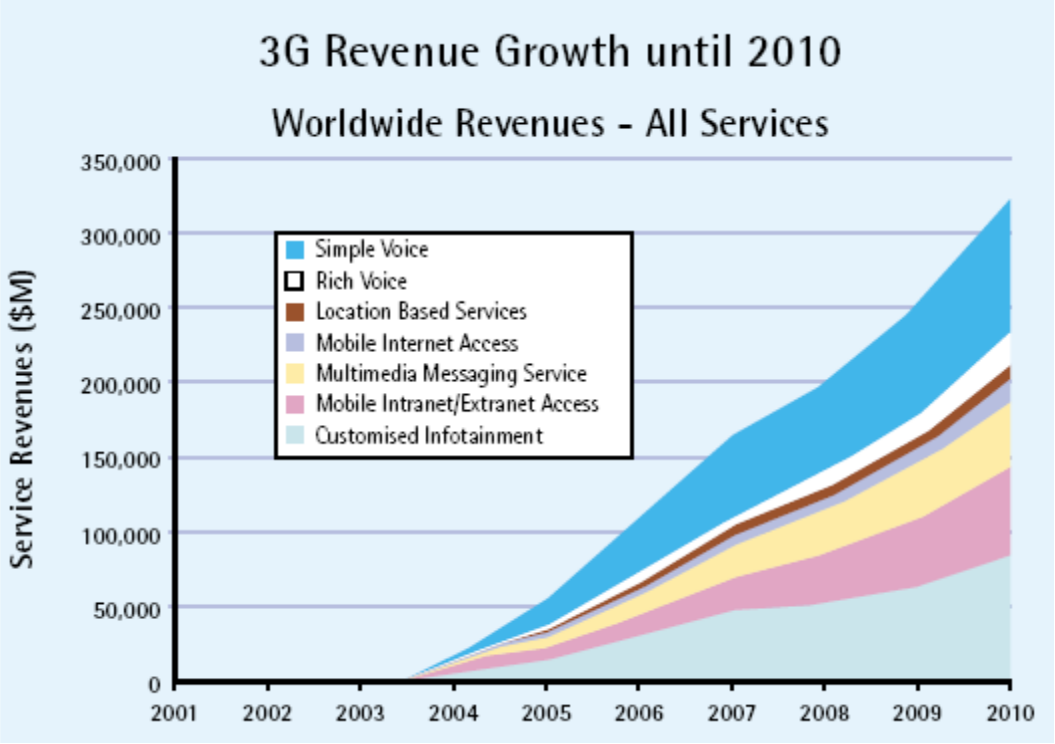


Figura 0.129: Predicción de los ingresos por servicios móviles Fuente: UMTS-Forum

Las posibilidades en este sentido son infinitas. Aún así no todo funciona comercialmente. Durante estos últimos años hemos asistido tanto al éxito de aplicaciones como a los mayores fracasos. No sólo la idea de la aplicación debe ser buena, además hay que cuidar infinidad de detalles relacionados con el correcto funcionamiento de la aplicación con la mayoría de las terminales del mercado o con la usabilidad de la aplicación con los interfaces de usuario de los terminales.

Por muy bueno y necesario que sea el concepto de una aplicación no puede tener éxito si no funciona en los dos modelos de terminal más vendidos o si para realizar cualquier acción hay que pulsar infinidad de enlaces y botones. Uno de los errores más comunes a la hora de implementar una aplicación es tratar de copiar la funcionalidad y la estructura de la aplicación WEB equivalente en la aplicación destinada para terminales móviles, ya sea con páginas WML, una aplicación nativa o un interfaz de comandos SMS [26-29].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Cada vez es más evidente que de alguna forma la mayoría de dispositivos personales, como reproductores de música, consolas portátiles, agendas o teléfonos, van a acabar convergiendo a un dispositivo con un sistema operativo, pantalla táctil y comunicaciones. En realidad va a ser un pequeño ordenador de bolsillo multifunción. En ese contexto las aplicaciones que se pueden realizar para este tipo de dispositivos son muy variadas. Desde servicios de comunicaciones instantáneas hasta control remoto de máquinas, pasando por compra de entradas o videojuegos.

Figura 0.130: Convergencia de dispositivos móviles

Esta evolución de los dispositivos va a ir en paralelo con la evolución de las tecnologías de desarrollo de software para móviles, y con la mejora considerable de las redes de comunicaciones móviles. Por esta razón, y después de haber visto las tecnologías actuales para el desarrollo de aplicaciones móviles, en este apartado vamos a tratar de clasificar y analizar los diferentes tipos de aplicaciones y servicios posibles en el mercado de las tecnologías móviles.

Durante el próximo análisis no debemos perder de vista que todas las aplicaciones que pensemos se pueden desarrollar con variedad de tecnologías, incluso simultáneamente (aplicaciones multiacceso), aunque muchas veces hay una que especialmente se adapta bien. Por ejemplo, si pensamos en una aplicación de comercio electrónico móvil (*m-commerce*) como la compra de entradas de cine, se puede realizar tanto por páginas WML como por una aplicación nativa desarrollada en J2ME o Mophun, incluso mediante por SMS se podría llegar a crear un interfaz de acceso. Evidentemente, aunque posible esta última opción no sería la mejor, aunque las otras dos podrían convivir perfectamente tratando de llegar al mayor número de terminales y por lo tanto de usuarios.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Antes de entrar en los diferentes tipos de aplicaciones vamos a ver cómo ha cambiado el concepto de aplicación según han ido apareciendo nuevas redes de comunicaciones.

8.1 Servicios 3G

El concepto de servicio sobre el que vamos a basarnos para abordar este apartado y los siguientes hace referencia al conjunto de funcionalidades que ofrece la red para usuarios finales o proveedores de aplicaciones para utilizarlos en caso de los primeros y para crear aplicaciones en el caso de los segundos.

Trataremos este concepto en el contexto de las redes de 2G y de 3G estableciendo las diferentes aproximaciones que se realizan en los dos casos para luego entrar en nuevas arquitecturas de servicios previstas en las redes 3G.

8.1.1 Concepto de servicios en 2G y 3G

Según hemos definido el concepto de servicio, en las redes 1G sólo existía uno, la voz. En cambio con la aparición de las redes 2G la idea se vuelve más amplia. En principio los servicios eran voz, fax, datos y servicios suplementarios (desvío de llamadas, mensajes cortos, etc..). Posteriormente aparecen funcionalidades de red inteligente a través de la arquitectura CAMEL (*Customized Applications for Mobile network Enhanced Logia*) permitiendo crear servicios más avanzados como localización, buzón de voz, etc..

El problema de este enfoque era que CAMEL está pensado para una red en la que el servicio estrella es la voz. En cambio UMTS y las tecnologías 3G en general dan por lo menos una importancia equivalente a las aplicaciones de datos, por lo que CAMEL no estaba preparado para lo que se avecinaba. Para más problemas, CAMEL se diseñó con la idea de que los servicios iban a ser implementados totalmente por el operador, dentro de este esquema no entraba el concepto de proveedor de aplicaciones que utilizan los servicios provistos por la red para crear aplicaciones para el usuario final.

El concepto de servicio en las redes 3G cambia substancialmente, se deja de estandarizar los servicios finales y se concluye que lo que debe quedar definido son las interfaces para ofrecerlos y poder crear aplicaciones encima. Además se hace un gran énfasis en la personalización de los servicios y las aplicaciones así como la movilidad de estos en redes extranjeras.

8.1.2 Arquitectura de servicios: OSA

La arquitectura de servicios prevista para UMTS es la denominada Open Service Architecture (OSA). En esta arquitectura se separan y se especifican claramente los servicios de los elementos de red encargados de implementarlos. Es decir, es una forma de publicar las facilidades de la red móvil UMTS para que puedan ser fácilmente utilizadas, ocultando las características y problemáticas de implementación de los distintos proveedores de redes de telecomunicación. Para realizar esta división se establecen dos conceptos nuevos:

- SCF (*Service Capability Features*), funcionalidades de las capacidades del servicio que es accesible a través de una interfaz estandarizada. Permiten a las aplicaciones (externas o no) utilizar las capacidades de servicio de la red de forma segura y abierta, independizándolas de

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

la tecnología de la red. Estas SCFs incluyen funcionalidades como autenticación, autorización, registro, consulta de capacidades de servicio, localización, mensajería, etc..

- SCS (*Service Capability Server*) entidades lógicas en las que residen las SCFs, y que proporcionan los interfaces OSA. A su vez se comunican con las capacidades del servicio de la red que no son sino la mensajería, la identificación, WAP, localización, etc..

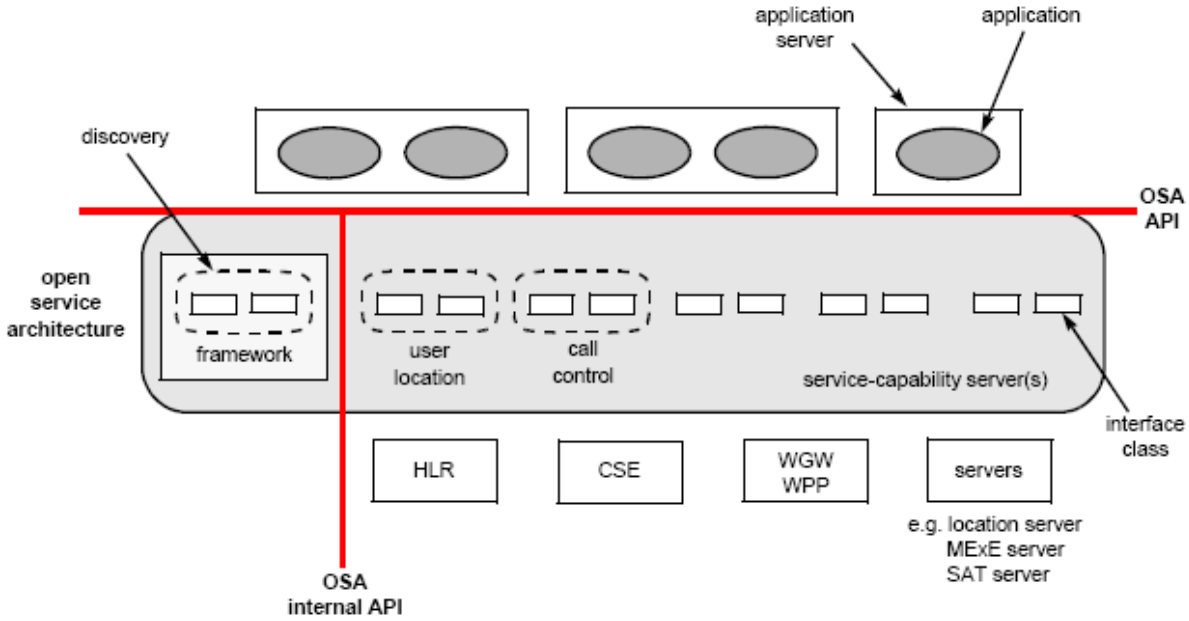


Figura 0.131: Diagrama de la arquitectura OSA Fuente: Grupo 3GPP

Actualmente hay dos grandes implementaciones de OSA en marcha, Parlay y JAIN. Esta segunda está coordinada por SUN Microsystems y se basa en Java. Estos estándares e implementaciones permitirán la aparición de numerosos proveedores de aplicaciones móviles puesto que podrán desarrollar sus aplicaciones contra interfaces independizando no sólo el operador final sino incluso la tecnología de red móvil usada.

8.1.3 Servicios personales: VHE

Ya hemos visto cómo facilitar el desarrollo de aplicaciones móviles independientes de la implementación de los servicios en las redes móviles. Pero también comentamos que había dos componentes más en el concepto de servicios 3G, la personalización y la movilidad de los servicios. Para alcanzar estos objetivos se introduce la idea de *Virtual Home Environment* (VHE).

Mediante este concepto el usuario tendrá un entorno personal de ejecución de servicios que describe el modo de percibir e interactuar con los servicios a los que esté subscrito. Esto se consigue en UMTS mediante el perfil de usuario, que de hecho pueden ser varios por usuario, según la hora, el día de la semana o su elección. Además este perfil se mantendrá aunque el usuario esté en una red ajena o no use el terminal habitual con el que suele trabajar. Evidentemente, la forma de identificar al usuario y mantener la seguridad es a través de la USIM.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

8.2 Aplicaciones de mensajería

Después del éxito de todas las aplicaciones basadas sobre SMS, MMS es la tecnología sobre la que muchos ponen todas sus esperanzas. Los servicios basados en mensajería han sido los más exitosos de los últimos años. No sólo en el entorno móvil, sino que también en el Internet fijo con los *e-mails* o la mensajería instantánea. De hecho se espera que estas dos aplicaciones se introduzcan tarde o temprano en el entorno móvil de forma definitiva, el consorcio OMA ya tiene abiertos grupos de trabajo al respecto [30-34].

Las aplicaciones de mensajería se pueden clasificar independientemente de la tecnología usada en cuatro grandes grupos, desde los simples envíos de mensajes a aplicaciones más complejas de entretenimiento. La disposición funcional de esta tipología se puede situar en forma de anillos concéntricos que indican que cada capa exterior contiene a las interiores extendiéndolas.

Figura 0.132: Tipología de las aplicaciones de mensajería móviles

- Los mensajes P2P (*Peer to Peer*) son los mensajes que se envían los usuarios unos a otros sin intervención de aplicaciones más avanzadas. Es el servicio de mensajería más básico y en el caso de arquitecturas *Store&Forward* como MMS y SMS se puede descomponer en dos servicios, uno de envío de mensajes MO (*Mobile Originated*) y otro de envío de mensajes MT (*Mobile Terminated*).

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Las aplicaciones básicas de almacenamiento y envío suplantan a uno de los extremos en el anterior servicio. Podemos dividir a estas aplicaciones según el extremo suplantado:
    - Aplicaciones de envío: Estas aplicaciones permiten desde WEB, WAP, J2ME o cualquier otro mecanismo enviar mensajes MT. Las aplicaciones más características de este tipo son por un lado los compositores de mensajes, y por otro las aplicaciones de descarga de contenidos como logos, melodías, etc..
    - Aplicaciones de almacenamiento: Estas aplicaciones recibir mensajes MO, el usuario puede enviar sus mensajes desde el móvil a estas aplicaciones que lo que harán será almacenar los contenidos de tal forma que puedan ser recuperados posteriormente. Así tendremos un álbum multimedia en el caso de MMS o sencillamente un repositorio de mensajes en el caso de SMS. Si queremos imaginarnos una aplicación de este tipo con PTT, podríamos pensar en una agenda de notas vocales que podríamos recuperar, por ejemplo, mediante un acceso WAP o mensajes SMS.

  Este tipo de aplicaciones son muy utilizadas por los operadores y cada uno de ellos tienen su propios servicios de envío y almacenamiento, sobre todo en el caso MMS. Además combinando correctamente estas tecnologías se puede dar un soporte para envío y recepción de MMS para terminales que no los soporten.

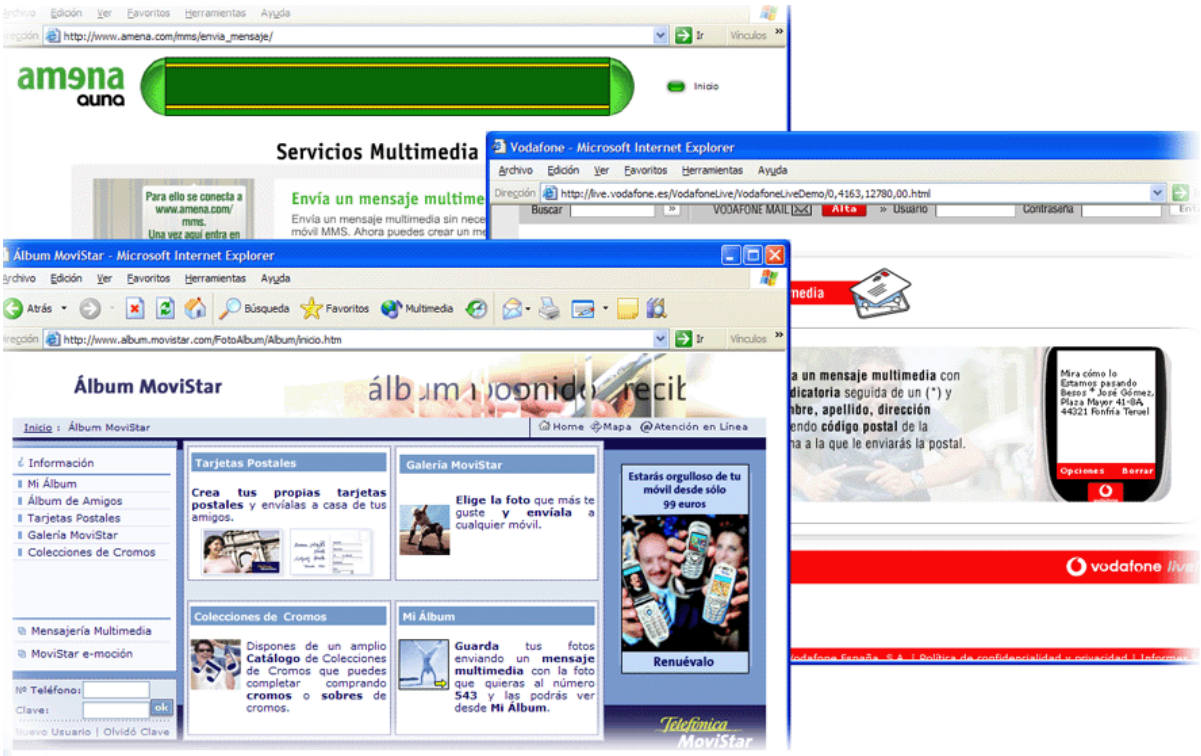


Figura 0.133: Ejemplos de aplicaciones WEB de mensajería

- Las aplicaciones avanzadas de mensajería son tantas como la imaginación pueda generar. En este punto vamos a ver las más típicas pero las posibilidades son infinitas. Antes de entrar a

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

citarlas, debemos tener claro la diferencia entre aplicaciones *Push* y *Pull*. Las aplicaciones *Pull* son aquellas iniciadas por el usuario, es decir, es él que mediante una acción sobre la aplicación desencadena el envío del mensaje que le va a llegar con el contenido pedido. Por el contrario, en las aplicaciones *Push*, el usuario recibe un mensaje sin haberlo pedido, ya sea porque se subscribió a esa alerta, porque entró en una zona geográfica especial, etc.

- Aplicaciones de avisos

Dentro de las aplicaciones de avisos se pueden englobar la mayoría de aplicaciones basadas en el modelo *Push*. Dentro de estas aplicaciones podemos encontrar:

- Notificaciones del operador con información sobre el saldo, con información sobre promociones, etc..
- Agenda con alarmas, combinando la mensajería con otra tecnología podremos tener una agenda en la que además nos avise mediante mensajes que tenemos una reunión o una cita.
- Mensajes con publicidad de terceras empresas. Esta es una práctica que no está muy extendida pero que seguramente empezará pronto. En el entorno móvil, el gran aliciente sería mezclarla con la localización. De esta forma el público objetivo de la campaña sería mucho más restringido y útil. Por ejemplo en el momento que entrásemos en un centro comercial nos podría llegar un mensaje de una tienda ofreciéndonos cualquier producto (con su foto si es MMS) a un precio de oferta si enseñamos ese mensaje en la compra.
- Alertas. Cualquier información que varíe con el tiempo puede ser sujeta de realizar alertas sobre ella. El usuario recibirá un mensaje con la nueva información justo en el momento en el que ésta haya cambiado o periódicamente. Ejemplos de informaciones susceptibles de ser utilizadas para hacer alertas, son la información bursátil, el tráfico de una ciudad, el tiempo, los acontecimientos deportivos, programas de televisión como Gran Hermano o OT, etc..

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Figura 0.134: Ejemplo de alerta MMS Fuente: Comunicaciones I+D

- Aplicaciones de información

Además de las alertas, el usuario podría tratar de obtener todo tipo de información mediante el envío de un mensaje. En este caso ya estaríamos hablando de esquemas *Pull*. Todo tipo de información se puede utilizar para estas aplicaciones, desde titulares y noticias, resultados deportivos o de azar, tiempo, tráfico, etc..

- Aplicaciones de juegos y entretenimiento

En este punto las posibilidades se multiplican hasta donde quiera la imaginación. Desde juegos, hasta tiras cómicas, pasando por encuestas o tests, hasta llegar seguramente a la aplicación más exitosa de todas, el chat. También entrarían aquí el envío de postales físicas mediante el envío de un MMS, fotomontajes, caricaturas, análisis de la firma, felicitaciones, etc.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Figura 0.135: Trailer enviado por MMS

Todas las aplicaciones que hemos visto hasta ahora se han orientado sobre todo a las tecnologías de mensajería que actualmente más están funcionando como SMS, MMS, WAPPush e incluso USSD. Evidentemente, aunque muchas aplicaciones que se nos ocurran se puedan realizar con todas las tecnologías, siempre habrá una o dos más adecuadas que serán las que deberemos usar. Aún así, si es posible siempre es bueno utilizar por lo menos SMS para poder llegar a todos los usuarios y no restringirnos a los terminales más avanzados y modernos.

8.3 Aplicaciones de acceso a Internet/Intranet

Una gran parte de las aplicaciones que se pueden realizar en los teléfonos móviles es las que van a permitir el acceso a páginas WAP para obtener información de Internet. La primera reflexión y seguramente la más importante aquí será si es realmente posible trasladar los contenidos de la WEB al entorno móvil y si realmente es asimilable el concepto de Internet al entorno móvil [35-40].

Evidentemente hay bastantes puntos que nos hacen pensar que realmente hay demasiadas diferencias para poder suponer viable el Internet que conocemos en los móviles actuales:

- Las velocidades de conexión son todavía mucho menores en el terreno móvil. Si bien es cierto que esto va a cambiar con la aparición de las redes 3G, lo más seguro que el precio de los datos transmitidos sea demasiado alto como para poder realizar la misma navegación desde el terminal fijo y desde el móvil.
- Las capacidades multimedia son realmente diferentes. En un PC se puede tener fácilmente pantallas con una resolución de 1024x768 píxeles con millones de colores. En un terminal

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

muy avanzado encontramos pantallas de 160x120 píxeles con 4000 colores. En cuanto al sonido también las diferencias son grandes, tanto por altavoces como por tarjetas de sonido. Y siempre comparando con terminales móviles de gama alta.

- Finalmente la interfaz de usuario es realmente diferente. Mientras que en un PC podemos encontrar un teclado normal y un ratón, en un terminal encontraremos un teclado numérico y 2 ó 3 botones para navegar. Este punto es el único en el que el futuro puede favorecer a los terminales móviles, pues es fácilmente posible que en breve espacio de tiempo la mayoría incorporen pantallas táctiles con lo que la situación cambiaría considerablemente.

Todos estos puntos nos pueden hacer pensar que las dificultades de trasladar Internet al entorno móvil son muy altas. Y es verdad. Pero lo más importante es saber y conocer estos problemas. Aún así la mayoría de contenidos y aplicaciones de Internet son trasladables al mundo móvil, si bien hay que tener muy claro que no se deben copiar los interfaces, sino que se deben sólo trasladar los contenidos y las funcionalidades amoldándolos a las características de los terminales.

Vamos a analizar en los siguientes apartados las posibilidades de los terminales móviles para acceder a contenidos de Internet y a Intranets corporativas, es decir contenidos y aplicaciones del sistema de información de su empresa.

8.3.1 Internet móvil

Como ya hemos visto, casi todas las aplicaciones y contenidos de Internet se pueden trasladar al mundo móvil adaptando los interfaces de usuario para las características especiales de los terminales. Pero no hemos hablado de qué pueden aportar las redes y los terminales móviles a este acceso. Hay una serie de ventajas exclusivas del concepto de Internet móvil que son realmente importantes y que van a dar a un proveedor de aplicaciones grandes posibilidades:

- El usuario siempre está identificado por el operador. Puesto que la relación terminal-usuario es de uno a uno, cualquier aplicación podrá autenticar y registrar a un usuario automáticamente siempre que llegue a un acuerdo con el operador.
- El operador también puede dar información de localización geográfica al proveedor.
- El proveedor de aplicaciones tiene multitud posibilidades tecnológicas para realizar su aplicación o servir sus contenidos. Desde portales de voz, pasando por envíos de mensajes con alertas hasta aplicaciones nativas para el acceso a funcionalidades avanzadas.
- Finalmente, el usuario tiene movilidad, es decir puede conectarse a Internet dónde y cuándo quiera.

Estas son ventajas importantes para un proveedor de aplicaciones. Pero realmente quién tiene más posibilidades es el operador de telefonía móvil. Aunque existen varios modelos de negocio posibles (ver la siguiente figura), la mayoría de los operadores han optado por diversificarse verticalmente entrando también en el negocio de la provisión de aplicaciones y portales móviles.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

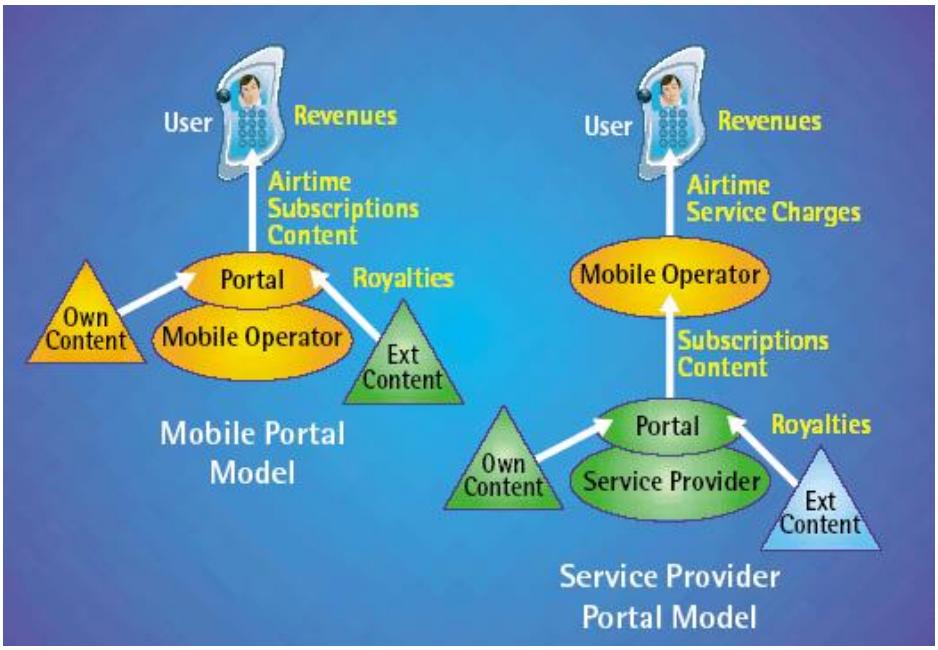


Figura 0.136: Modelos de negocio para portales móviles Fuente: UMTS-Forum

Muchos proveedores de aplicaciones han creado portales para la telefonía móvil, pero también casi todos los operadores han creado portales de entrada con todo tipo de contenidos al estilo de Yahoo, Lycos o MSN en Internet. Y además con grandes ventajas sobre los proveedores de aplicaciones clásicos:

- El terminal viene ya configurado para poder acceder a Internet mediante el operador. Esto supone que el usuario sin tocar nada acabará en el portal del operador.
- Puesto que el operador ya cobra a sus clientes por otros servicios, le resultaría muy favorable incluir en la factura los servicios *Premium* que utilice el usuario. Después ya saldaría cuentas con el proveedor de la aplicación y de esa forma controla el flujo de dinero, cuestión en absoluto trivial.
- Además el esquema de red de un operador móvil que utilice una pasarela WAP permitirá tener contenidos exclusivos (por ejemplo el portal) que sólo podrán se accesibles desde terminales de su red. No sólo eso, también podría restringir a determinados usuarios el acceso a cualquier contenido.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

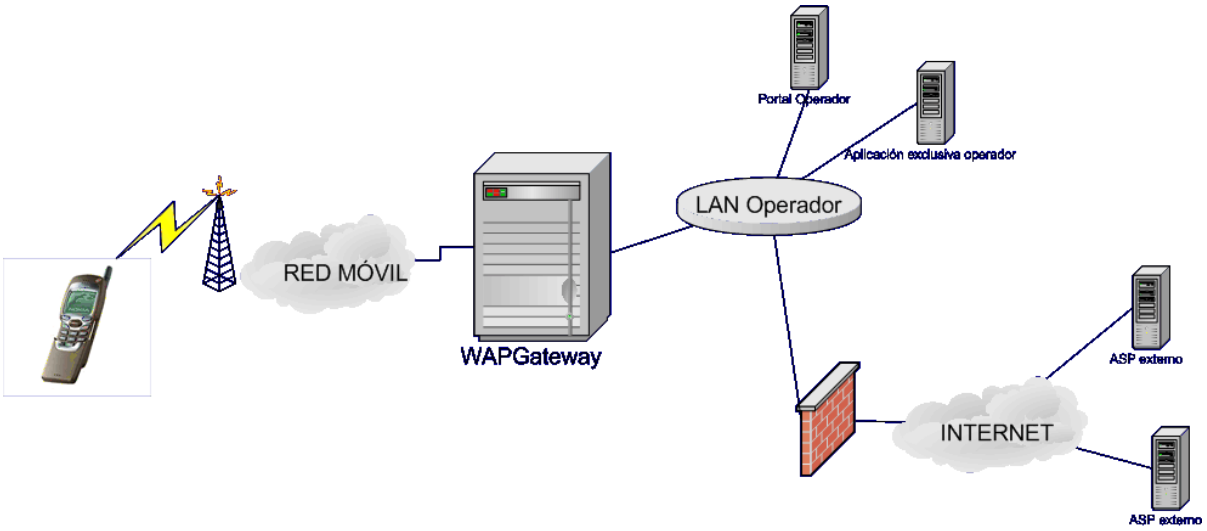


Figura 0.137: Esquema de provisión de contenidos móviles para operadores

- Puesto que el terminal es asociable a la persona que lo utiliza es realmente sencillo obtener los gustos del usuario mediante el tipo de accesos que hace al portal, permitiendo una gran personalización de los contenidos.

Como vemos, aunque realmente Internet tal y como lo conocemos tiene grandes capacidades, existen determinadas ventajas comparativas que son las que hay que explotar para conseguir que la gente use su dispositivo móvil para conectarse a Internet. Sin apoyarse en estas diferencias el acceso a Internet desde el móvil está avocado al fracaso como ha pasado hasta hace poco.

8.3.2 Redes móviles corporativas

Una aplicación importante para los terminales móviles es el acceso a los sistemas de información empresariales o intranets corporativas. Estos portales de información tendrían las mismas capacidades que ya tienen los sistemas de información actuales como agenda, mail, gestión de flotas, gestión de inventarios, etc. pero con las ventajas del entorno móvil:

- Movilidad, los empleados podrán acceder a las aplicaciones del sistema de información incluso estando fuera de la oficina.
- Posibilidades de localización, permitiendo aplicaciones avanzadas de gestión de flotas.
- Aplicaciones de notificación mediante mensajería móvil, permitiendo avisos instantáneos para aplicaciones de inventarios, correos, alarmas de la agenda, etc..

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Figura 0.138: Prototipo de terminal empresarial Fuente: Ericsson

8.4 Aplicaciones con localización

La localización geográfica de usuarios en tiempo real es sin lugar a dudas uno de los elementos con los que las aplicaciones móviles se pueden distinguir y especializarse. Si a la localización le añadimos información georeferenciada como calles, o restaurantes las oportunidades que se abren son realmente importantes. Este tipo de aplicaciones que utilizan localización suelen denominarse *Location Based Services* (LBS).

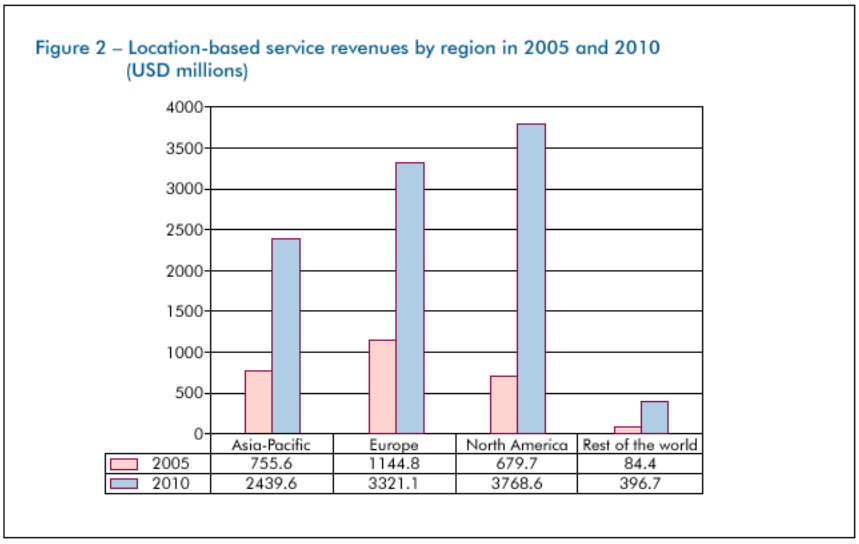


Figura 0.139: Ingresos previstos para de los servicios móviles con localización Fuente: UMTS-Forum

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Dentro de las aplicaciones que podamos imaginar con localización se pueden hacer cuatro grupos principales:

- Aplicaciones de información

La aplicación más importante que suministra información utilizando la localización del usuario son las páginas amarillas. Mediante este servicio el usuario puede averiguar que cines, restaurantes, gasolineras, etc. están cerca suyo y cuál es la distancia. Otra aplicación característica es la provisión de mapas a partir del lugar donde está el usuario.

Figura 0.140: Ejemplo de aplicaciones con localización

Por poner un último ejemplo, también se podría pensar en una aplicación que suministre información turística según la posición del usuario en la ciudad mediante mensajes MMS.

- Aplicaciones de seguridad

En este punto podemos incluir los números 112 de emergencia o números de ayuda a colectivos de riesgo como mujeres maltratadas. Estos números permiten avisar de situaciones de riesgo que están sucediendo a nuestro alrededor, si añadimos la localización a la llamada podemos filtrar falsas alarmas y realizar procedimientos más rápidos y efectivos.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

También se pueden incluir en este apartado todo tipo de equipos que incorporando terminales móviles permiten situarlos para realizar funciones de seguridad como localización de vehículos robados.

- Juegos y aplicaciones de entretenimiento

Dentro de los juegos, las posibilidades que brinda la localización permiten añadir movilidad a las aplicaciones. Detectando la posición del usuario podemos obligarle a tener que ir realmente a los sitios para realizar acciones. De esta forma se pueden realizar gymkhanas o juegos cómo la búsqueda del tesoro.

También se puede utilizar la localización para realizar cálculos de distancias o direcciones. Así se pueden pensar en juegos donde haya que disparar bombas a otros adversarios indicando la dirección del envío y la fuerza.

Finalmente también se puede utilizar la localización para saber que jugadores están cerca del usuario. Con este mecanismo se pueden realizar chats con localización o juegos de lucha en los que tienes que estar cerca para combatir.

- Aplicaciones de seguimiento

Por aplicaciones de seguimiento entendemos aquellas en las que mediante la introducción de un terminal móvil en algo podemos seguir el camino de este elemento y comprobar su trayectoria en el tiempo. Las aplicaciones más evidentes se refieren a servicios de gestión de flotas para empresas, pero también nos podemos imaginar servicios de seguimiento para adolescente, mayores con problemas de Alheimer, etc.

Todas estas aplicaciones deben cumplir con las regulaciones europeas en materia de privacidad. Estas directivas (95/46/EC y 97/66/EC) introducen los siguientes principios:

- Se debe obtener el consentimiento explícito del usuario localizado.
- Se debe proveer mecanismos al usuario para restringir o anular la localización sobre el mismo.
- Se debe proveer una completa información sobre el uso y el almacenamiento de la información
- La información sólo se debe usar para el uso para el que fue obtenida
- Se debe borrar la información personal una vez usada o se debe hacerla anónima.
- No se debe transmitir la información a terceros sin el consentimiento del usuario.

8.5 Entretenimiento móvil

Cuando hablamos del entretenimiento móvil podemos incluir, los juegos, la descarga de contenidos como logos, melodías, la reproducción de música y cualquier tipo de aplicación que en general permita a los usuarios distraerse en su tiempo libre. Si bien la descarga de tonos y logos y el chat ha sido hasta ahora las aplicaciones estrella, en el futuro se espera que la descarga de juegos y los juegos online lleguen a igualarlas e incluso a superarlas en facturación y uso [41-43].

Realmente no hay mucho que decir en torno a la descarga de tonos y logos o sobre el chat. Son aplicaciones de sobra conocidas y sin capacidad para grandes variaciones. Donde se puede experimentar más es el mercado de los juegos con el móvil. No hay que olvidar que actualmente la industria

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

del videojuego mueve anualmente más dinero que el cine o la música en España. Si a esto le sumamos el éxito de plataformas de juegos portátiles como la Gameboy de Nintendo (100 millones de consolas vendidas), se puede esperar un considerable uso del móvil como videoconsola.

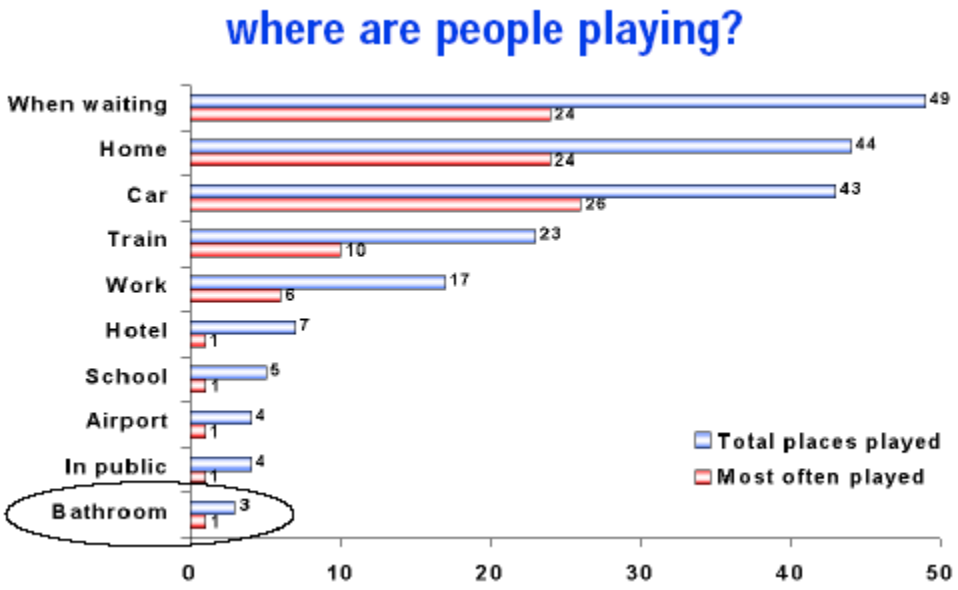


Figura 0.141: Estadística de lugares donde juega la gente Fuente: Nokia

De hecho desde hace años el móvil se ha venido usando como plataforma de juegos mediante las aplicaciones que los fabricantes embebían en sus terminales. Seguramente el que más éxito tuvo con este tipo de juegos fue Nokia que después de sacar el juego *Snake* continuó creando juegos para sus terminales a un ritmo considerable.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Figura 0.142: Ejemplos de juegos embebidos en los terminales

Esos juegos eran muy sencillos y no tenían en absoluto conectividad por lo que no existía modelo de negocio alguno, sencillamente servían para que los usuarios compraran más terminales del modelo con mejores juegos. Siemens intentó incluir en sus teléfonos juegos de este tipo, pero con gráficos mejores y comunicaciones SMS para juegos multijugador. El mecanismo que se incluyó en juegos para el Siemens C45 era sencillo y se basaba en los juegos que ya funcionaban en WEB mediante e-mails y páginas WEB. El usuario escogía una serie de movimientos (por ejemplo, de lucha) y los enviaba a un contrincante, éste respondía con otra serie. Los terminales interpretaban los SMS y mostraban al usuario una animación con el combate. Un juego sencillo, pero con mucha capacidad de generar una masa importante de usuarios [44-48].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

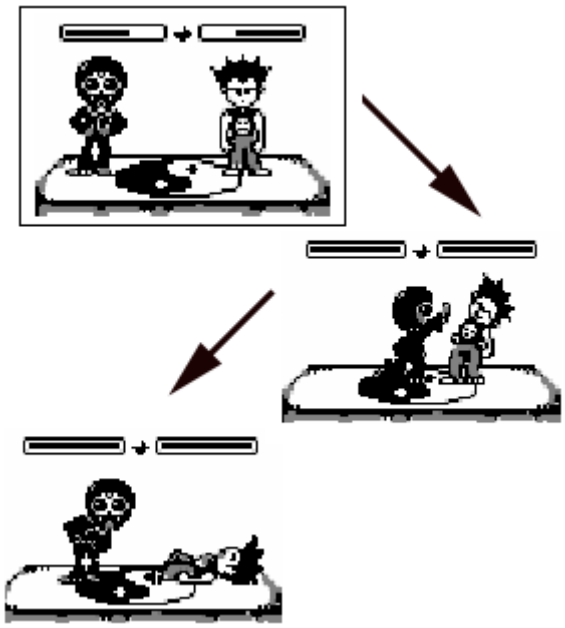


Figura 0.143: Ejemplo de juego por turnos basado en SMS

Posteriormente han ido apareciendo juegos y plataformas sobre todo tipo de tecnologías como por ejemplo WAP o MMS. Pero es en las aplicaciones que se ejecutan en el teléfono donde los juegos tienen mayores posibilidades, principalmente a que son programas que necesitan de gran espectacularidad gráfica y gran interactividad con el usuario. Como hemos podido ver anteriormente la mayoría de las tecnologías pensadas para ejecutar programas en el terminal han sido concebidas para realizar juegos.

Actualmente estamos superando una serie de fases en el desarrollo de juegos nativos (sobre todo J2ME) para acabar teniendo juegos multijugador al estilo de los que hoy funcionan en Internet:

- Juegos monojugador

Estos juegos son los que actualmente más abundan. Suelen se juegos cuya idea original surge a partir de los juegos equivalentes en las consolas y que únicamente permiten jugar al usuario del terminal puesto que no tienen posibilidades de conectividad.

- Juegos monojugador con envío de puntuaciones

Es la evolución natural de los anteriores, sencillamente se añade la posibilidad de mandar las puntuaciones a un servidor para su publicación por Internet o para repartir premios.

- Juegos multijugador

Los juegos multijugador permiten que dos o más usuarios puedan jugar unos contra otros o en equipos al mismo juego. Inicialmente estos juegos han sido aplicaciones basadas en turnos como juegos de mesa, de cartas o de rol.

- Juegos multijugador en tiempo real

Finalmente, con las mejoras en las redes han ido apareciendo juegos multijugador en tiempo real. Este tipo de juegos permiten jugar a distintos usuarios a aplicaciones en las que cada movimiento

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

se ejecuta en el instante en el que se ordena. Juegos de este tipo son los deportivos, plataformas, etc..

8.6 M-commerce

El comercio electrónico empieza a ser una realidad poco a poco. Más le está costando arrancar al comercio electrónico con el móvil o *m-commerce*. Su retraso está siendo provocado por la lenta y costosa penetración de los servicios de Internet en el móvil. Aún así el futuro que han previsto todas las consultoras y operadoras es realmente brillante.

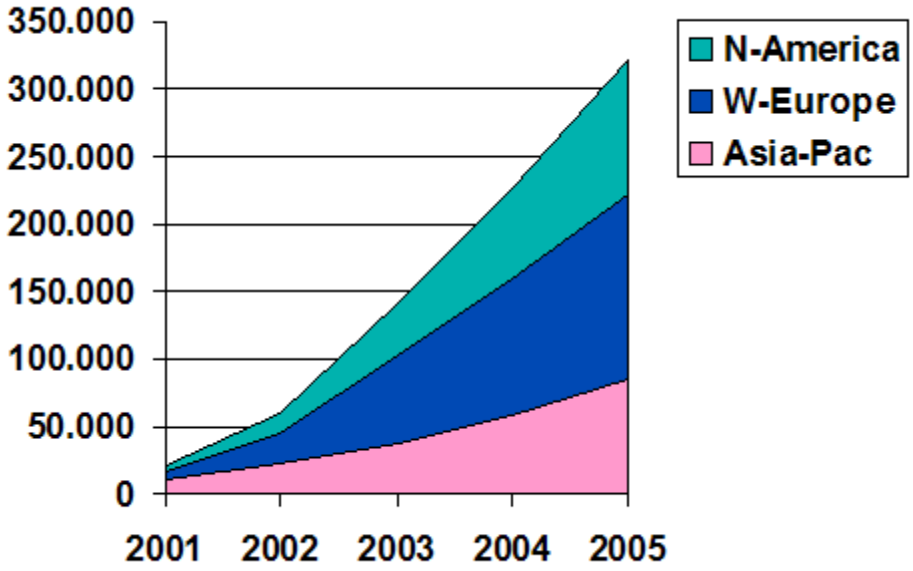


Figura 0.144: Miles de usuarios previstos para el *m-commerce* Fuente: T-Motion

Estas optimistas estimaciones nacen de las ventajas del *m-commerce* sobre el comercio electrónico convencional:

- Mucha mayor facilidad para cobrar tanto pagos grandes como pequeños. Al disponer el operador de una factura con el cliente se puede utilizar ésta para incluir los pagos realizados para comprar productos y servicios con el móvil.
- La movilidad que permiten las comunicaciones móviles permitirán a los usuarios utilizarlo como forma de pago en tiendas y comercios.
- La estrecha relación del terminal con su propietario añadirá además una importante capacidad para personalizar los portales y sitios de *m-commerce*.

Las posibilidades del *m-commerce* son realmente variadas, desde compras de latas de refresco, pasando por compra-venta de acciones o compra de juegos descargables, hasta llegar a la compra de entradas de espectáculos (*ticketing*) o sitios de apuestas con el móvil. Vamos a ver ahora los dos grandes grupos de aplicaciones que destacan en el comercio con el móvil.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Figura 0.145: Ticket electrónico con un MMS Fuente: Comunicaciones I+D nº21

8.6.1 Banca móvil

Dentro del mundo de la banca, las entidades que se apoyan en los nuevos canales basados en el teléfono y en Internet están ganando importantes cuotas de mercado. Su ventaja son los precios competitivos que ofertan debido a la reducción de costes y la posibilidad de operar a cualquier hora sin necesidad de ir a una sucursal.

La banca en el móvil sigue el mismo concepto. Con una ventaja sobre Internet, se puede operar y realizar consultas en cualquier sitio en cualquier momento. Esta posibilidad, que en otros aspectos del comercio no es tan importante, en la banca es muy relevante primero porque siempre pueden surgir necesidades de dinero y segundo porque muchas veces las inversiones hay que hacerlas en el momento oportuno sin dilación.

Mención aparte en este apartado merece las operaciones de inversión a través de portales móviles. Como decíamos antes, las compras y ventas de productos financieros deben realizarse en el momento oportuno, porque minutos más tarde el precio y por lo tanto la rentabilidad pueden haber cambiado mucho. Es por esta razón que una de las aplicaciones que se esperan tengan más éxito son los servicios de compra-venta de acciones y otros productos financieros.

Figura 0.146: Ejemplo de aplicación de negociación de valores bursátiles

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

8.6.2 El móvil como medio de pago

El terminal móvil tiene unas enormes oportunidades para convertirse en un importante medio de pago para todos los consumidores. Para empezar, es un medio muy extendido en la población, además es un dispositivo que la mayoría de gente lleva consigo casi permanentemente. Finalmente, la factura del móvil ya existe y por lo tanto el paso de cobrar el dinero podría estar resuelto.

Esto no quiere decir que al final no se opte por cobrar directa e instantáneamente contra un banco, aunque se use el móvil para ello. Las razones que pueden provocar esta situación son dos, primero que los bancos no van a dejar fácilmente que los operadores móviles se hagan con un pastel que ahora mismo controlan ellos con las tarjetas de débito y crédito. Segundo, el pago con la factura móvil (no con el prepago) tiene el inconveniente que funciona como un crédito en el sentido en el que se paga todas las compras a final de mes, con lo que el operador entraría en un terreno que no es el suyo y que no conoce.

Actualmente en España hay tres iniciativas de pagos por el móvil, Paybox (participada por Deutsche Bank), CaixaMovil y Movilpay (apoyada por Telefónica, Vodafone, BBVA y la mayoría de bancos). Ninguna de las tres ha tenido el éxito esperado aún habiéndose lanzado alguna de ellas ya en el año 2001. El mayor problema que se han enfrentado estas plataformas ha sido la dificultad de crear una red de comercios que sea lo suficientemente grande como para atraer a los usuarios.

8.7 Aplicaciones M2M

La aplicación M2M (*Machina to Machina*) son aplicaciones que se basan en las comunicaciones entre máquinas. Su uso se basa en la monitorización o gestión de máquinas que no tenían por sí mismas conectividad y que su acceso no siempre resulta sencillo por su entorno natural de operación (movilidad, zonas remotas o inaccesibles, etc..)

Las aplicaciones basadas en M2M son casi ilimitadas. Por el momento podemos imaginar un impacto aproximado en los principales sectores en los que puede ser aplicado:

- La automoción

  Instalando módulos con conectividad en los coches son varias las aplicaciones que ya empiezan ha ser desarrolladas y probadas. Por un lado equipos de diagnosis de averías serían realmente prácticos, tanto para las marcas, como para el cliente como para los talleres. Permitiría pasar de un mantenimiento correctivo a uno preventivo con todo lo que eso lleva.

  Otra posibilidad es la realización de cajas negras para el análisis de la conducción y recogida de datos de accidentes. Ya hay compañías aseguradoras que están apostando por estos mecanismos para agilizar los partes de accidentes y para calcular la prima de riesgo de los conductores según su forma de conducir.

  Siguiendo esta línea, algunos gobiernos nórdicos están planteándose cobrar los carburantes según el despilfarro que hagan de ellos los conductores. Conducciones económicas proporcionarán precios bajos o descuentos en las gasolineras, y al contrario para los conductores más derrochadores.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- *Vending*

El sector de las máquinas expendedoras supone un importante negocio, y incluyendo el pago por el móvil supondría primero que habría menos dinero en las máquinas (menos posibilidad de robo y menos necesidad de recoger ese dinero) y segundo que los usuarios no tendrían que disponer de dinero en efectivo.

Además, las posibilidades no se quedan en el cobro, la gestión remota de las máquinas permitiría conocer de antemano la avería que podría sufrir la máquina, así como una gestión de inventario mucho más eficiente y sencilla.

- Empresas *commodities*

Este tipo de empresas, como las que proveen agua, luz, gas o teléfono, tienen dos grandes aplicaciones M2M pensadas. Por un lado este tipo de empresas tienen en muchos casos centros remotos de operación como centrales hidroeléctricas, centrales de conmutación, etc.. Este tipo de centros serían fácilmente gestionables de forma remota. Y no sólo eso, además uno de los problemas de este tipo de empresas es la gestión correcta de todas las llaves de todas estos edificios o casetas. Mediante cerraduras conectadas a equipos M2M se podría dar acceso a empleados mediante mensajes de texto, sólo habría que comprobar que el número remitente este autorizado y abrir la puerta.

Además, este tipo de empresas obtendrían un importante ahorro en el caso de empezar a obtener la información de los contadores de forma remota.

- Otros sectores empresariales y domésticos

Hay infinidad de sectores en los que se pueden aplicar las soluciones M2M. Por hacer una breve enumeración a continuación aparecen los más importantes:

  - Electrodomésticos
  - Seguridad
  - Control de tráfico
  - Bienes de equipo
  - Etc.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

9 Páginas web de consulta

- www.gsmworld.com
- http://www.onforum.com/tutorials/gsm/
- http://www.wmlclub.com/
- http://www.cellular.co.za/
- http://www.auladatos.movistar.com/Aula-de-Datos/Tutoriales-y-Documentacion/
- http://www.mobilepositioning.com/
- http://www.tomiahonen.com/3gmap.html
- http://telecom.iespana.es/telecom/telef/gsm-info.htm#fr
- http://www.3g-generation.com/
- http://www.moconews.net/
- www.thefeature.com
- http://www.cellular-news.com/
- http://www.openmobilealliance.org/
- http://www.forum.nokia.com/main.html
- http://www.openwave.com/
- http://www.motocoder.com
- http://www.umts-forum.org
- http://www.3gpp.org/
- http://www.itu.int
- http://www.etsi.org/
- http://java.sun.com/j2me/
- http://developers.sun.com/techtopics/mobility/
- http://www.nttdocomo.com/
- www.qualcomm.com/brew/
- www.9dots.net/about/wge.html
- www.mophun.com/
- http://www.infusio.com/

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Hernando Rábanos, J.M (2000) Comunicaciones Móviles GSM. Edita Fundación Airtel. Madrid, España
2. Hernando Rábanos, J.M (2000) GPRS Tecnología Servicios Y Negocios. Edita Telefónica Móviles España. Madrid, España
3. Hernando Rábanos, J.M (2001) Comunicaciones Móviles De Tercera Generación UMTS 2 Vols. Edita Telefónica Móviles España. Madrid, España
4. Huidobro Moya, J.M. (2002) Comunicaciones Móviles. Edita Paraninfo. Madrid, España.
5. Adam Macintosh, Ming Feisiyau, Mohammed Ghavami (2014). Impact of the Mobility Models, Route and Link connectivity on the performance of Position based routing protocols. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
6. Ana Silva, Tiago Oliveira, José Neves, Paulo Novais (2016). Treating Colon Cancer Survivability Prediction as a Classification Problem. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1
7. Anna Vilaro, Pilar Orero (2013). User-centric cognitive assessment. Evaluation of attention in special working centres: from paper to Kinect. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
8. Casado-Vara, R., & Corchado, J. (2019). Distributed e-health wide-world accounting ledger via blockchain. Journal of Intelligent & Fuzzy Systems, 36(3), 2381-2386.
9. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
10. Casado-Vara, R., de la Prieta, F., Prieto, J., & Corchado, J. M. (2018, November). Blockchain framework for IoT data quality via edge computing. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (pp. 19-24). ACM.
11. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
12. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
13. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
14. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
15. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
16. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
17. Chamoso, P., Rivas, A., Martín-Limorti, J. J., & Rodríguez, S. (2018). A Hash Based Image Matching Algorithm for Social Networks. In Advances in Intelligent Systems and Computing (Vol. 619, pp. 183–190). https://doi.org/10.1007/978-3-319-61578-3_18
18. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
19. Choon, Y. W., Mohamad, M. S., Deris, S., Illias, R. M., Chong, C. K., Chai, L. E., … Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. PLoS ONE, 9(7). https://doi.org/10.1371/journal.pone.0102744
20. Corchado, J. M., & Fyfe, C. (1999). Unsupervised neural method for temperature forecasting. Artificial Intelligence in Engineering, 13(4), 351–357. https://doi.org/10.1016/S0954-1810(99)00007-2
21. Corchado, J., Fyfe, C., & Lees, B. (1998). Unsupervised learning for financial forecasting. In Proceedings of the IEEE/IAFE/INFORMS 1998 Conference on Computational Intelligence for Financial Engineering (CIFEr) (Cat. No.98TH8367) (pp. 259–263). https://doi.org/10.1109/CIFER.1998.690316

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

22. Costa, Â., Novais, P., Corchado, J. M., & Neves, J. (2012). Increased performance and better patient attendance in an hospital with the use of smart agendas. Logic Journal of the IGPL, 20(4), 689–698. https://doi.org/10.1093/jigpal/jzr021
23. Daniel Fuentes, Rosalía Laza, Antonio Pereira (2013). Intelligent Devices in Rural Wireless Networks. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
24. Di Mascio, T., Vittorini, P., Gennari, R., Melonio, A., De La Prieta, F., & Alrifai, M. (2012, July). The Learners' User Classes in the TERENCE Adaptive Learning System. In 2012 IEEE 12th International Conference on Advanced Learning Technologies (pp. 572-576). IEEE.
25. Fyfe, C., & Corchado, J. (2002). A comparison of Kernel methods for instantiating case based reasoning systems. Advanced Engineering Informatics, 16(3), 165–178. https://doi.org/10.1016/S1474-0346(02)00008-3
26. Fyfe, C., & Corchado, J. M. (2001). Automating the construction of CBR systems using kernel methods. International Journal of Intelligent Systems, 16(4), 571–586. https://doi.org/10.1002/int.1024
27. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. Expert Systems with Applications, 41(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003
28. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
29. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
30. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
31. Hoon Ko, Kita Bae, Goreti Marreiros, Haengkon Kim, Hyun Yoe, Carlos Ramos (2014). A Study on the Key Management Strategy for Wireless Sensor Networks. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
32. Hugo López-Fernández, Miguel Reboiro-Jato, José A. Pérez Rodríguez, Florentino Fdez-Riverola, Daniel Glez-Peña (2016). The Artificial Intelligence Workbench: a retrospective review. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1
33. Javier Gómez, Xavier Alamán, Germán Montoro, Juan C. Torrado, Adalberto Plaza (2013). AmICog – mobile technologies to assist people with cognitive disabilities in the work place. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
34. Jose-Luis Jiménez-García, David Baselga-Masia, Jose-Luis Poza-Luján, Eduardo Munera, Juan-Luis Posadas-Yagüe, José-Enrique Simó-Ten (2014). Smart device definition and application on embedded system: performance and optimization on a RGBD sensor. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
35. Leonardo Ochoa-Aday, Cristina Cervelló-Pastor, Adriana Fernández-Fernández (2016). Discovering the Network Topology: An Efficient Approach for SDN. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
36. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
37. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). Random finite set-based Bayesian filters using magnitude-adaptive target birth intensity. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637788&partnerID=40&md5=bd8602d6146b014266cf07dc35a681e0
38. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. Applied Mathematics and Computation, 270, 756–767. https://doi.org/10.1016/j.amc.2015.08.059
39. Mar López, Juanita Pedraza, Javier Carbó, José M. Molina (2014). The awareness of Privacy issues in Ambient Intelligence. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
40. Muñoz, M., Rodríguez, M., Rodríguez, M. E., & Rodríguez, S. (2012). Genetic evaluation of the class III dentofacial in rural and urban Spanish population by AI techniques. Advances in Intelligent and Soft Computing (Vol. 151 AISC). https://doi.org/10.1007/978-3-642-28765-7_49

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

41. Paula Andrea Rodríguez Marín, Mauricio Giraldo, Valentina Tabares, Néstor Duque, Demetrio Ovalle (2016). Educational Resources Recommendation System for a heterogeneous Student Group. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 3
42. Ricardo Faia, Tiago Pinto, Zita Vale (2016). Dynamic Fuzzy Clustering Method for Decision Support in Electricity Markets Negotiation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1
43. Rodriguez-Fernandez J., Pinto T., Silva F., Praça I., Vale Z., Corchado J.M. (2018) Reputation Computational Model to Support Electricity Market Players Energy Contracts Negotiation. In: Bajo J. et al. (eds) Highlights of Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection. PAAMS 2018. Communications in Computer and Information Science, vol 887. Springer, Cham
44. Rodríguez, S., De La Prieta, F., Tapia, D. I., & Corchado, J. M. (2010). Agents and computer vision for processing stereoscopic images. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6077 LNAI). https://doi.org/10.1007/978-3-642-13803-4_12
45. Rodríguez, S., Tapia, D. I., Sanz, E., Zato, C., De La Prieta, F., & Gil, O. (2010). Cloud computing integrated into service-oriented multi-agent architecture. IFIP Advances in Information and Communication Technology (Vol. 322 AICT). https://doi.org/10.1007/978-3-642-14341-0_29
46. Sittón, I., & Rodríguez, S. (2017). Pattern Extraction for the Design of Predictive Models in Industry 4.0. In International Conference on Practical Applications of Agents and Multi-Agent Systems (pp. 258–261).
47. Víctor Corcoba Magaña, Mario Muñoz Organero (2014). Reducing stress and fuel consumption providing road information. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
48. Víctor Parra, Vivian López, Mohd Saberi Mohamad (2014). A multiagent system to assist elder people by TV communication. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Desarrollo de aplicaciones para sistemas móviles con tecnología Symbian y Mobile Widgets

Javier García Puga [1]

[1] Telefónica Investigación y Desarrollo, Spain
jgarcia_puga@telefonica.es

Resumen: En este capitulo trabajaremos sobre el concepto de widget, que es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine. Esta tecnología facilita el acceso a funciones frecuentemente usadas y proveer de información visual. Sin embargo, los widgets pueden hacer todo lo que la imaginación desee e interactuar con servicios e información distribuida en Internet; pueden ser vistosos relojes en pantalla, notas, calculadoras, calendarios, agendas, juegos, ventanas con información del clima en su ciudad, etcétera". En este capítulo se presenta esta tecnología y se muestran algunos ejemplos. Una vez analizados algunos ejemplos se mostrarán los diferentes tipos de widgets, clasificados en función del entorno de ejecución: widgets de escritorio (PC), widgets Web, mobile widgets, etc.

Palabras clave: Symbian; Mobile Widgets

Abstract. In this chapter we will work on the concept of widget, which is a small application or program, usually presented in files or small files that are executed by a widget engine or Widget Engine. This technology facilitates access to frequently used functions and provides visual information. However, widgets can do whatever the imagination desires and interact with services and information distributed on the Internet; they can be eye-catching on-screen clocks, notes, calculators, calendars, agendas, games, windows with information about the weather in your city, etc.". This chapter introduces this technology and shows some examples. Once some examples have been analysed, the different types of widgets will be shown, classified according to the execution environment: desktop widgets (PC), Web widgets, mobile widgets, etc.

Keywords: Symbian; Mobile Widget

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

1. Introdución

Según la wikipedia, un "*widget"* es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine. Entre sus objetivos están los de dar fácil acceso a funciones frecuentemente usadas y proveer de información visual. Sin embargo, los widgets pueden hacer todo lo que la imaginación desee e interactuar con servicios e información distribuida en Internet; pueden ser vistosos relojes en pantalla, notas, calculadoras, calendarios, agendas, juegos, ventanas con información del clima en su ciudad, etcétera" [1-9].

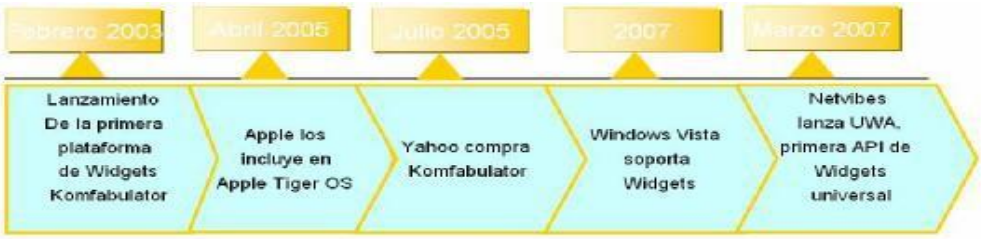
También son conocidos por términos equivalentes como "page components", "startlets", "gadgets", "web badges" o "modules", aunque la denominación más conocida es la de Widgets.

Existen diferentes tipos de widgets si se atiende al entorno donde éstos son ejecutados: widgets de escritorio (PC), widgets Web, mobile widgets, etc.

1.1 Características principales

- Son aplicaciones vistosas, ligeras y sencillas diseñadas para una finalidad concreta.
- Su aspecto es el de una aplicación independiente integrada dentro del escritorio
- Muy usables
- Gran nivel de acceso a los recursos de la máquina
- Fácil instalación / desinstalación
- Gran comunidad de desarrolladores
- Enmarcados dentro del movimiento Web 2.0 (AJAX, ...)

1.2 Evolución



Este tema se centrará en el desarrollo de widgets para terminales móviles (mobile widgets), especialmente en la plataforma Widsets y Nokia Web RunTime (WRT).

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

2. Widsets

2.1 Introducción

WidSets es una plataforma de widgets terminales móviles, basada en Java MIDP 2.0 y desarrollada por una *spin off* de Nokia.
La plataforma permite el desarrollo de widgets, utilizando el lenguaje de scripting Helium, que se ejecutarán en una máquina virtual (Widsets VM), la cual reside en el terminal móvil. Esta máquina virtual puede ser instalada en cualquier terminal que soporte Java MIDP 2.0 [10-15].

Las características principales de esta plataforma son las siguientes:
- No permite acceso a recursos del terminal (cámara, llamadas,…)
- Exige descarga de la plataforma (Widsets VM) y registro de usuarios en el terminal.
- El descubrimiento de widgets se puede realizar a través de web y se descarga en el terminal mediante sincronización.
- Permite creación de widgets por parte de usuarios mediante plantillas y el lenguaje de scripting Helium.
- Permite a los usuarios monitorizar el tráfico de que generan sus terminales.

Antes de empezar a enumerar los componentes de los widgets de esta plataforma conviene definir algunos conceptos:
- Gestor de WidSets (WidSets manager): Se trata de una aplicación web que proporciona a los usuarios de Widsets funcionalidades como la sincronización de widgets, gestión de la cuenta de usuario, monitorización del tráfico, etc.
- Biblioteca de WidSets (WidSets Library): Biblioteca publicada en un servidor donde residen los widgets una vez publicados.

2.2 omponentes de los widgets

Un widget está formado normalmente por los siguientes componentes:
- Fichero widget.xml. Este fichero contiene los detalles específicos del widget: meta-datos, servicios utilizados por el widget, hoja de estilo, recursos y layout de las vistas.
- Fichero script .he. Contiene los scripts de Helium utilizados para dotar de funcionalidad al widget.
- Ficheros de recursos:
  - web_icon.png: Imagen de 60*40px que será utilizada en la librería de widgets o en el estante (*shelf*) del Widgets Manager
    -

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Figura 2.1 web_icon.png

- web_minimized.png: Esta imagen se utiliza para mostrar la vista minimizada del widget dentro del WidSets Manager de la web. El ancho de la imagen debe ser de 110 píxeles. La altura es flexible.

Figura 2.2 web_minimized.png

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- web_maximized.png: Esta imagen se utiliza para mostrar la vista maximizada del widget dentro del WidSets Manager de la web. El tamaño de la imagen debe ser de 176x208px.

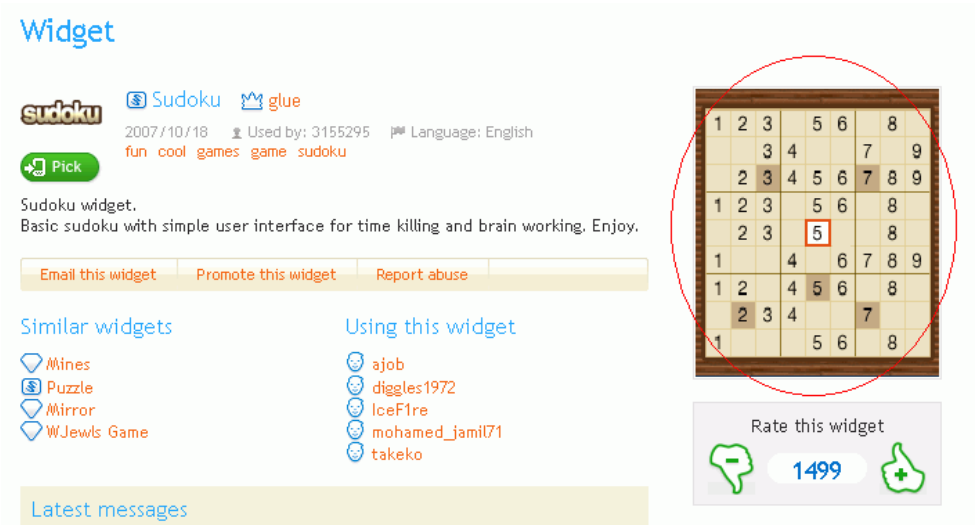


Figura 2.3 web_ maximized.png

2.2.1 Fichero widget.xml

El fichero *widget.xml* contiene todas las propiedades que definen el widget: nombre, versión, recursos, etc.
Este fichero debe comenzar con las siguientes líneas:

```
<?xml version="1.0" encoding="UTF-8"?>
  <widgetspec_version=”2.0”>
```

A continuación se debe incluir el elemento <info>, el cual contiene los siguientes elementos:

- name: Nombre del widget
- version: Versión
- author: Nombre del desarrollador.
- shortdescription: Descripción corta
- clientversion: (opcional) Versión del cliente que se necesita para ejecutar el widget
- longdescription: (opcional) Descripción corta
- tags: (opcional) Tags que describen el widget.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<info>
   <name>HolaMundo</name>
   <version>1.0</version>
   <author>Javier</author>
   <shortdescription>Descripcion   corta</shortdescription>
   <clientversion>0.97</clientversion>
   <longdescription> Descripción
      larga
   </longdescription>
   <tags>deportesnoticias</tags>
</info>
```

A parte del elemento *info* hay otros que simplemente se enumerarán a continuación:

- Help: Definir la ayuda del widget
- Services: Permiten realizar llamadas servicios ubicados en el servidor.
- Parameters: Los parámetros sirven para almacenar datos de configuración dinámicos.
- Variables: Variables del widget. Pueden hacer referencia a parámetros utilizando la sintaxis ${}
- Resources: Define los recursos del widget: imágenes, código, hoja de estilos, etc.
- Layout: Define las vistas y layout del widget.

2.3 Introducción al lenguaje de scripting Helium

Esta sección se enumerará de forma muy concisa las principales funcionalidades del lenguaje Helium. Como se observará, se trata de un lenguaje similar a Java.

Para más información http://dev.widsets.com/apidocs/

2.3.1 Elementos básicos

Comentarios

```
/* bloque */
// línea
```

Literales

```
”lorem ipsum” /* string */ ’\n’     /*
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
Caracter        */
123456          /* enter */
0777777         /* octal */
0xcafebabe      /* hexa */
0b101010101     /* binario*/
```

Control de flujo

```
if   else   while   do   for   switch   case   default
return   break   continue   foreach
```

Operadores

```
+    -    *    /    %    &    |    ^    <    >    !    ~
<<   >>   +=   -=   *=   /=   %=   &=   |=   ^=   <<=
<=   >=   ==   !=   &    ||   ++   --
instanceof      new ?:
```

Tipos primitivos

```
boolean /* true/false */
int         /* con signo 32 bit */ long
            /* sin signo 64 bit */
```

Keywords

```
class const false null return struct true void
```

2.3.2 Funciones más comunes

Las funcionalidades de los widgets pueden ser divididas en las siguientes categorías:

Interfaces relacionados con el ciclo de vida del widget

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

void startWidget(); //Llamada cuando el usuario inicia WidSets.
void stopWidget(); //Llamada cuando el usuario elimina o recarga el widget, o cuando se cierra WidSets .
Shell openWidget(); // Cuando se selecciona un widget para abrirlo (modo maximizado)
void closeWidget(); // Cuando se sale del modo maximizado del widget

Interfaces relacionados con menú de control del widget

MenuItem getSoftKey(Shell shell, Component focused, int key); //Llamada cuando el usuario pulsa una softKey
Menu getMenu(Shell shell, Component focused); //Llamada cuando el usuario pulsa una softKey asociada a un menú

Interfaces relacionados con las acciones de los usuarios

void actionPerformed(Shell shell, Component source, int action); //Llamada cuando el usuario pulsa una softKey
boolean keyAction(Component source, int op, int code); //Llamada cuando el usuario pulsa una tecla

Interface para la creación de la vista del widget.

Flow createView(String name, Object context)
Component createElement(String viewId, String elementId,
Style style, Object context);

Función de callback para eventos del timer

void timerEvent(Timer timer);

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Funciones de callback para eventos del servidor de comunicaciones

void onSuccess(Object state, Value returnValue); void onFailure(Object state, String errorMessage);

2.3.3 Reglas del lenguaje

- Todas las funciones y variables deben estar implementadas siempre dentro de una clase.

```
class
{
   int globalVar = 100;

   void firstFunction()
   {
      int localVar = 0;
   }
}

//No obstante, pueden existir funciones dentro de otra función:


Flow createAddressView(String name, String addr, String city)
{
   Flow flow = new Flow(getStyle("address.view"));

   add("Name", name);
   add("Address", addr);
   add("City", city);

   void add(String name, String value)
   {
      String text = format("%s: %s", name, value); Label label =
      new Label("address.field"), text); label.setPreferred-
      Width(-100); label.setFlags(VISIBLE|LINEFEED);
      flow.add(label);
   }

   return flow;
}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Todas las variables deben ser inicializadas y tienen alcance delimitado por el bloque en la que están contenidas.

```
voidfirstFunction()
{
   int j; /* error */
   {
      int c = 0;
   }
   {
      int d = c; /* error, 'c' no es visible aqui*/
   }
}
```

- Type-casting explícito

```
boolean flag = false; Text
displayText;

voidsomeFunction()
{
   int zeroOrOne = int(flag); /* casting boolean a int */
   ...
   String str = String(zeroOrOne); /* casting int a string */ displayText.setText(str);
}
```

2.3.4 *Limitaciones de Helium*

De momento, no muchas funcionalidades del terminal son accesibles a través del lenguaje Helium. A continuación se enumeran algunos ejemplos:

Acceso a memoria persistente del terminal.

Aunque no se puede acceder al sistema de ficheros del teléfono, se puede almacenar datos en memoria persistente utilizando la clase Store.

Acceso a Galería/Imágenes

N/
D

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Acceso a contactos

N/
D

Bluetooth

N/
D

Envío recepción de SMS

N/
D

Realización de llamadas

N/
D

Lanzar el navegador.

De momento no está disponible, aunque es posible que en un futuro se implemente esta opción.

Reproducción de sonidos

Sí, utilizando la clase Player aunque no hay soporte para streaming.

Reproducción de videos

N/
D

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

2.4 Instalación de la plataforma de WidSets en el terminal

Para la instalación de la plataforma en un móvil es necesario realizarlo a través de su sitio web y realizar la descarga inicial de cómo mínimo un Widget, para que este instale de forma automática el motor de WidSets [16-20]. Es necesario realizar la creación de una cuenta de usuario con la que autenticarse en posteriores usos.

A continuación se indican los pasos a seguir para la instalación de la plataforma:

- Para instalar WidSets por primera vez en un móvil hay que visitar la web, www.widsets.com e ir a la sección "Get Started".
- A continuación se deberá marcar la casilla Pick que hay en cada Widget disponible (se marcarán las casillas Pick de tantos como se quiera instalar) y a continuación pulsar “Download the client and picked widgets”.
- Se completan los datos de registro y se eligen cualquiera de las opciones para transferir al móvil. La opción más cómoda es la de recibir la URL de descarga por SMS, para ello será necesario introducir el número de teléfono.
- En este momento un SMS es enviado al número de móvil introducido, con un link para descargar el motor de WidSets. En la página web se muestra un mensaje que lo confirma.
- Una vez que se recibe el SMS en el móvil hay que abrir el link de descarga.
- Una vez que la instalación ha finalizado, para ejecutar la aplicación hay que ir a la carpeta donde se ha guardado la aplicación, que dependiendo del teléfono ésta será Aplicaciones, Mis Cosas, etc.
- Pulsar sobre ”WidSets” y responder sí a la pregunta ¿Permitir a la aplicación WidSets usar la red y enviar y recibir datos?
- Al abrir la aplicación se informará de que es necesario registrarse en www.widsets.com, para ello hay que hacer lo siguiente:
- En el correo recibido pulsar en “Validate now”. Se valida el correo y se informa de que los procedimientos en caso de olvido de la password.
- Una vez completado el proceso de instalación de la plataforma, si se quieren añadir nuevos Widgets se puede realizar activando el Widget de Sistema desde el móvil. Se selecciona “library”, se marca la casilla Pick del Widget que se quiera instalar y seleccionar “Back”. Automáticamente el nuevo Widget se cargará en el móvil.
- La carga de nuevos Widgets en el móvil también se puede realizar desde la web www.widsets.com.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

2.5 WidSets SDK

El SDK para WidSets, también conocido como DevKit, contiene todas las herramientas necesarias para poder desarrollar Widgets basados en esta tecnología [21-25]. El DevKit contiene un compilador de Helium, un emulador, ejemplos y documentación; y puede ser descargado desde la siguiente URL: http://dev.widsets.com/downloads/devkit.zip

El DevKit se utiliza una interfaz basada en comandos. Su sintaxis es la siguiente:

devkit <options> command <arguments>

Antes de poder utilizar el SDK, será necesario crear una cuenta en el site *http://dev.widsets.com*, puesto que la obtenida anteriormente en www.widsets.com no será válida.

Una vez creada la cuenta, se procederá a logarse en el sistema utilizando el siguiente comando:

devkit login <usuario> <clave>

Para lanzar el emulador se utilizará el comando:

devkit run

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Figura 2.4 Emulador WidSets

Una vez tengamos desarrollada la aplicación, se podrá subir al site *dev.widsets.com* y verla en el emulador utilizando el comando: devkit upload <directorio proyecto>

2.6 Información de Referencia

- http://dev.widsets.com/wiki/Getting_started
- http://developer.widsets.com/apidocs

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

3. Nokia Web Run Time

3.1 Introducción

La plataforma S60, a partir de su 3ª edición FP 2, incorpora soporte para widgets por medio del Web Run-Time (WRT).
Los widgets se desarrollan de forma similar a los widgets web, utilizando JavaScript, CSS y HTML. WRT soporta las APIs de JavaScript de acuerdo con la 3ª edición de la especificación ECMA-262. También soporta JavaScript DOM y XMLHttpRequest. A parte del estándar, WRT proporciona APIs para permitir el acceso a varias propiedades y funcionalidades del terminal móvil [26-30].

Un widget debe estar compuesto como mínimo por dos ficheros, los cuales deben estar situados en el directorio raíz del proyecto. Además puede contener otros ficheros opcionales como un icono, ficheros css, ficheros javascript y otras imágenes.
La siguiente tabla resume los ficheros que componen un widget así como su ubicación relativa dentro del directorio donde se ubica el proyecto:

Fichero	Obl./Opc.	Ruta	Descripción
info.plist	Obligatorio	/	Fichero XML que contiene las propiedades del widget
[nombre].html	Obligatorio	/	Fichero HTML que contiene la estructura del widget. El nombre de este fichero debe venir especificado en el archivo info.plist
icon.png	Opcional	/	
*.css	Opcional	/ o cualquier subdirectorio	
*.js	Opcional	/ o cualquier subdirectorio	
*.jpg/bmp/gif/png	Opcional	/ o cualquier subdirectorio	

3.2 Fichero info.plist

Se trata de un fichero XML en el cual se definen las propiedades y las opciones de configuración del widget.
A continuación se muestra un ejemplo de la estructura de este fichero así como una tabla con la descripción de cada una de las propiedades que lo componen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Nokia//DTD PLIST 1.0//EN"
"http://www.nokia.com/NOKIA_COM_1/DTDs/plist-1.0.dtd">
<plist version="1.0">
<dict
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

```
        </dict>
<key>DisplayName</key>
<string>Nombre del Widget</string>
<key>Identifier</key>
<string>com.company.widget.name</string>
<key>MainHTML</key>
<string>Main.html</string>
<key>Version</key>
<string>1.0</string>
<key>AllowNetworkAccess</key>
<false/>
        </plist>
```

Nombre	Tipo	Obl./Opc.	Descripción
DisplayName	String	Obligatorio	Nombre que aparecerá en el menú de aplicaciones
Identifier	String	Obligatorio	Identificador único del widget
MainHTML	String	Obligatorio	Nombre de la página HTML que será cargada cuando se ejecute el widget.
Version	String	Opcional	Versión del widget
AllowNetworkAccess	Bolean	Opcional	Indica si el widget requiere acceso a la red.

3.3 Fichero HTML

Un widget debe tener un fichero HTML en el cual se definen su apariencia y funcionalidad. Se recomienda seguir la especificación HTML 4.01 a la hora de implementar este fichero.

El nombre de este fichero deberá coincidir con lo especificado en la propiedad MainHTML del fichero info.plist.

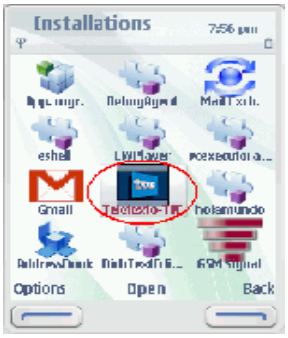
Durante la ejecución del widget, este documento HTML no podrá ser sustituido por otro archivo HTML. Si un widget necesita mostrar una página HTML adicional, esta página deberá mostrarse directamente desde el browser web. Para ello se podrá utilizar la función *widget.openURL()* analizada más adelante.

3.4 icon.png

El fichero icon.png será el que utilice el sistema operativo a la hora de mostrar el widget dentro del menú de aplicaciones. Deberá ser una imagen de 88x88 píxeles con formato png.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



En el caso que no se incluya este fichero, el sistema operativo asignará un icono por defecto a widget.

3.5 Ficheros .css

El fichero CSS (u hoja de estilo) sirve para configurar el layout y estilo del widget. Symbian acepta cualquier fichero siga la especificación CSS nivel 2 o 3.
Aunque el estilo del widget puede estar contenido dentro del fichero Main HTML, utilizando los atributos *style* dentro de los tags html, se recomienda la utilización del fichero CSS de cara a separar la apariencia del modelo de datos. Un widget puede tener tantos ficheros CSS como se necesiten [31-36].
Para enlazar una CSS a un fichero HTML se podrá utilizar cualquiera de los tags de HTML
utilizados para tal
efecto:

```
<link rel="stylesheet" type="text/css" href="Stylesheet.css" />
```

También es posible enlazar ficheros CSS que se encuentren en un servidor remoto. En este caso el valor de la propiedad *AllowNetworkAccess* del fichero info.plist deberá ser *true*.

```
<link rel="stylesheet" type="text/css" href="http://myserver.com/Stylesheet.css"/>
```

3.6 Ficheros Javascript

Los ficheros .js contienen el código JavaScript que contendrá la lógica del widget. Aunque el código JavaScript puede estar contenido dentro del fichero HTML, se recomienda la utilización de ficheros .js por motivos de modularidad y reutilización de código. Un widget puede tener tantos ficheros .js como se necesiten.

```
<script type='text/JavaScript' src='JavaScript.js'></script>
```

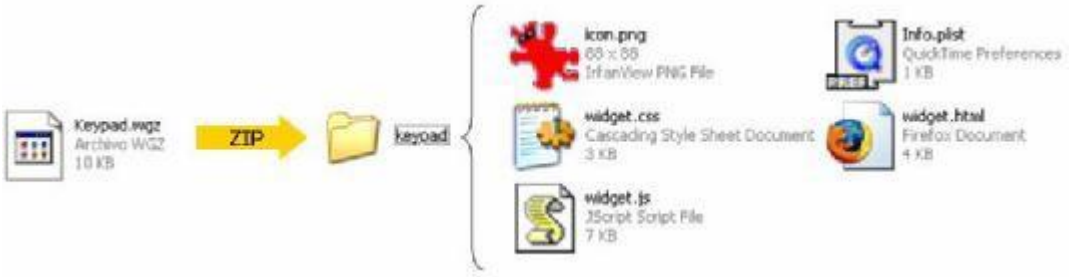
Al igual que en el caso anterior, también es posible enlazar ficheros .js que se encuentren en un servidor remoto. En este caso el valor de la propiedad *AllowNetworkAccess* del fichero info.plist deberá ser *true*.

```
<script type='text/JavaScript' src='http://www.widget.server/JavaScript.js'></script>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3.7 Preparación paquete del paquete de instalación

Antes de poder instalar el *widget* en el terminal es necesario generar el paquete de instalación. Este proceso es muy sencillo y consiste en comprimir con ZIP la carpeta donde está contenido el proyecto. A continuación deberá renombrarse la extensión del archivo a *.wgz.*



3.8 Instalación del widget

Los widgets pueden ser desplegados en los terminales móviles por varias formas:

- Por bluetooth o infrarrojos.
- Por medio del puerto USB o copiándolo directamente en la tarjeta multimedia MMC
- A través del navegador web, siempre que el servidor tenga configurado el tipo mime *x- nokia-widget* para los ficheros con extensión *.wgz*

3.9 Desarrollo en PC

Los widgets basados en WRT pueden ser probados y depurados en un PC, utilizando Firefox y algunos de sus extensiones:

- Extensión Firebug para depurar el widget.
- Extensión GreaseMonkey junto con el script "XmlHttpRequest Bypass Security", para eliminar alguna restricción de seguridad del objeto XmlHttpRequest

Por otra parte, existe un JavaScript (en desarrllo) en el cual están definidos algunos de los objetos JavaScript propietarios utilizados en WRT (widget, menú, etc). Este JavaScript se puede descargar de:
http://wiki.forum.nokia.com/images/e/e5/Widget_desktop_21112007.zip

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

3.10 API JavaScript

En el siguiente apartado se expone la API de JavaScript que pueden utilizar los desarrolladores a la hora de dotar de funcionalidad el widget [37-40].
A parte de los objetos y clases proporcionadas por el estándar JavaScript, el WRT incorpora los siguientes nuevos objetos:

- widget
- menu
- MenuItem
- SystemInfo

3.11 API JavaScript: Objeto widget

El objeto widget dispone de los siguientes métodos y propiedades:

- Métodos:
  - void openURL(String url)
  - void setPreferenceForKey (String preference, String key)
  - String preferenceForKey (String key)
  - void prepareForTransition(String transitionState)
  - void performTransition(void)
  - void setNavigationEnabled(Boolean flag)
  - void openApplication(HexNumber Uid, String param)
  - void setDisplayLandscape(void)
  - void setDisplayPortrait(void)

- Propiedades:
  - String identifier
  - void [Function] onshow
  - void [Function] onhide
  - Boolean isrotationsupported

3.11.1 *void openURL(String url)*

Syntaxis:

- [void] window.widget.openURL(String url)
- [void] widget.openURL(String url)

Descripción

- Abre la url especificado en el browser del terminal. El widget permanecerá abierto aunque en segundo plano.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Parámetros de entrada:

- url: La url que se desea abrir en el navegador.

Valor de retorno:

- Ninguno

Ejemplo:

- widget.openURL("www.google.es");

3.11.2 setPreferenceForKey()

Syntaxis:

- [void] window.widget.setPreferenceForKey(String preference, String key)

DescripciónPermite almacenar un par clave/valor de forma persistente. Las claves únicamente son eliminadas del terminal en caso de que el widget sea desinstalado. También se puede eliminar una clave especificando como valor "null"

Parámetros de entrada:

- preference: valor
- key: clave

Valor de retorno:

- Ninguno

Ejemplo:

```
var valor = document.forms[0].input1.value; widget.setPreferenceForKey('input1',
valor);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

3.11.3 preferenceForKey()

Syntaxis:

- [string] window.widget.preferenceForKey(String key)

Descripción

- Permite obtener un valor almacenado previamente por la función *setPreferenceForKey()*

Parámetros de entrada:

- Clave del valor

Valor de retorno:

- Cadena conteniendo el valor o *undefined* en caso de que la clave no exista.

Ejemplo:

```
if(widget.preferenceForKey('input1'))
{
var valor = widget.preferenceForKey('input1'); document.forms[0].input1.value =
valor;
}
```

3.11.4 prepareForTransition()

Syntaxis:

- [void] widget.prepareForTransition(String transitionMode)

Descripción

- Este método se utiliza conjuntamente con performTransition y su finalidad es preparar al widget antes de modificar su apariencia, para prevenir posibles parpadeos (*flickering*).
- Por ello es conveniente llamarlo antes de utilizar las siguientes propiedades de los objetos que componen la vista:
  - [object].style.display = “block”: Para hacer visible un elemento.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- [object].style.display = “none”: Para ocultarlo.

Parámetros de entrada:

- transitionMode: Cadena que define el modo de transición deseado. Actualmente solo está soportado el modo "fade" que provoca que la apariencia del widget se modifique con un efecto de desvanecimiento [41-45].

Valor de retorno:

- Ninguno

3.11.5 performTransition()

Syntaxis:

- [void] widget.performTransition(void)

Descripción

- Este comando se tiene que utilizar para actualizar la pantalla, una vez que se ha llamado al método prepareForTransition y se han realizado los cambios pertinentes en la vista.

Parámetros de entrada:

- Ninguno

Valor de retorno:

- Ninguno

Ejemplo:

Fichero .js

```
function toMain(main) { widget.prepareForTransi-
        tion("fade"); if (main) {
        document.getElementById("config").style.display = 'none';
        document.getElementById("main").style.display='block';
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        }
        else {
        document.getElementById("main").style.display = 'none'; docu-
        ment.getElementById("config").style.display = 'block';
        }
        widget.performTransition();
}
```

Main.html

```
<html>
<head>
        <title>Widget</title>
        <script type='text/JavaScript' src='holamundo.js'></script>
</head>

<body>
<div id='main'> Hola
Mundo !!
<input type="button" value="AcercaDe" onclick="toMain(0);" />
</div>
<div id='config' style="display:none">
<img src="usal.jpg" alt=""/>
<br/>
Universidad de Salamanca
<input type="button" value="Volver" onclick="toMain(1);" />
</div>
</body>
</html>
```

3.11.6 setNavigationEnabled()

Syntaxis:

- window.widget.setNavigationEnabled(Boolean navigationMode)

Descripción

- Este método se utiliza para cambiar la forma de navegar por el widget. Se puede elegir entre navegación con el cursor o tabulada.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Parámetros de entrada:

- navigationMode: Si es true: modo navegación por cursor. Si es false: Modo navegación tabulada.

Valor de retorno:

- Ninguno

Ejemplo:

```
// Modo Tab widget.setNavigationEna-
bled(false);
// Modo Cursor
widget.setNavigationEnabled(true);
```

La siguiente figura representa los dos modos de navegación, por cursor y tabulado:

Si el modo de navegación es tabulado (*navigationMode=false*), se pueden capturar eventos del teclado (*keyevents*) utilizando JavaScript:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
document.onkeydown = keyDown; func-
tion keyDown(event) {
    if(event.keyCode==49) { //“1”
```

La siguiente figura recoge los códigos de tecla que se pueden capturar:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Key	Key press	Key down	Key up
0	48/48	48/48	48/48
1	49/49	49/49	49/49
2	50/50	50/50	50/50
3	51/51	51/51	51/51
4	52/52	52/52	52/52
5	53/53	53/53	53/53
6	54/54	54/54	54/54
7	55/55	55/55	55/55
8	56/56	56/56	56/56
9	57/57	57/57	57/57
*	56/42	42/42	56/42
#	51/35	35/35	51/35
C (del)	8/8	8/8	8/8
Call creation key (green)	0/63586	63586/63586	0/63586
Center	0/63557	63557/63557	n/a
Left	37/63495	63495/63495	n/a
Up	38/63497	63497/63497	n/a
Right	39/63496	63496/63496	n/a
Down	40/63498	63498/63498	n/a

3.11.7 openApplication()

Syntaxis:

- [void] window.widget.openApplication(HexNumber Uid, String param)

Descripción

- Lanza una aplicación nativa (S60 C++) en modo stand-alone.

Parámetros de entrada:

- Uid: UID de la aplicación nativa
- param: Cadena que especifica el argumento pasar a la aplicación nativa

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Valor de retorno:

- Ninguno

Ejemplo:

```
// Lanza la aplicación de contactos widget.openApplication(0x101f4cce,"");
```

3.11.8 setDisplayLandscape() / setDisplayPortrait()

Syntaxis:

- widget.setDisplayLandscape(void)
- widget.setDisplayPortrait(void)

Descripción

- Estos métodos se utilizan para cambiar la orientación de la pantalla del terminal.
  Se puede elegir entre modo retrato (setDisplayPortrait)o apaisado (setDisplayLandscape).

Parámetros de entrada:

- Ninguno

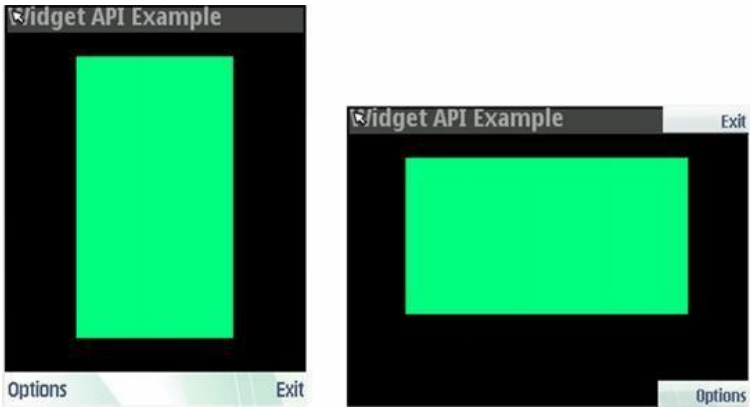
Valor de retorno:

- Ninguno

Ejemplo:

```
if(widget.isrotationsupported) widget.setDisplay-
        Landscape();
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc



3.11.9 identifier

Syntaxis:

- [String] widget.identifier

Descripción

- Propiedad de solo lectura. Valor único que identifica a un widget, según está definido en la entrada "Identifier" del fichero info.plist.

3.11.10 onshow

Syntaxis:

- [void] widget.onshow = function() { }

Descripción

- Función de callback que será llamada cuando un widget pase de segundo plano a primer plano.

3.11.11 onhide

Syntaxis:

- [void] widget.onhide = function() { }

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Descripción

- Función de callback que será llamada cuando un widget pase de primer plano a segundo plano.

3.11.12 isrotationsupported

Syntaxis:

- [Boolean] widget.isrotationsupported

Descripción

- Propiedad de solo lectura que retorna un valor booleano indicando si el terminal soporta orientación de pantalla apaisada o retrato. Si el valor es true, el terminal soporta ambas orientaciones de pantalla [46-50].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3.12 API JavaScript: Objeto menu

El objeto menu, junto con el objeto MenuItem, se utiliza para gestionar el menú de opciones del widget y las soft-keys. Este objeto dispone de los siguientes métodos y propiedades:

- Métodos:
  - void append(MenuItem menuItem)
  - void remove(MenuItem menuItem)
  - Object MenuItem getMenuItemById(Integer menuItemId)
  - Object MenuItem getMenuItemByName(String menuItemLabel)
  - void setRightSoftkeyLebel(String label, Function callbackfunction)
  - void showSoftkeys()
  - void hideSoftkeys()
  - void clear()
- Propiedades:
  - void [Function] onshow

3.12.1 *append()*

Syntaxis:

- [void] menu.append(MenuItem menuItem)

Descripción

- Este método permite añadir un ítem en el menú de opciones. Los ítems se muestran dentro del menú siguiendo el orden en el que son añadidos.

Parámetros de entrada:

- menuItem: instancia del objeto MenuItem que desea ser añadido en el menú de opciones.

Valor de retorno:

- Ninguno

3.12.2 *remove()*

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Syntaxis:

- menu.remove(MenuItem menuItem)

Descripción

- Elimina una entrada del menú de opciones.

Parámetros de entrada:

- menuItem: instancia del objeto MenuItem que desea ser eliminado del menú de opciones.

Valor de retorno:

- Ninguno

3.12.3 getMenuItemById()

Syntaxis:

- [MenuItem] menu.getMenuItemById(Integer id)

Descripción

- Permite recuperar el handle a una instancia de un ítem de menú específico a partir de su id.

Parámetros de entrada:

- id: El identificador de un ítem de menú existente.

Valor de retorno:

- Instancia del objeto MenuItem. Si el id especificado es inválido, el método retorna "undefined".

3.12.4 getMenuItemByName()

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Syntaxis:

- [MenuItem] menu.getMenuItemByName(String menuItemLabel)

Descripción

- Permite recuperar el handle a una instancia de un ítem de menú específico a partir de su nombre (label)

Parámetros de entrada:

- menuItemLabel: Nombre del ítem de menú que desea ser recuperado.

Valor de retorno:

- Instancia del objeto MenuItem. Si el nombre especificado es inválido, el método retorna "undefined".

Ejemplo:

3.12.5 setRightSoftkeyLabel()

Syntaxis:

- menu.setRightSoftkeyLabel(String label, Function callbackfunc)

Descripción

- Este método permite personalizar el nombre y la acción asociada a la softkey derecha. Por defecto, esta softkey está asignada a la función "Salir", la cual termina la ejecución del widget.
- Para reestablecer la softkey por defecto, se deberá llamar al método pasándole los

siguientes parámetros: menu.setRightSoftkeyLabel('', null)

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Parámetros de entrada:

- label: Cadena de texto que especifica el título de la softkey
- callbackfunc: Referencia a la función de callback, la cual será llamada cuando se presione la softkey derecha.

Valor de retorno:

- Ninguno

Ejemplo:

```
function switchToSettingsView()
{
window.menu.setRightSoftkeyLabel('Back',   returnToMainView);
}
functionreturnToMainView()
{
// Cambiar la vista
...
// Reestablecer la softkey por defecto window.menu.setRightSoftkeyLabel('',
null);
}
```

3.12.6 showSoftkeys()

Syntaxis:

- [void] menu.showSoftkeys(void)

Descripción

- Muestra el panel inferior que contiene las softkeys, el cual está oculto por defecto.

Parámetros de entrada:

- Ninguno

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Valor de retorno:

- Ninguno

Ejemplo:

```
window.menu.showSoftkeys();
```

3.12.7 hideSoftkeys()

Syntaxis:

- [void] menu.hideSoftkeys(void)

Descripción

- Oculta el panel que contiene las *softkeys*,

Parámetros de entrada:

- Ninguno

Valor de retorno:

- Ninguno

Ejemplo:

```
functionuseFullScreen()
{
window.menu.hideSoftkeys();
...
}
```

3.12.8 clear()

Syntaxis:

- [void] menu.clear(void)

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Descripción

- Elimina todos los ítems del menú de opciones.

Parámetros de entrada:

- Ninguno

Valor de retorno:

- Ninguno

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3.12.9 Propiedad: onShow

Syntaxis:

- [void] menu.onShow = menupaneShown;

Descripción

- Función de callback que será llamada cuando el menú de opciones es abierto, es decir, cuando el usuario presione el soft-button izquierdo.

3.13 API JavaScript: Objeto MenuItem

El objeto **MenuItem** permite gestionar, junto con el objeto menu, el menú principal del widget así como sus submenús [51].
El objeto MenuItem se crea utilizando el operador *new* y dispone de los siguientes métodos y propiedades:

- Métodos:
  - Object MenuItem(String label, Integer Id) o void append(MenuItem childMenuItem) o void remove(MenuItem childMenuItem) o void setDimmed(Boolean true|false)
- Propiedades:
  - void [Function] onSelect

3.13.1 Constructor

Syntaxis:

- [MenuItem] new MenuItem(String label, Integer id)

Descripción

- Se utiliza el operador new para crear una instancia del objeto.

Parámetros de entrada:

- label: Nombre del ítem
- id: Identificador único del ítem

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Valor de retorno:

- Instancia del objeto MenuItem.

3.13.2 append()

Syntaxis:

- [void] MenuItem.append(MenuItem childMenuItem)

Descripción

- Este método permite crear submenús dentro del menú de opciones. Los ítems se muestran dentro del menú siguiendo el orden en el que son añadidos [52].

Parámetros de entrada:

- childMenuItem: instancia del objeto MenuItem que desea ser añadido en el menú padre.

Valor de retorno:

- Ninguno

3.13.3 remove()

Syntaxis:

- [void] MenuItem.remove(MenuItem childMenuItem)

Descripción

- Elimina una entrada (y sus hijos) de un menú padre.

Parámetros de entrada:

- childMenuItem: instancia del objeto MenuItem que desea ser eliminado del menú.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Valor de retorno:

- Ninguno

3.13.4 setDimmed()

Syntaxis:

- [void] MenuItem.setDimmed(Boolean flag)

Descripción

- Este método se utiliza para mostrar u ocultar un ítem de menú existente. Por defecto, un ítem de menú se muestra cuando se añade al menú de opciones.

Parámetros de entrada:

- flag: true para mostrar el ítem, false para ocultarlo.

Valor de retorno:

- Ninguno

3.13.5 Propiedad onSelect

Syntaxis:

- MenuItem.onSelect = function(Integer id) { }

Descripción

- Función de callback que es llamada cada vez que un usuario selecciona un ítem del menu. El argumento que se le pasa a esta función es el id del ítem que se ha seleccionado.
- También es posible asignar una función de callback independiente para cada ítem.
  En este caso, el parámetro id pasado a la función de callback puede ser ignorado.
- La función de callback debe ser asignada después de que el menú se haya añadido al menú principal.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Ejemplo:

Creación del menú

```
window.onload = createMenu();
function createMenu()
{
var optionsMenu = window.menu;

optionsMenu.onShow = function()
{
alert('Evento: optionsMenu.onShow');
}
// Crear 2 items
var m1 = new MenuItem('Beverages', 2001); var m2 =
new MenuItem('Snacks', 2002);
// Asignar la function de callback m1.onSelect =
menuEventHandler; m2.onSelect = menuE-
ventHandler;

optionsMenu.append(m1); optionsMenu.append(m2);
var m11 = new MenuItem('Coca Cola', 3001);
var m12 = new MenuItem('Pepsi', 3002); optionsMenu.getMenu-
ItemById(2001).append(m11); optionsMenu.getMenuItemByName('Beverag-
es').append(m12); m11.onSelect = submenuEventHandler;
m12.onSelect = submenuEventHandler;
}
```

Implementación del manejador de eventos:

```
function menuEventHandler(id)
{
switch (id)
{
case 2001:
break; case
2002:
// TODO
break;
}
}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3.14 API JavaScript: SystemInfo API

Este API permite al widget acceder a ciertas propiedades del terminal así como controlar algunas de sus funcionalidades [53].
El API está implementado como un plug-in, y por tanto debe ser cargado añadiendo el siguiente código en el fichero html principal:

<embedtype="application/x-systeminfo-widget"hidden="yes"></embed> La refe-

rencia al objeto puede ser recuperada vía JavaScript utilizando el siguiente código:

var sysinfo = document.embeds[0];

Los servicios proporcionados por este plug-in están divididos en las siguientes categorías, conteniendo cada una los métodos y propiedades que siguen a continuación:

- Información de la batería
  - Integer chargelevel
  - String [Function] onchargelevel
  - Boolean chargerconnected
  - String [Function] onchargerconnected
- Información de la red
  - Integer signalbars
  - String networkname
  - Integer networkregistrationstatus
- Información y gestión de la luz de la pantalla y teclado del dispositivo.
  - Const int lightminintensity
  - Const int lightmaxintensity
  - Const int lightdefaultintensity
  - Const int lightinfiniteduration
  - Integer lightmaxduration
  - Integer lightdefaultcycletime
  - Const int lighttargetprimarydisplayandkeyboard

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

    - Const int lighttargetsystem
    - Void lighton(Int lighttarget, Int duration, Int intensity, Bool fadein);
    - Void lightoff(String lighttarget, Int duration, Int fadeout);
    - Void lightblink(String lighttarget, Int duration, Int onduration, Int offduration, Int intensity);
- Información y gestión del vibrador.
    - Const integer vibramaxintensity o Const integer vibramaxduration o Const integer Vibrasettings
    - Void startvibra(Int duration, Int intensity);
    - Void stopvibra();
- Gestión de tonos.
    - Void beep(Integer frequency, Integer duration);
- Información de la memoria y sistema de ficheros.
    - Integer totalram o Integer freeram o String drivelist
    - Integer drivesize(String drivename)
    - Integer drivefree(String drivename)
- Información del idioma.
    - String language

3.14.1 chargelevel

Syntaxis:

- [int] sysinfo.chargelevel;

Descripción

- Propiedad de solo lectura que retorna un valor entero de entre 0 a 100 que indica el valor actual de batería en %.

3.14.2 onchargelevel

Syntaxis:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- sysinfo.onchargelevel = "batteryLevelChange()";
- function batteryLevelChange() { ...};

Descripción

- Esta propiedad es el gestor del evento que se produce cuando cambia el nivel de batería.

Ejemplo:

```
window.onload = function(){
var batteryLevel = sysinfo.chargelevel; alert("Nivel de batería:
" + batteryLevel + "%");

sysinfo.onchargelevel = "batteryLevelChange()";
}
function batteryLevelChange()
{
var batteryLevel = sysinfo.chargelevel; alert("Nivel de bat-
ería:" + batteryLevel + "%");
}
```

3.14.3 chargerconnected

Syntaxis:

- [boolean] sysinfo.chargerconnected;

Descripción

- Propiedad de solo lectura que retorna un booleano indicando el estado del cargador.
- Si el cargador está conectado devolverá true, si no false.

3.14.4 onchargerconnected

Syntaxis:

- sysinfo.onchargerconnected = "chargerConnectedEventHandler()";
- function chargerConnectedEventHandler(){…}

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Descripción

- Esta propiedad es el gestor del evento que se produce cuando el cargador se conecta o desconecta al terminal.
- Debido a que el evento se lanza antes de actualizar el valor de la propiedad chargerconnected, se recomienda aplicar un pequeño retardo antes de leer dicho valor.

Ejemplo:

```
window.onload = function(){
sysinfo.onchargerconnected="chargerConnectedEvent()";
}
functionchargerConnectedEvent()
{
setTimeout("readValue();", 500);
}
function readValue()
{
if(sysinfo.chargerconnected) alert("El carga-
dor está conectado "); else
alert(“El cargador no está conectado”);
}
```

3.14.5 signalbars

Syntaxis:

- [int] sysinfo.signalbars;

Descripción

- Propiedad de solo lectura que retorna un entero (de 0 a 7) indicando el estado de la señal de red.
- 0 indica que no hay señal y 7 que la señal es máxima

Ejemplo:

```
var signalStrength = sysinfo.signalbars; if (signal-
Strength == 0)
alert("¡Sinseñal!");
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3.14.6 networkname

Syntaxis:

- [string] sysinfo.networkname;

Descripción

- Propiedad de solo lectura que devuelve una cadena conteniendo el nombre de la red móvil registrada en ese momento por el terminal.

3.14.7 networkregistrationstatus

Syntaxis:

- [int] sysinfo.networkregistrationstatus;

Descripción

- Propiedad que retorna un entero (de 0 a 7) indicando el estado de registro de la red.
  0. Desconocido
  1. No registrado. El terminal no puede detectar otras redes.
  2. No registrado. El terminal puede detectar otras redes en las únicamente se puede realizar una llamada de emergencia.
  3. No registrado, pero el terminal está buscando a otra red con registrarse. operador de red. El terminal puede detectar otras redes cuales únicamente se puede realizar una llamada de emergencia.
  4. Registrado, red ocupada
  5. Registrado en red local
  6. Registro denegado
  7. Registrado en red visitante (roaming)

Ejemplo:

```
window.onload = function(){ net-
workRegEventHandler();
}
networkRegEventHandler() { var
textInfo = null;
var networkName = "";
var status = sysinfo.networkregistrationstatus; switch (status) {
case 0:
case 1:
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
case 2:
case 3:
case 4:
case 6:
textInfo = "Fallo en el registro de red"; break;
case 5:
networkName = sysinfo.networkname; textInfo
="Registrado en la red local "; break;
case 7:
networkName = sysinfo.networkname;
textInfo =" Registrado en la red visitante "; break;
default:
break;
}
alert(textInfo + networkName);
}
```

3.14.8 lightminintensity

Syntaxis:

- [const int] sysinfo.lightminintensity

Descripción

- Propiedad de solo lectura que devuelve una constante que determina el brillo mínimo soportado por el terminal.

3.14.9 lightmaxintensity

Syntaxis:

- [const int] sysinfo.lightmaxintensity

Descripción

- Propiedad de solo lectura que devuelve una constante que determina el brillo máximo soportado por el terminal.

3.14.10 lightdefaultintensity

Syntaxis:

- [const int] sysinfo.lightdefaultintensity

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Descripción

- Propiedad de solo lectura que devuelve una constante que determina el brillo por defecto.

3.14.11 lighttargetprimarydisplayandkeyboard

Syntaxis:

- [const int] sysinfo.lighttargetprimarydisplayandkeyboard
-

Descripción

- Constante para especificar el display primario y la luz del teclado.

3.14.12 lighttargetsystem

Syntaxis:

- [const int] sysinfo.lighttargetsystem

Descripción

- Constante para especificar todos los displays (primario y secundario) y la luz del teclado.

3.14.13 lighton()

Syntaxis:

- [void] sysinfo.lighton(Int lighttarget, Int duration, Int intensity, Bool fadein)

Descripción

- Este método conmuta la luz especificada durante un tiempo e intensidad determinada.

Parámetros de entrada:

- lighttarget: tipo de luz.
- duration: Periodo (en ms.) durante el cual debe encenderse la luz. Si la duración se especifica como *lightinfiniteduration*, la luz permanecerá encendida indefinidamente.
- intensity: Define la intensidad (brillo) de la luz. El valor especificado debe estar

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

dentro de los rangos definidos por *lightminintensity* y *lightmaxintensity*.
- fadein: Si el valor es true, enciende la luz utilizando el efecto de fade-in.

Valor de retorno:

- Ninguno

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Ejemplo:

```
functioncallLightOn()
{
var sysinfo = document.embeds[0];
var target = sysinfo.lighttargetsystem; var duration
= sysinfo.lightmaxduration;
var intensity = sysinfo.lightdefaultintensity;
sysinfo.lighton(target, duration, intensity, true);
}
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

3.14.14 lightoff()

Syntaxis:

- [void] sysinfo.lightoff(String lighttarget, Int duration, Int fadeout)

Descripción

- Este método apaga la luz durante el tiempo especificado.

Parámetros de entrada:

- lighttarget: Tipo de luz a apagar.
- duration: Periodo (en ms.) durante el cual se mantendrá la luz apagada. Si la duración se especifica como lightinfiniteduration, la luz permanecerá apagada indefinidamente.
- fadeout: Si el valor es true, se apagará la luz utilizando el efecto de fade-out. Si es
  false, se apagarán de forma inmediata

Valor de retorno:

- Ninguno

Ejemplo:

```
function callLightOff()
{
var sysinfo = document.embeds[0];
var target = sysinfo.lighttargetsystem; var duration
= 5000; // 5 seconds sysinfo.lightoff(target,dura-
tion,true);
}
```

3.14.15 lightblink()

Syntaxis:

- [void] sysinfo.lightblink(String lighttarget, Int duration, Int onduration, Int offduration, Int intensity)

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Descripción

- Este método enciende y apaga la luz de forma intermitente durante la duración especificada.

Parámetros de entrada:

- lighttarget: Tipo de luz a utilizar.
- duration: Duración (en ms.) durante el cual se mantendrá la luz de forma intermitente. Si la duración se especifica como lightinfiniteduration, la duración de la intermitencia será indefinida.
- onduration: En cada ciclo de intermitencia, la luz estará encendida durante el tiempo especificado por este valor (en ms.)
- offduration: En cada ciclo de intermitencia, la luz estará apagada durante el tiempo
  especificado por este valor (en ms.)
- Define la intensidad (brillo) de la luz. El valor especificado debe estar dentro de los rangos definidos por lightminintensity y lightmaxintensity.

Valor de retorno:

- Ninguno

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Ejemplo:

```
function callLightBlink()
{
// get the Embed element reference var sys-
info = document.embeds[0];
var target = sysinfo.lighttargetsystem; var duration
= 5000; // 5 segundos
var onDur = sysinfo. lightdefaultcycletime; var offDur =
sysinfo. lightdefaultcycletime; var intensity = sys-
info.lightdefaultintensity;
sysinfo.lightblink(target, duration, onDur, offDur, intensity);
}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3.14.16 vibraminintensity

Syntaxis:

- [int] sysinfo.vibraminintensity

Descripción

- Propiedad de sólo-lectura que representa la intensidad mínima de vibración.

3.14.17 vibramaxintensity

Syntaxis:

- [int] sysinfo.vibramaxintensity

Descripción

- Propiedad de sólo-lectura que representa la intensidad máxima de vibración.

3.14.18 vibramaxduration

Syntaxis:

- [int] sysinfo.vibramaxduration

Descripción

- Propiedad de sólo-lectura que representa la duración máxima permitida de vibración.

3.14.19 vibrasettings

Syntaxis:

- [int] sysinfo.vibrasettings

Descripción

- Propiedad de sólo-lectura que devuelve el modo de vibración establecido en el perfil de usuario activo.
- Los valores que puede devolver esta propiedad son los siguientes:
  - 0: Error o no inicializado.
  - 1: Vibrador activado

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- 2: Vibrador desactivado

3.14.20 startvibra()

Syntaxis:

- [void] sysinfo.startvibra(Int duration, Int intensity)

Descripción

- Enciende el vibrador durante el tiempo e intensidad especificados.

Parámetros de entrada:

- duration: Duración en milisegundos. El valor 0 indica duración indefinida.
- intensity: Intensidad de la vibración. Valor debe estar contenido en el rango de -100 a 100. Si el valor es negativo, el motor del vibrador rota en sentido negativo, si no en sentido positivo.

Valor de retorno:

- Ninguno

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Ejemplo:

```
function startVibration()
{
var duration = sysinfo.vibramaxduration; var inten-
sity = sysinfo.vibraminintensity
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
sysinfo.startvibra(duration,  intensity);
}
```

3.14.21 stopvibra()

Syntaxis:

- [void] sysinfo.stopvibra()

Descripción

- Apaga el vibrador de forma inmediata

Parámetros de entrada:

- Ninguno

Valor de retorno:

- Ninguno

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Ejemplo:

```
function callVibration()
{
var duration = 0;
var intensity = sysinfo.vibraminintensity sysinfo.startvi-
bra(duration,  intensity);
// Pulsar cualquier tecla para parar el vibrador
document.addEventListener("keypress", kpListener, false);
}

functionkpListiner(event)
{
sysinfo.stopvibra(); document.removeEventListener("key-
press",kpListener, false);
}
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

3.14.22 beep()

Syntaxis:

- [void] sysinfo.beep(Int frequency, Int duration)

Descripción

- Reproduce un tono con la frecuencia y duración especificada.

Parámetros de entrada:

- frequency: Frecuencia del tono medida en Hz.
- duration: Duración en ms. Si el valor es 0, entonces el tono durará indefinidamente. o Para parar la reproducción de un tono con duración indefinida, será necesario volver llamar a este método especificando un valor distinto de 0
  para el parámetro duration.

Valor de retorno:

- Ninguno

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Ejemplo:

```
function beepAlert()
{
sysinfo.beep(220, 2000);
}
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

3.14.23 totalram

Syntaxis:

- [int] sysinfo.totalram

Descripción

- Propiedad de sólo-lectura que devuelve la cantidad total de RAM del terminal, medida en bytes.

Ejemplo:

```
var totalRamSize = sysinfo.totalram;
alert("RAM Total: " + totalRamSize/1024 + "kB");
```

3.14.24 freeram

Syntaxis:

- [int] sysinfo.freeram

Descripción

- Propiedad de sólo-lectura que devuelve la cantidad de RAM libre del terminal, medida en bytes.

Ejemplo:

```
var freeram= sysinfo.freeram;
alert("RAM Libre: " + freeram/1024 + "kB");
```

3.14.25 drivelist

Syntaxis:

- [string] sysinfo.drivelist

Descripción

- Propiedad de sólo-lectura que devuelve los nombres de las unidades de usuario separadas por un espacio.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Las unidades z: y d: son unidades del sistema y por lo tanto no son devueltas por esta propiedad.

3.14.26 drivesize()

Syntaxis:

- [int] sysinfo.drivesize(String drivename)

Descripción

- Este método permite conocer el tamaño total de la unidad especificada.

Parámetros de entrada:

- drivename: Unidad a ser examinada

Valor de retorno:

- Tamaño total (en bytes) de la unidad.

3.14.27 drivefree ()

Syntaxis:

- [int] sysinfo.drivefree(String drivename)

Descripción

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- Este método permite conocer el espacio libre que hay disponible en la unidad especificada.

Parámetros de entrada:

- drivename: Unidad a ser examinada

Valor de retorno:

- Espacio libre (en bytes) de la unidad.

Ejemplo:

```
function checkDrivesInformation() { var space = 0;
var allDrives = sysinfo.drivelist; var drives = allDrives.split("
");

 for (var i=0; i<drives.length; i++) {
        space=sysinfo.drivesize(drives[i]); space /=1024; // bytes ->
        kB
        alert("Total"+drives[i]+"="+space+"kB"; space=sysinfo.drive-
        free(drives[i]); space /=1024; // bytes -> kB
        alert("Libre"+drives[i]+"="+space+"kB";
                }
               }
```

3.14.28 language

Syntaxis:

- [string]sysinfo.language

Descripción

- Propiedad de sólo lectura que retorna el idioma utilizado por el terminal en formato ISO 639-1. Por ejemplo:
  - Inglés: en
  - Alemán: de
  - Español: es

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

References

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

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

20. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. Expert Systems with Applications, 41(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003
21. García, O., Chamoso, P., Prieto, J., Rodríguez, S., & De La Prieta, F. (2017). A serious game to reduce consumption in smart buildings. In Communications in Computer and Information Science (Vol. 722, pp. 481–493). https://doi.org/10.1007/978-3-319-60285-1_41
22. Glez-Bedia, M., Corchado, J. M., Corchado, E. S., & Fyfe, C. (2002). Analytical model for constructing deliberative agents. International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications, 10(3).
23. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. BMC Bioinformatics, 10. https://doi.org/10.1186/1471-2105-10-187
24. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
25. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
26. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
27. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
28. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. Applied Mathematics and Computation, 270, 756–767. https://doi.org/10.1016/j.amc.2015.08.059
29. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. Expert Systems with Applications, 36(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
30. Méndez, J. R., Fdez-Riverola, F., Díaz, F., Iglesias, E. L., & Corchado, J. M. (2006). A comparative performance study of feature selection methods for the anti-spam filtering domain. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4065 LNAI, 106–120. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-33746435792&partnerID=40&md5=25345ac884f61c182680241828d448c5
31. Pablo Campillo-Sánchez, Juan Antonio Botía, Jorge Gómez-Sanza (2013). Development of Sensor Based Applications for the Android Platform: an Approach Based on Realistic Simulation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 1
32. Palomino, C. G., Nunes, C. S., Silveira, R. A., González, S. R., & Nakayama, M. K. (2017). Adaptive agent-based environment model to enable the teacher to create an adaptive class. Advances in Intelligent Systems and Computing (Vol. 617). https://doi.org/10.1007/978-3-319-60819-8_3
33. Sigeru Omatu, Hideo Araki, Toru Fujinaka, Mitsuaki Yano, Michifumi Yoshioka, Hiroyuki Nakazumi, Ichiro Tanahashi (2012). Mixed Odor Classification for QCM Sensor Data by Neural Network. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
34. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. International Journal of Ambient Computing and Intelligence, v 1, n 1(1), 15–26. https://doi.org/10.4018/jaci.2009010102
35. Xian Wang, Paula Tarrío, Ana María Bernardos, Eduardo Metola, José Ramón Casar (2012). User-independent accelerometer-based gesture recognition for mobile devices. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
36. Angelo Costa, Stella Heras, Javier Palanca, Paulo Novais, Vicente Julián (2016). Persuasion and Recommendation System Applied to a Cognitive Assistant. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
37. David Griol, Jose M. Molina (2016). A proposal to manage multi-task dialogs in conversational interfaces. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
38. Marco Antonio Ameller, María Angélica González (2016). Minutiae filtering using ridge-valley method. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

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

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Desarrollo de juegos con J2ME

Manuel-Jesús Prieto Martín [1]

[1] Telefónica Investigación y Desarrollo, Spain
mjprieto@telefonica.es

Resumen: Este capítulo hacer una pequeña introducción al mundo del entretenimiento móvil y muestra como J2ME ha sido un elemento clave en él. Debido a que las tecnologías y los terminales avanzan y mejoran a gran velocidad, los servicios también deben evolucionar de forma rápida, lo que provoca un gran dinamismo en el mercado y un gran abanico de posibilidades para las empresas. A continuación, vamos a ver algunos de los marcos de trabajo en los que han aparecido servicios de entretenimiento para el mundo móvil. Todos los analistas coinciden en que este mercado será un gran negocio para todos los actores involucrados. Por un lado, tendremos a los operadores de telefonía que aumentarán sus beneficios, ya que serán ellos los que proveerán estos servicios y por lo tanto, generarán negocio en un primer momento con la provisión de los juegos y posteriormente con el uso de los mismos. Por otra parte, tenemos a las empresas de desarrollo de servicios para móviles. Entre estos servicios, tenemos los juegos (sea cómo sean estos y sea cual sea la tecnología en la que se basan) y dichas empresas de desarrollo se beneficiarán del aumento de la demanda de este tipo de servicios por parte de las operadoras. En este capítulo se presenta como J2ME está contribuyendo al desarrollo de este campo.

Palabras clave: J2ME; Juegos

Abstract. This chapter makes a small introduction to the world of mobile entertainment and shows how J2ME has been a key element in it. Because technologies and terminals are advancing and improving at high speed, services must also evolve rapidly, resulting in a very dynamic market and a wide range of possibilities for businesses. Below we will look at some of the frameworks in which entertainment services have appeared for the mobile world. All analysts agree that this market will be a big business for all actors involved. On the one hand, we will have the telephone operators that will increase their profits, since they will be the ones that will provide these services and therefore, they will generate business in a first moment with the provision of the games and later with the use of the same ones. On the other hand, we have the mobile services development companies. Among these services, we have games (no matter how they are and no matter what technology they are based on) and these development companies will benefit from the increased demand for this type of services by operators. In this chapter it is presented how J2ME is contributing to the development of this field.

Keywords: J2ME; Games

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

1 INTRODUCCIÓN A LOS JUEGOS EN MÓVILES

Dentro del mundo de los negocios basados en tecnologías móviles, uno de los campos que más expectativas está generando y que según los analistas crecerá exponencialmente en los próximos años es el mundo del entretenimiento a través del móvil. En este documento, vamos a hacer una pequeña introducción al mundo del entretenimiento móvil y vamos a situar la tecnología J2ME dentro de este campo, de tal manera que nos sirva de punto de comienzo para nuestra incursión en el entretenimiento móvil [1-5].

Si dejamos a un lado algunos casos aislados como Japón y Corea del Sur, el mercado del entretenimiento móvil, basado en terminales y tecnologías móviles, está aún en su comienzo, pero está empezando a ser una realidad indudable y palpable, sin duda alguna. Todos los analistas coinciden en que este mercado será un gran negocio para todos los actores involucrados. Por un lado, tendremos a los operadores de telefonía que aumentarán sus beneficios, ya que serán ellos los que proveerán estos servicios y por lo tanto, generarán negocio en un primer momento con la provisión de los juegos y posteriormente con el uso de los mismos. Por otra parte, tenemos a las empresas de desarrollo de servicios para móviles. Entre estos servicios, tenemos los juegos (sea cómo sean estos y sea cual sea la tecnología en la que se basan) y dichas empresas de desarrollo se beneficiarán del aumento de la demanda de este tipo de servicios por parte de las operadoras. Su negocio básico tiene dos vertientes, por una parte, pueden dedicarse a crear servicios para las operadoras y que estas sean las distribuidoras, o por otra parte, pueden crear un servicio y explotarlo junto con el operador, de tal manera que ambos se benefician de los resultados de dicho servicio. Este último modelo es el más común dentro del mercado japonés, en el que las empresas desarrollan un servicio y lo explotan junto con la operadora, repartiéndose los beneficios.

A pesar de estar en la fase inicial, las previsiones son muy alentadoras. Hace unos años, se hablaba de la gran cantidad de terminales móviles que habría en el futuro. Hoy, esas previsiones se han cumplido y la masa crítica de terminales móviles a la que se pueden orientar el mercado y ofrecer servicios es enorme. Además, los terminales se usan cada vez más para operaciones diferentes a la simple comunicación por voz. El más claro ejemplo de esto son los mensajes SMS que hoy por hoy son una verdadera fiebre para muchos usuarios. Si unimos estas cuestiones al hecho de que cada vez los terminales son mucho más potentes y atractivos, disponen de mejores pantallas y poseen sistemas de sonido cada vez más sofisticados, tendremos un número mayor de factores a favor de la explosión definitiva de los servicios móviles y entre ellos, de los servicios de entretenimiento móviles. Actualmente todas las operadoras están tomando posiciones y cada vez son más los servicios disponibles. Debido a que las tecnologías y los terminales avanzan y mejoran a gran velocidad, los servicios también deben evolucionar de forma rápida, lo que provoca un gran dinamismo en el mercado y un gran abanico de posibilidades para las empresas.

A continuación, vamos a ver algunos de los marcos de trabajo en los que han aparecido servicios de entretenimiento para el mundo móvil.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

1.1 ENTRETENIMIENTO BASADO EN SMS Y MMS

SMS son las siglas de *Short Message Service*. La mensajería SMS se basa en el envío de mensajes de texto entre los terminales móviles. Como podemos comprender, la comunicación a través de mensajes de texto cortos es bastante limitada, pero este servicio ha sido un verdadero éxito en el mundo de las telecomunicaciones. En la actualidad hay en el mercado un amplio muestrario de servicios móviles basados en SMS, entre ellos, muchos servicios de entretenimiento.

Los juegos basados en SMS, como podemos comprender, deben ser muy sencillos ya que lo único que tenemos son mensajes de texto y además de un tamaño limitado. Algunos juegos típicos en este tipo de entorno son los juegos de pregunta-respuesta, o los juegos de aventuras basados en texto.

Otro tipo de entretenimiento basado en SMS y que no son exactamente juegos, son los sistemas de *chat* o similares que se utilizan este tipo de tecnología e incluso la integran con tecnologías no móviles, de tal forma que un usuario con un terminal móvil, puede *chatear* con un usuario que está frente a su PC.

Los servicios SMS pueden ser mejorados usando otras tecnologías, como la localización. La localización permite conocer de manera aproximada la posición del terminal móvil. De esta manera, las capacidades de los juegos y servicios de entretenimiento, pueden ser aumentadas y los resultados serán más atractivos para el usuario.

La evolución de los SMS han sido los MMS (si pasamos por alto los EMS). MMS es el acrónimo de *Multimedia Messaging Service*. Como su nombre indica, son mensajes en los que aparte de texto, se pueden incluir elementos multimedia, como son imágenes, sonidos e incluso video. Esta tecnología abre nuevas posibilidades en el mundo de los juegos basados en SMS ya que el resultado puede ser mucho más rico y por lo tanto el número de juegos y servicios que podemos crear es mayor [6-10].

1.2 ENTRETENIMIENTO BASADO EN WAP

Los juegos y servicios de entretenimiento basados en sistemas de navegación, trabajan sobre WAP (*Wireless Application Protocol*) y utilizan lenguajes de marcado como WML (*Wireless Markup Language*) o xHTML. De forma poco formal, podemos decir que WAP se puede considerar como la tecnología que nos permite navegar por Internet desde terminales móviles.

El tipo de servicios que nos permite ofrecer esta tecnología no son muy completos, pero

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

también hay un pequeño nicho de oportunidades en este campo. Los juegos que podemos desarrollados para este tipo de tecnología son similares a los presentados en el caso de los juegos basados en mensajería.

1.3 ENTRETENIMIENTO BASADO EN APLICACIONES

Existen varias tecnologías que nos permiten desarrollar aplicaciones que se ejecutan en el terminal. Entre estas aplicaciones, podemos incluir los juegos, y por lo tanto crear juegos propiamente dichos, que aprovechen todas las capacidades del terminal. Estas aplicaciones se ejecutan dentro del terminal y se instalan en el mismo a través de una conexión directa o a través de un proceso de descarga.

Sin duda alguna esta es la opción de las existentes que más posibilidades presenta y será la más demandada por parte del usuario ya que es la que aporta más a la experiencia del usuario y la que aprovecha al máximo todas las capacidades del terminal.

Para realizar las aplicaciones, tendremos varias tecnologías disponibles. Tenemos, entre otras, los teléfonos con sistema operativo Symbian, los teléfonos con sistemas Windows de Microsoft, también denominados *SmartPhones*, tenemos J2ME que será la tecnología en la que se basa este documento, la tecnología BREW...

Las dos primeras opciones, los sistemas Symbian y los *Smartphone* tienen la ventaja de que las aplicaciones se desarrollan como aplicaciones nativas del sistema, por lo que tienen la capacidad de aprovechar totalmente todos los elementos del terminal y por lo tanto, se pueden realizar aplicaciones, juegos en nuestro caso, más completas y ambiciosas.

Otra tecnología disponible para el desarrollo de aplicaciones y que actualmente es una de las más extendidas e implantadas en el mercado, es J2ME. A estas alturas ya tenemos un conocimiento claro de lo que es J2ME y deberíamos conocer su potencia y saber también sus carencias. Entre otras cosas, no tenemos acceso a todas las capacidades del terminal (agenda, APIs de bajo nivel...) y debemos tener en cuenta que los *midlets* J2ME funcionan sobre una máquina virtual, por lo que siempre serán más lentos que una aplicación nativa.

En lo comentado en el párrafo anterior, tenemos también la gran ventaja de J2ME y es su portabilidad, como es común en Java. Es decir, cuando creamos un *midlet* este puede instalarse y ejecutarse en terminales diferentes, de diferentes marcas, capacidades y con diferentes prestaciones. Como ya hemos comentado anteriormente y como veremos más adelante, para conseguir un resultado óptimo, habrá que ajustar el *midlet* a cada terminal. Esto se puede hacer en tiempo de ejecución o en tiempo de desarrollo, pero incluso con este esfuerzo extra, la reutilización de código es importante y por lo tanto el valor de J2ME como lenguaje de desarrollo para crear aplicaciones para un gran número de terminales muy diferentes entre si permanece intacto [11-15].

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Por otra parte, se está trabajando y se están definiendo constantemente ampliaciones de la especificación básica de J2ME para la provisión de APIs que aumenten las posibilidades funcionales de J2ME y así se aprovechen al máximo las capacidades y las prestaciones de los terminales. Por ejemplo, tenemos APIs adicionales (creadas o en proceso de definición) para la gestión de la tecnología *BlueTooth*, para la utilización de las capacidades de mensajería de los terminales o para la gestión de ficheros.

2 JUEGOS ESTÁTICOS

2.1 INTRODUCCIÓN

En primer lugar, como primera aproximación a la programación de videojuegos con J2ME, vamos a ver cómo desarrollar lo que denominaremos juegos estáticos. Este tipo de juegos, son juegos en los que no hay un movimiento continuo o masivo de elementos dentro de las pantallas del juego. Dentro de este grupo de juegos, podemos incluir los juegos de sobremesa, en los que se nos presenta una pantalla, realizamos un movimiento y se nos presenta una nueva pantalla, siendo todas estas pantallas netamente estáticas con respecto a los elementos gráficos que la componen.

El objetivo de este capítulo es comenzar con la gestión avanzada de gráficos y de la interacción con el usuario, para afianzar una serie de conceptos e ideas que serán básicos en el desarrollo de juego más complicados y completos. Para ilustrar todo esto, vamos a realizar un juego de este tipo. En concreto, el juego a desarrollar en el conocido buscaminas, que al ser muy popular, nos evita el plantearnos las normas o reglas del juego y al ser tan sencillo, nos permite centrarnos en los puntos del desarrollo que realmente nos interesan para nuestro objetivo.

Toda la funcionalidad contenida en este ejemplo y por lo tanto todo lo que se presenta en este capítulo está basada en la versión 1.04 de MIDP, por lo que no usa ninguna de las características orientadas al desarrollo de juegos que se incluyen dentro de la versión 2.0 de la especificación MIDP.

2.2 INTRODUCCIÓN AL OBJETO GRAPHICS

Antes de comenzar con el desarrollo del juego propiamente dicho, vamos a ver una introducción al objeto *Graphics*, de tal forma que el proceso de creación de las pantallas posteriormente sea más sencillo de comprender.

El objeto *Graphics*, perteneciente al paquete *javax.microedition.lcdui*, es el componente esencial de cualquier operación gráfica que se realice dentro de los midlets. El objeto *Graphics*, proporciona funciones para dibujar texto, imágenes, líneas, rectángulos y arcos de curva.

Como veremos en el código de ejemplo dentro de este mismo capítulo, las operaciones de dibujo se pueden realizar tanto sobre la pantalla del terminal *directamente* como sobre una imagen, que se compondrá totalmente independiente de la pantalla y que luego será mostrada en la pantalla de una sola vez. Esta solución evita que se produzcan parpadeos en las animaciones

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

y en los procesos de repintado de la pantalla. Para dibujar sobre una imagen externa a la pantalla, crearemos primero un objeto *Image* con las dimensiones deseadas (típicamente el tamaño de la pantalla o lo que es lo mismo, del *Canvas* final en el que vamos a dibujar). Una vez que tenemos la imagen, llamamos al método *getGraphics*, que nos retornará un objeto *Graphics*. Este objeto se puede usar tantas veces como queramos y definitivamente,

dentro del método *paint* del *Canvas* dibujaremos la imagen compuesta, lo que provoca que se dibuje esta imagen definitivamente sobre la pantalla.

El sistema de coordenadas utilizado para las operaciones de dibujo tiene su punto de origen en la esquina superior-izquierda del rectágulo de dibujo, es decir, de la pantalla o de la imagen en el caso de estar dibujando sobre una imagen externa.

Para controlar mejor los efectos sobre la pantalla de las operaciones de dibujo, podemos definir un rectángulo de *clipping* o recorte, de tal manera que sólo dentro de dicho rectángulo tendrán efecto las operaciones de dibujo. Se puede definir un rectángulo de *clipping* de tamaño cero, de tal forma que las operaciones de dibujo no tienen ningún efecto sobre la pantalla [16].

Los puntos de anclaje o *anchor points* se utilizan para posicionar el texto y las imágenes a la hora de dibujarlos. Estos puntos de anclaje se utilizan para reducir la necesidad de computación necesaria para situar el texto o la imagen. Por ejemplo, para centrar un texto, la aplicación necesita determinar la anchura del mismo a través de los métodos *stringWidth* o *charWidth* y realizar los cálculos necesarios para situar el texto en la posición correcta. Para definir el punto de anclaje debemos especificar un valor para el parámetro horizontal y otro valor para el parámetros vertical. Los valores que puede tomar el parámetro horizontal son las constantes LEFT, HCENTER y RIGHT y los valores que puede tomar el parámetro vertical son las constantes TOP, BASELINE y BOTTOM. Para combinar ambos parámetros se utiliza el operado OR:

```
g.drawString("Texto de prueba", 10, 10,
Graphics.TOP | Graphics.LEFT);
```

La posición del texto con respecto al punto especificado en el método de dibujo es determinada por el punto de anclaje.

2.3 COMENZANDO: PANTALLAS DE PRESENTACIÓN E INSTRUCCIONES

Para comenzar vamos a ver cómo se construye la pantalla de presentación del juego. Esta pantalla está presente en la mayoría de los juegos y en nuestro caso nos permitirá realizar nuestra primera aproximación al objeto *Graphics* y al funcionamiento de los objetos tipo *Canvas*.

Lo primera cuestión a plantearnos a la hora de desarrollar un juego, una vez que ya tenemos claras las especificaciones del juego en cuestión, es el diseño del software a desarrollar. Este punto es

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

esencial ya que nos facilitará enormemente el desarrollo y hará que el software creado sea de mayor calidad, al favorecer la portabilidad y adaptabilidad del software. Estas características son muy importantes en cualquier software, pero en el caso de los midlets son aún más importantes, ya que es un imperativo en el proceso de desarrollo de software J2ME si queremos que nuestro midlet se adapte a las capacidades de los diferentes terminales y que la implementación de esta adaptación no suponga un esfuerzo enorme.

El objetivo que nos propondremos al crear un juego, será que este se adapte de la mejor forma posible a las características de los terminales. Para conseguir este ambicioso objetivo, tendremos que hacer el diseño de las pantallas con sumo cuidado y adaptar nuestras operaciones de dibujo siempre que sea posible, a las características del terminal.

Lamentablemente, para conseguir un resultado óptimo y totalmente satisfactorio, deberemos crear en la mayoría de los casos una versión diferente del midlet para cada terminal. Cada una de estas versiones, tendrá una serie de imágenes adaptadas a las características del terminal. Bien, si partimos de este supuesto, tendremos que poner especial cuidado en el diseño del software, para que se puedan sustituir ciertas partes del código, las correspondientes a las pantallas que debamos adaptar, de forma sencilla y el impacto de esta acción se mínimo.Como primera aproximación al mundo de los gráficos orientados a juegos en J2ME, vamos a hacer una pantalla de presentación, sin muchas pretensiones pero que nos servirá para ver algunas de las técnicas básicas de trabajo con J2ME en general y con la programación de juegos en particular. La pantalla de presentación que vamos a hacer será una pantalla sencilla con una imagen. La primera versión de la pantalla de presentación será la siguiente:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
import javax.microedition.lcdui.*;
import java.io.IOException;
public class Presentacion extends Canvas{ private Buscabombas
    buscabombas = null; private Menu menu = null;
    private boolean salir = false;
    private Image logo; private Image tex-
    toImg;

    Font f1 = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,Font.SIZE_MEDIUM);
    int ancho; int alto;
    int anchoTit2; int an-
    choTit1; int ximg;
    int yimg;

    public Presentacion(Buscabombas buscabombas) { this.buscabombas = buscabom-
        bas;
        menu = new Menu(Recursos.getString(
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                Recursos.ID_NOMBRE),

                List.IMPLICIT,buscabombas);

        try{

            logo = Image.createImage("/LogoBuscabombas.png");

        }catch(IOException ioe){ ioe.printStack-

            Trace();

        }

        ancho = this.getWidth(); alto =

        this.getHeight();

        ximg = (int)((ancho-logo.getWidth())/2);

        yimg = (int)((alto-logo.getHeight())/2);

    }

    protected void keyPressed(int keyCode){ buscabom-

        bas.mostrar(menu);

    }

    protected void paint(Graphics g) {

        // Borramos la pantalla, pintándola de blancho
          g.setColor(0xFFFFFF);

          g.fillRect(0,0,ancho,alto);

          g.drawImage(logo,ximg,yimg, Graphics.LEFT |

                  Graphics.TOP);

      }
}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

En esta primera clase ya presenta ciertas cuestiones que debemos tener siempre presentes. Lo primero que debemos resaltar es el hecho de que todas las inicializaciones de las variables y todos los cálculos que se pueda, se deben realizar en el constructor de la clase. Los terminales móviles tienen una capacidad de proceso reducida y por lo tanto, cualquier operación tiene un cierto impacto en el rendimiento del midlet. Si todas estas operaciones las hacemos antes de que la pantalla sea mostrada, el funcionamiento posterior será mucho más ágil y este tiempo extra necesario para la creación de la clase es transparente en cierta medida para el usuario y por lo tanto, la experiencia de este es el uso del midlet es mucho mejor [17-20].

Para realizar nuestra pantalla de presentación, los cálculos a realizar son tan sencillos como recoger el tamaño del *Canvas* de la pantalla y calcular el punto en el que se tiene que dibujar la imagen, para que esta parezca centrada dentro de la pantalla.

Si durante la visualización de la pantalla de presentación el usuario pulsa alguna tecla, se pasa directamente a la pantalla principal de opciones del juego que explicaremos más adelante.

El método *paint* del juego lo único que hace es dibujar la imagen en la pantalla de terminal, después de limpiar esta. El resultado dentro del emulador es el siguiente:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

2.4 PANTALLAS BÁSICAS BASADAS EN CANVAS

A continuación, vamos a ver algunas pantallas del juego, que sin ser la pantalla principal del mismo, tendrán ciertas características y funciones que son importantes y que conviene remarcar. Son pantallas basadas en *Canvas*, con alguna animación simple. En el caso concreto que nos ocupa, su función no es más que servir de pantalla de configuración o de puntuación, pero implementan ciertas características que pueden ser muy útiles en el desarrollo de los juegos.

Para comenzar, vamos a ver la pantalla de configuración.

La pantalla de configuración del nivel de dificultad del juego, está contenida dentro de la clase Nivel. Esta pantalla en un objeto Canvas, que presenta una lista con el número de bombas que hay ocultas en el juego, y por lo tanto, cuanto mayor sea el número de bombas, mayor será la dificultad del juego. Para hacer esta pantalla, y como primera aproximación a la creación de listas de opciones más atractivas que las que proporcionan los terminales J2ME dentro de sus implementación, tendremos una lista de elementos, pero en este caso, será la clase la que controle totalmente el proceso de pintado de la lista. De momento, nos conformaremos con que la lista muestre en negrita el elemento seleccionado [21]. A partir de esta implementación y viendo el sencillo funcionamiento de esta pantalla, podemos utilizar nuestra imaginación para hacer listas tan complicadas y atractivas como queramos. Para confeccionar esta lista, debemos tener claro que hay que controlar los siguientes elementos:

- Proceso de dibujo de los elementos de la lista
- Proceso de control y dibujo del elemento seleccionado
- Control de la transición de un elemento selecciona a otro
- Gestión de las acciones del usuario

A parte de estos elementos, podríamos incluir un hilo o *thread* que realice ciertas operaciones sobre el listado, aumentando así su vistosidad. Típicamente, podríamos hacer alguna animación sobre los elementos de la lista, especialmente sobre el elemento seleccionado en cada momento.

El siguiente listado muestra el código fuente de la clase Nivel:

```
import javax.microedition.lcdui.*;

public class Nivel extends Canvas implements CommandListener {
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
private Buscabombas buscabombas = null; private Opciones opcio-
nes = null; private Command salir = null;
private String msg = "";
private Font f1 = Font.getFont(Font.FACE_MONOSPACE, Font.STYLE_BOLD, Font.SIZE_MEDIUM);
private Font f2 = Font.getFont(Font.FACE_MONOSPACE, Font.STYLE_PLAIN, Font.SIZE_MEDIUM);
int ancho;
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
int alto;
private int selected = 0; private String[] items = null;;
private int[] niveles = null;

public Nivel(Buscabombas buscabombas, Opciones opciones) {
    this.opciones = opciones; this.buscabombas = buscabombas;

    ancho = this.getWidth(); alto = this.getHeight();
    niveles = Recursos.getNiveles(); items = new
    String[niveles.length]; for (int i=0; i<niveles.length; i++){
        items[i] = niveles[i] + " bombas";
    }
    selected = buscabombas.getNivel();
    setCommandListener(this);
    salir = new Command(Recursos.getString( Recursos.ID_SALIR), Com-
                mand.EXIT, 1);
    addCommand(salir);
}
public void commandAction(Command command, Displayable displayable)
                {
    buscabombas.mostrar(opciones);
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
}

public void paint (Graphics g){ g.setColor(255, 255, 255);
    g.fillRect(0, 0, ancho, alto); g.setColor (0, 0, 0);

    int y = 3; g.setFont(f2);
    for (int i=0; i<items.length; i++){ if (i == selected){
            // Pinto el elemento seleccionado g.setFont(f1); g.draw-
            String(items[i], 2, y,
                    Graphics.TOP | Graphics.LEFT); g.setFont(f2);
        } else { g.drawString(items[i], 2, y,
                    Graphics.TOP | Graphics.LEFT);
        }
        y += f2.getHeight()+2;
    }
}

public void keyPressed (int tecla){ tecla = getGameAction(tecla);

    // Hacia abajo
    if (tecla == Canvas.RIGHT || tecla == Canvas.DOWN){ if (selected < items.length-1){
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
            selected++;
            repaint(); this.serviceRepaints(); return;
        }
    }
    // Hacia arriba
    if (tecla == Canvas.LEFT || tecla == Canvas.UP){ if (selected > 0) {
            selected--; repaint();
            this.serviceRepaints(); return;
        }
    }

    // Gestión del click
    if (tecla == Canvas.FIRE){ buscabombas.setNi-
        vel(selected); buscabombas.mostrar(opciones);
    }
}

                }
```

Dentro del método paint de este listado, vemos como debemos dibujar todos los elementos y comprobar cuál es el elemento seleccionado para dibujarlo con alguna característica especial. En este caso, se cambia la fuente de texto con la que se escribe dicho elemento, pero podríamos ponerle una imagen a la izquierda indicando que es el elemento seleccionado o hacer que dicho elemento realice una determinada animación, que lo diferencie del resto.

Otro punto a destacar es la gestión de la pulsación de las teclas por parte del usuario (método *keypressed*). En este método, se mueve la selección dentro de la lista para abajo o para arriba en

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

función de la tecla que se pulse, y si se pulsa la tecla de fuego (*fire*), se toma como selección definitiva el elemento selecciona en ese momento y se va directamente a la pantalla de opciones principales de nuestro juego. En este caso en concreto, se modificará el nivel de dificultad del juego a través de la modificación del número de bombas escondidas [22-25].

2.5 PANTALLA CON UNA ANIMACIÓN MUY SIMPLE

A continuación vamos a ver una pantalla con una animación muy sencilla, que nos servirá para tener un primer acercamiento al desarrollo de animaciones y al uso de hilos o *threads*. La pantalla es cuestión, dentro de nuestro juego será la pantalla correspondiente a la opción de *Acerca de*, es decir, la pantalla que presente al desarrollador del juego. En nuestro caso, esta pantalla mostrará una imagen, el nombre del juego y el nombre del desarrollador. El código de esta pantalla es el siguiente:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
import javax.microedition.lcdui.*;
import java.io.IOException;
public class AcercaDe extends Canvas implements Runnable{

    private Buscabombas buscabombas = null; private Menu menu =
    null;
    private boolean salir = false;
    private Thread thread; private Image logo;

    Font f1 = Font.getFont(Font.FACE_SYSTEM,
                Font.STYLE_BOLD,Font.SIZE_MEDIUM);
    Font f2 = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,Font.SIZE_SMALL);
    int ancho; int alto;
    String tit1 = "BUSCABOMBAS";
    String tit2 = "Manuel J. Prieto (2003)"; int anchoTit2;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
int anchoTit1;

int  xf1;  int  yf1;

int xf2; int yf2;



int ximg; int yimg;


int x;

int incremento=1;

/** Constructor */

public AcercaDe(Buscabombas buscabombas, Menu menu) { this.buscabombas = buscabombas;

    this.menu = menu;

    try{

        logo = Image.createImage("/bomba.png");

    }catch(IOException ioe){ ioe.printSta-

        ckTrace();

    }

    ancho = this.getWidth();
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
    alto = this.getHeight();
    anchoTit2 = f2.stringWidth(tit2); anchoTit1 =
    f1.stringWidth(tit1); xf1 = (int)((ancho-anchoTit1)/2); yf1 =
    (int)alto/2;
    xf2 = (int)((ancho-anchoTit2)/2);
    yf2 = yf1+f1.getHeight()+5;
    ximg = (int)((ancho-logo.getWidth())/2); yimg = (int)((yf1-
    logo.getHeight())/2);



    x=xf1;

    thread = new Thread(this); thread.start();
}

public void run(){ while (!salir){
        this.repaint(); this.serviceRepaints(); x+=incremento;
        if (x==xf1+anchoTit1) incremento = -1; if (x==xf1) incremento=1;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        try{
            thread.sleep(50);
        }catch(InterruptedException ie){ ie.printStackTrace();
        }
    }
    buscabombas.mostrar(menu);
}

protected void keyPressed(int keyCode){ salir = true;
}
protected void paint(Graphics g) {
    // Borramos la pantalla, pintándola de blancho g.setColor(0xFFFFFF); g.fi-
    llRect(0,0,ancho,alto);

    g.drawImage(logo,ximg,yimg, Graphics.LEFT | Graphics.TOP);

    g.setColor(0x0000FF);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        // Título principal de la pantalla de presentación
        g.setFont(f1); g.drawString(tit1,xf1,yf1,
                Graphics.LEFT | Graphics.TOP);
        // Título principal de la pantalla de presentación
        // Si el texto es muy grande, se disminuye el
        // tamaño de la fuente y el texto if (anchoTit2 > ancho){
            tit2 = "Manuel J. Prieto"; anchoTit2 = f2.stringWidth(tit2);
            xf2 = (int)((ancho-anchoTit2)/2);
        }
        g.setFont(f2); g.drawString(tit2,xf2,yf2,
                Graphics.LEFT | Graphics.TOP);
        g.setColor(0xFFFFFF); g.fillRect(x,yf1,3,f1.getHeight());

        // Hacemos un recuadro g.setColor(0x0000FF); g.drawRect(1,1,ancho-
        3,alto-2);

    }
}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El resultado de este código se muestra en la siguientes figuras, en la que se muestra la pantalla de presentación del juego. Como vemos en la figura de la izquierda, el texto *Manuel J. Prieto (2003)* es demasiado grande para mostrarlo en la pantalla y es recortado. Este hecho está controlado dentro del método *paint*, tal como se comenta dentro del propio código. Sin embargo, como se puede ver en la imagen de la derecha, las dimensiones de la pantalla del terminal nos permiten mostrar el texto completo, sin recortar. Esto es un ejemplo claro de lo que hablábamos al comienzo del capítulo, en el que se comentaba que debíamos de tener siempre presente la adaptación de nuestros midlets a las características de cada terminal. En este caso, la adaptación es muy sencilla y se puede hacer en tiempo de ejecución dicha adaptación [26-30], pero en la mayoría de las situaciones, las adaptaciones serán más complicadas o imposibles, y tendremos que realizar varias *versiones* del midlet, adaptadas a diferentes tipos de terminales. Por ejemplo, si tenemos imágenes como la que hemos visto en la pantalla de presentación, tendremos que tener varias versiones con diferentes tamaños, para los diferentes grupos de terminales. Una solución dinámica podría ser la de incluir en la distribución del midlet, versiones de la imagen de diferentes tamaños, y en tiempo de ejecución recoger el tamaño del *Canvas* y cargar la imagen correspondiente. Esta solución presenta un gran problema que la hace inviable en la mayoría de los casos y es que el tamaño del fichero *jar* a distribuir sería demasiado grande. Sería demasiado grande desde varios puntos de vista:

Cuanto mayor sea el tamaño del fichero *jar*, mayor será el tiempo de descarga del mismo, con todo lo que esto implica

Muchos terminales tienen límites de tamaño para los *midlets*, de tal forma, que si se intenta descagar un *midlet* con un tamaño superior al límite, obtendremos un error y el *midlet* no se instalará

El espacio de almacenamiento disponible en los terminales para los *midlets* es limitado y por lo tanto, cuanto mayor sea el *midlet*, menos espacio dejará libre y más posibilidades hay de que no se pueda instalar por falta de espacio de almacenamiento

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En las figuras anteriores vemos una zona blanca sobre la M del título en el caso de la figura de la izquierda y vemos una zona del mismo color del fondo sobre la primera A del título en el caso de la figura de la derecha. Bien, esta franja del color del fondo de la pantalla será el único elemento animado de la pantalla de presentación y lo que hará será moverse de izquierda a derecha y viceversa sobre el texto del título del juego. Es una animación muy simple, pero servirá como primer acercamiento y punto de partida para empezar al modificar el código que se proporciona y crear animaciones más complejas y vistosas.

En el código del canvas correspondiente al *canvas* de la pantalla de presentación que se ha presentado anteriormente, vemos que se inicializan todas las variables posibles dentro del constructor de la clase. Como ya hemos comentado, esto se hace porque todos estos valores son usados en el método *paint* cada vez que se le llama, y se pueden calcular desde el primer momento, por lo que si demoramos su cálculo y lo hacemos dentro del método *paint*, haremos los mismos cálculos, con los mismos resultados cada vez que se invoque al método *paint*, mientras que en este caso estas operaciones se realizan sólo una vez. Debemos tener siempre en la cabeza que el objetivo último de los juegos desarrollados en J2ME es la ejecución de los mismos en un dispositivo móvil, con unos recursos muy limitados en cuanto a memoria y capacidad de proceso se refiere. Por esta razón debemos optimizar el código todo lo que podamos y siempre que podamos. En el ejemplo que nos ocupa, al realizar una animación, estamos llamando constantemente al método *paint* por lo que la mejora de rendimiento de nuestra aplicación al inicializar todos los valores en el constructor será lo suficientemente rentable como para optar por esta solución.

Otro punto a destacar del ejemplo anterior es la implementación de la interfaz *Runnable* y el uso de *Threads* o hilos de ejecución para realizar la animación del título. Como vemos, el método central de todo el proceso será el método *run* que será el método que se ejecute y en el que debemos crear el bucle que actualice y gestione constantemente todo el proceso correspondiente con el *canvas*, especialemente con la animación. Dentro del método *run* cabe destacar la invocación del método *repaint*, que indica al sistema que debe repintar el *canvas* (la pantalla de presentación en este caso) y también cabe destacar el método *serviceRepaints* que obliga al sistema a repintar (a cumplir con lo ordenado en el método *repaint*) de forma inmediata, dando prioridad total

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

a esta acción. El método *serviceRepaints* es muy importante si queremos conseguir que las animaciones en nuestros juegos funcionen correctamente y el resultado sea agradable para el usuario y profesional. Para finalizar el proceso que se realiza constantemente dentro del método *run* tenemos la orden de dormir o para el *thread.* Si no hacemos esta pequeña pausa, la animación irá demasiado rápida y no conseguiremos el efecto deseado. El código anterior está preparado para ser ejecutado dentro del emulador por lo que la pausa puede ser demasiado larga para la capacidad de proceso de un terminal real. Aún así, esta pausa no será de la misma cantidad de tiempo en todos los terminales, ya que no todos los terminales tienen la misma capacidad de proceso. En un futuro ya veremos cómo solucionar este tipo de situaciones y como adaptarnos dinámicamente a la capacidad de proceso del terminal.

Cuando se pulsa una tecla, el método *keyPressed* establece la variable que controla el bucle constante del método *run* a *true* de tal forma que se finaliza el proceso. Es decir, el proceso continuo que tiene lugar dentro de nuestra clase de presentación es parado al pulsar una tecla del terminal. Dentro del método *run*, una vez que finaliza el proceso principal, se avisa al *midlet* (clase principal de la aplicación) de este hecho para que muestre el menú principal de la aplicación.

Por último, el método *paint* de la pantalla de presentación, lo único que hace es dibujar cada elemento en su lugar correspondiente dentro de la pantalla cada vez que es invocado. La posición de elemento animado es lo único variable a lo largo de la ejecución. El método *paint* se podría mejorar desde el punto de vista del rendimiento haciendo que el cálculo que se hace para comprobar si el texto *Manuel J. Prieto (2003)* se puede mostrar o debe ser recortado, se podría hacer un única vez dentro dentro del constructor.

2.6 LÓGICA Y PANTALLA PRINCIPAL DEL JUEGO

Por fin vamos a ver la última clase del juego, que tendrá la lógica del mismo y además presentará las técnicas básicas del desarrollo de los juegos que hemos denominado estáticos.

Hemos denominada a la clase Juego y por supuesto, extiende a la clase *Canvas*. Desde el punto de vista del desarrollo de los juegos en general, los puntos que debemos destacar en esta clase comienzan en el constructor. Como ya hemos comentado, hacemos todas las inicializaciones posibles dentro del constructor, para que una vez que se comienza a jugar de verdad, el funcionamiento sea más dinámico, ya que ahorramos gran cantidad de tiempo, al tener todo lo posible precalculado. En este caso, realizamos ciertas operaciones en función del tamaño de la pantalla del terminal, de tal forma que la pantalla principal del juego se adapta de forma totalmente automática a las características del terminal sobre el que se está ejecutando.

Dentro del constructor, creamos la imagen de la que obtendremos el objeto *Graphics* que nos servirá para dibujar en cualquier punto del código. El código en el que obtenemos este objeto es el siguiente:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
img = Image.createImage(anchoCanvas,altoCanvas);
off = img.getGraphics();
```

A partir de este momento, cualquier operación gráfica se realiza sobre el objeto *off* y al final, en el método *main* lo único que tendremos que hacer es dibujar la imagen *img* que hemos ido componiendo en los diferentes métodos de la clase.

En nuestro caso, optamos por este tipo de funcionamiento del proceso de dibujo, para que los diferentes métodos de la clase puedan dibujar. Es decir, es consecuencia de una decisión simplemente de diseño del software. Sin embargo, esta técnica es muy utilizada dentro del campo del desarrollo de juegos, para implementar lo que se denomina *buffer* doble o *double buffering*. Esta técnica evita que al dibujar directamente sobre la pantalla se produzcan parpadeos. Si dibujamos todo en una imagen, sin mostrar nada por la pantalla y una vez compuesta la imagen se muestra esta por pantalla, el resultado es mucho más satisfactorio. Hay terminales móviles que ya implementan de forma interna esta técnica y que por lo tanto no necesitan que se use esta técnica para evitar los parpadeos de pantalla. La clase *Canvas* nos provee un método para comprobar si el terminal ya implementa la técnica del *double buffering* en la implementación de *Graphics*. Una implementación típica del proceso de dibujo, en base a este hecho es la siguiente:

```
protected void paint(Graphics g) {

    if (this.isDoubleBuffered()){

            // Dibujamos directamente sobre g
    } else {

            // Creo la imagen, obtengo el objeto Graphics

            // Dibujos fuera de la pantalla

            // Dibujo la pantalla compuesta

}
```

Otro punto a destacar dentro la clase Juego, es la gestión de las acciones del usuario, es decir, cómo responde el midlet a las pulsaciones de las teclas del usuario. Evidentemente, dentro de un juego es básico que las acciones del usuario sean bien gestionadas y que este reciba una respuesta ágil y rápida a sus acciones. Cuando el usuario pulsa una tecla, este evento se recoge dentro del método *keyPressed*, que recibe como parámetro el código de la tecla pulsada. Este código se debe comparar con las constantes que provee la clase *Canvas*, para comprobar qué tecla se ha pulsado.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Para recuperar el valor de la tecla pulsada, se debe utilizar el método *getGameAction*.

Una vez que hemos realizado las operaciones oportunas, en función de la tecla pulsada, repercutimos los cambios en la pantalla del terminal. Para hacer esto, solicitamos el repintado de la pantalla con el método *repaint* y para que este repintado se haga de forma inmediata, se invoca al método *serviceRepaints*.

Desde el punto de vista de la lógica de este juego en particular, tenemos las siguientes estructuras de datos básicas:

- campo – Será una matriz de dos dimensiones que simboliza el contenido de todas las celdas del juego. Si el valor de una celda es –1, indicará que posee una bomba y si el valor de distinto de –1, indicará el número de bombas que posee la celda alrededor de si misma
- estado – Será una matriz de dos dimensiones que mapea las celdas del juego y su contenido indica si la celda en cuestión ha sido destapada por el jugador o si permanece cubierta
- x – Contiene la posición en el eje de abscisas de cada una de las celdas. Esta información es calculada al inicio y se usa en el dibujo de las celdas a lo largo de todo el juego
- y – Contiene la posición en el eje de ordenadas de cada una de las celdas. Esta información es calculada al inicio y se usa en el dibujo de las celdas a lo largo de todo el juego

Al comenzar el juego, se crear el número de bombas adecuado y se calculan todos los valores de las celdas dentro de la variable *campo*.

Cuando el usuario mueve el cursor, se repinta la celda que estaba seleccionada hasta ese momento tal y como debe ser repitanda y se pinta la nueva celda seleccionada, para denotar su nuevo estado.

Al hacer *click* sobre una celda, en función del contenido de la misma, pueden ocurrir varias cosas:

- Si la celda en cuestión no oculta una bomba, se muestra su contenido. Si se destapa la última celda que no oculta una bomba, el juego finalizará, se mostrará un mensaje de felicitación y se almacenará la puntuación alcanzada
- Si la celda en cuestión oculta una bomba, el juego finalizará y se mostrará el contenido de todas las celdas y se almacenará la puntuación alcanzada

La puntuación que se almacena, será el número de celdas destapadas, y el número de bombas ocultas. Lo que se almacena no es un número, es una cadena de texto con la información recién descrita.

El siguiente listado muestra la clase Juego:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
import javax.microedition.lcdui.*;

import java.util.Random;
import javax.microedition.rms.RecordStore; import javax.microedi-

tion.rms.RecordEnumeration;


public class Juego extends Canvas implements CommandListener {

    private Buscabombas buscabombas = null; private Menu menu =

    null;

    private boolean enPausa = false; private Command

    salir = null; private Command pausa = null; int mx =

    0;

    int my = 0;
    int lado = 0; int n = 0;


    // Contenido de cada una de las celdas int campo[][];

    int x[];

    int y[];
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
// Estado de cada una de las celdas (mostrada o no)
int estado[][];

int ladoCampo; Font f = null;
Font f2 = null;

int altoCanvas; int anchoCanvas;

int selecx = 0; int selecy = 0;
int antx = 0; int anty = 0; int
felx = 0; int fely = 0;

Image img; Graphics off;

private final static int BOMBA = -1; private final static int TAPADO = 0;
private final static int DESTAPADO = 1;
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
private boolean fin = false;

private int destapados=0; private int aDestapar = 0;

public Juego(Buscabombas buscabombas, Menu menu) { this.buscabombas = buscabom-
    bas;
    this.menu = menu;
    salir = new Command(Recursos.getString( Recursos.ID_SALIR), Com-
            mand.EXIT, 1);
    pausa = new Command(Recursos.getString( Recursos.ID_PAUSA),Command.SCREEN,1);
    this.addCommand(salir);
    this.addCommand(pausa); this.setCommandListener(this);

    altoCanvas = this.getHeight(); anchoCanvas =
    this.getWidth();

    img = Image.createImage(anchoCanvas,altoCanvas); off = img.getGraphics();


    f = Font.getFont(Font.FACE_MONOSPACE, Font.STYLE_PLAIN,Font.SIZE_SMALL);
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
f2 = Font.getFont(Font.FACE_MONOSPACE,
        Font.STYLE_BOLD,Font.SIZE_LARGE);

felx = (int)((anchoCanvas-f2.stringWidth( Recursos.getString( Recur-
        sos.ID_FELICITACION)))/2);
fely = (int)((anchoCanvas-f2.getHeight())/2);

  // Tamaño del lado de las celdas lado = 12;

  // Cálculo del número de celdas que podemos tener if (altoCanvas > anchoCanvas){
      n = (int)anchoCanvas/lado;
  }else{
      n = (int)altoCanvas/lado;
  }
  // Ancho del campo de celdas ladoCampo = n *
  lado;
  // Celdas a destapar para finalizar el juego aDestapar = (n*n)-Recursos.getNiveles()
        [buscabombas.getNivel()];
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
mx = (int)(anchoCanvas-ladoCampo)/2;

my = (int)(altoCanvas-ladoCampo)/2;

x = new int[n]; y = new int[n];

campo = new int[n][n]; estado = new int[n][n];



for (int i=0; i<n; i++){ for (int j=0; j<n; j++){

        estado[i][j]=Juego.TAPADO; campo[i][j]=0;

  }

}

Random r = new Random(System.currentTimeMillis()); int j=0;

// Calculamos el contenido de las celdas

while (j < Recursos.getNiveles() [buscabombas.getNivel()]){

        int a = Math.abs(r.nextInt() % n); int b = Math.abs(r.nextInt()

        % n); if (campo[a][b] != Juego.BOMBA){

            campo[a][b] = Juego.BOMBA;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                    sumaBomba(a-1,b-1);
                    sumaBomba(a-1,b); sumaBomba(a-1,b+1); suma-
                    Bomba(a,b-1); sumaBomba(a,b); sumaBomba(a,b+1);
                    sumaBomba(a+1,b-1); sumaBomba(a+1,b); suma-
                    Bomba(a+1,b+1);

                    j++;
                }
        }
        for (int i=0; i<n; i++){ x[i] = mx+(i*lado);
            y[i] = my+(i*lado);
        }

        pintaCampo();
    }
    protected void paint(Graphics g) {
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
    // Dibujamos la imagen que hemos ido componiendo
    // en la pantalla g.drawImage(img,0,0,Graphics.LEFT|Graphics.TOP);
}
protected void keyPressed(int keyCode){ keysManager(keyCode);
        repaint();
        this.serviceRepaints();
}
private void keysManager(int keyCode){ if (destapados == aDesta-
    par){
        // Se ha finalizado la partida, al descubrir
        // todas las celdas sin bomba oculta menu.inicializar(buscabombas);
        buscabombas.mostrar(menu);
        return;
    }
    keyCode = getGameAction(keyCode); antx = selecx;
    anty = selecy;
    if (keyCode == Canvas.LEFT){
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
    if (selecx > 0){
        selecx--;
    } else {
        if (selecy > 0){ selecx = n-1; selecy--;
        }
    }
    pintaCelda(antx,anty,0); pintaCelda(selecx,selecy,1);
}
if (keyCode == Canvas.RIGHT){ if (selecx == n-1){
        if (selecy < n-2){ selecx = 0; selecy++;
        }
    }else{
        selecx++;
    }
    pintaCelda(antx,anty,0); pintaCelda(selecx,selecy,1);
}
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
if (keyCode == Canvas.UP){ if (selecy > 0){
        selecy--;
    }
    pintaCelda(antx,anty,0); pintaCelda(selecx,selecy,1);
}
if (keyCode == Canvas.DOWN){ if (selecy != n-1){
        selecy++;
    }
    pintaCelda(antx,anty,0); pintaCelda(selecx,selecy,1);
}
if (keyCode == Canvas.FIRE){
    if (campo[selecx][selecy] == Juego.BOMBA){
        // Al pinchar en una bomba, se acaba la partida finPartida();
    }else{
        if (estado[selecx][selecy] != Juego.DESTAPADO) destapados++;
    }
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

```
estado[selecx][selecy]=Juego.DESTAPADO;
if (campo[selecx][selecy] == 0){
    // Al pinchar en una celda "en blanco"
    // compruebo el entorno
    // para descubrir todas las celdas adyacentes
    // también en blanco. compruebaEntorno(selecx, selecy);
}

if (destapados == aDestapar){ finPartida();
    // Dibujo un mensaje de felicitación off.setColor(0xFFFFFF); off.fi-
    llRect(0,fely-5,anchoCanvas, 10+f2.getHeight()); off.setCo-
    lor(0x0000FF); off.setFont(f2); off.setColor(0x000099);
    // Dibujo el texto dos veces, desplazando un
    // poco una de ellas, para crear un
    // efecto de sombra off.drawString(Recursos.getString( Re-
    cursos.ID_FELICITACION),  felx+1, fely+1,
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                Graphics.LEFT | Graphics.TOP);
                off.drawString(Recursos.getString( Recursos.ID_FELICITACION),
                felx, fely, Graphics.LEFT | Graphics.TOP);
            }
        }
    }
    private void sumaBomba(int x, int y){
        if (x >= 0 && x < n && y >= 0 && y < n && campo[x][y] != Juego.BOMBA) {
                campo[x][y]++;
        }
    }
    private void finPartida(){
        for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) {
                estado[i][j] = Juego.DESTAPADO; pintaCelda(i, j, 0);
            }
        }
        // Guardo la puntuación
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
    try{
        RecordStore rs = RecordStore.openRecordStore( "buscabombas", true);
        int i = 0;
        // Si hay más de 10 puntuaciones almacenadas
      .// se borra la más antigua de ellas if (rs.getNumRecords() > 10) {
            RecordEnumeration re = rs.enumerateRecords(
              null,null,false);
            if (re.hasNextElement()){ rs.deleteRecord(re.nextRecordId());
            }
        }
        String a = destapados + "(" + Recursos.getNiveles() [busca-
            bombas.getNivel()] + ")";
        rs.addRecord(a.getBytes(),0,a.length());
    }catch (Exception ex){ ex.printStackTrace();
    }
}
public void commandAction(Command cmd, Displayable dis) {
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
    if (cmd == salir) {
        menu.inicializar(buscabombas); buscabombas.mostrar(menu);
    }
    if (cmd == pausa){
        // Ponemos el juego en modo de pausa enPausa = true; menu.iniciali-
        zar(buscabombas); buscabombas.mostrar(menu);
    }
}
private void pintaCampo(){ off.setColor(0xFFFFFF); off.fillRect(0,0,anchoCan-
    vas,altoCanvas); off.setColor(0x000000);
    off.setFont(f);

    off.drawRect(mx,my,ladoCampo,ladoCampo);


    int myl = my+ladoCampo; int mxl = mx+lado-
    Campo; for (int i=0; i<n; i++){
        off.drawLine(x[i], my, x[i], myl);
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
            off.drawLine(mx, y[i], mxl, y[i]);

        }
        int l2 = lado-4;
        for (int i=0; i<n; i++){ for (int j=0; j<n; j++){

                if (estado[i][j] == Juego.TAPADO){ off.setColor(0x888888); off.fillRect(x[i]+2,y[j]+2,l2,l2);

                }else{

                    off.setColor(0x000000); off.drawString(""+campo[i][j],x[i]+2,y[j]-1,

                  Graphics.LEFT|Graphics.TOP);

                }

            }

        }
        off.setColor(0x000000); off.fillRect(x[selecx],y[selecy],lado,lado);

    }

    private void pintaCelda(int i, int j, int selec){ int l2 = lado-4;

        off.setColor(0xFFFFFF);

        off.fillRect(x[i]+1, y[j]+1, lado-1, lado-1);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
if (selec == 0){
    if (estado[i][j] == Juego.TAPADO) { off.setColor(0x888888);  off.fill-

        Rect(x[i] + 2, y[j] + 2, l2, l2);

    }
    else {
        if (campo[i][j] != 0){
            if (campo[i][j] != Juego.BOMBA){ off.setColor(0x000000);

                off.drawString("" + campo[i][j],

                    x[i] + 2, y[j] - 1,

                    Graphics.LEFT | Graphics.TOP);

            }else{
                // Dibujo la bomba off.setColor(0x000000); off.fi-

                llArc(x[i]+2,y[j]+3,

                    lado-3,lado-3,0,360); off.drawLine(x[i]+6,y[j]+2,x[i]+6,y[j]+1);

                off.drawLine(x[i]+6,y[j]+1,x[i]+8,y[j]+1);

            }
        }else{
            off.setColor(0xFFFFFF); off.fillRect(x[i]+1,y[j]+1,lado-1,lado-1);

        }
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

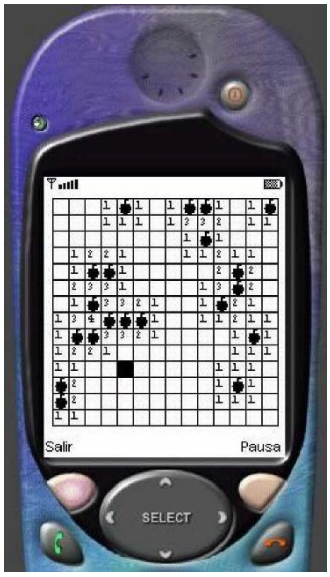
```
            }
        } else { off.setColor(0x000000);
            off.fillRect(x[i],y[j],lado,lado);
        }
    }
    public void continuar() { enPausa = false;
    }
    public boolean enPausa() { return enPausa;
    }
    private void compruebaEntorno (int i, int j){ int l2 = lado-4;
        for (int a = i - 1; a <= i + 1; a++) {
            for (int b = j - 1; b <= j + 1; b++) {
                // La celda actual no se procesa if (a != i || b != j){
                    // Si estoy fuera de los límites del campo,
                    // no hago nada
                    if (a < n && a >= 0 && b < n && b >= 0 &&
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                    estado[a][b] == Juego.TAPADO &&
                    campo[a][b] == 0) { estado[a][b] =
                Juego.DESTAPADO; off.setColor(0x888888);
                off.fillRect(x[a] + 2, y[b] + 2, l2, l2); pintaCelda(a,b,0);
                compruebaEntorno(a, b);
            }
        }
        }
    }
}
```

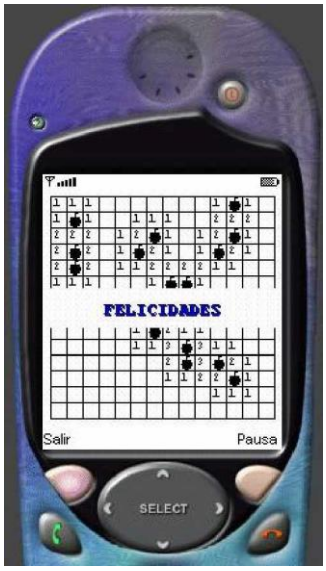
A continuación tenemos algunas pantallas del juego en diferentes terminales, para mostrar el resultado final de juego.



ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



3 JUEGOS DINÁMICOS

3.1 INTRODUCCIÓN

En este capítulo vamos a estudiar las técnicas básicas para el desarrollo de lo que denominaremos juegos dinámicos. Al contrario de los juegos estáticos vistos anteriormente, los juegos dinámicos se caracterizan por una presencia masiva de elementos móviles. Es decir son juegos en los que tendremos una serie de elementos (denominados *sprites* de forma general, sean móviles o no.) que se moverán por la pantalla del juego a lo largo de la ejecución del mismo. El movimiento de los *sprites*, conlleva consigo una serie de cuestiones a tener en cuenta como es la detección de colisiones entre *sprites* y la gestión de dichas colisiones [31-35].

A lo largo del capítulo veremos el desarrollo de un juego J2ME correspondiente a lo que podríamos denominar como un simulador de un minigolf. El desarrollo de este juego nos va a permitir explicar cómo se realizan los procedimientos básicos en el desarrollo de juegos J2ME relativamente complejos, de tal forma que podamos hacer, entre otras cosas y sin recurrir a las facilidades de MIDP 2.0 las siguientes operaciones:

- Gestión mejorada de la carga y distribución de imágenes (*sprites*)
- Gestión de la transparencia de los *sprites*
- Detección de colisiones entre *sprites* y gestión de las mismas
- Uso de *threads* o hilos de ejecución para hacer animaciones

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

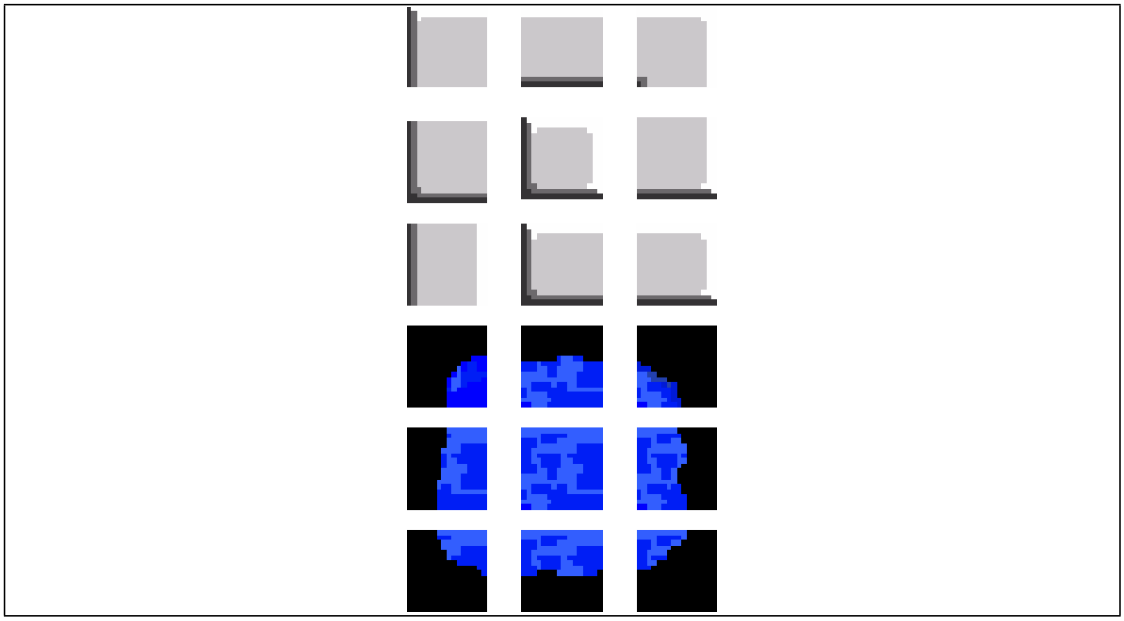
•

3.2 ALMACENAMIENTO Y DISTRIBUCIÓN DE LOS SPRITES

Como sabemos, en el desarrollo de *midlets* J2ME es básico controlar el tamaño final del fichero *jar* que debemos construir, ya que este fichero se ha de descargar posteriormente por red desde un terminal y además, algunos terminales presentan restricciones en cuanto al tamaño de los ficheros *jar* que son capaces de manejar. Uno de los elementos que se incluyen en este fichero y que suelen aumentar su tamaño preocupantemente, son las imágenes. Si bien el título de este juego hace referencia a *sprites*, en realidad vamos a ver cómo tratar las imágenes que componen la parte esencial de dichos *sprites [36]*.

En un primer momento lo que tendremos será una serie de imágenes con el formato *png*. Todas estas imágenes, estarán almacenadas en ficheros separados, y cada uno de estos ficheros contendrá la cabecera característica de los ficheros gráficos con formato *png*. Bien, la solución propuesta para ahorrar espacio, es la inclusión de todas las imágenes dentro de un único fichero. De esta forma tendremos una única cabecera y por lo tanto el fichero *jar* resultante será menor.

En nuestro caso, tendremos una serie de imágenes que corresponden a los elementos simples con los que compondremos el campo de minigolf. La siguiente figura muestra los *sprites* que se usan en el juego para componer los elementos que componen el campo de juego. En concreto tendremos bloques que nos permiten componer obstáculos en el campo y tendremos una serie de *sprites* para situar zonas de agua dentro del campo.

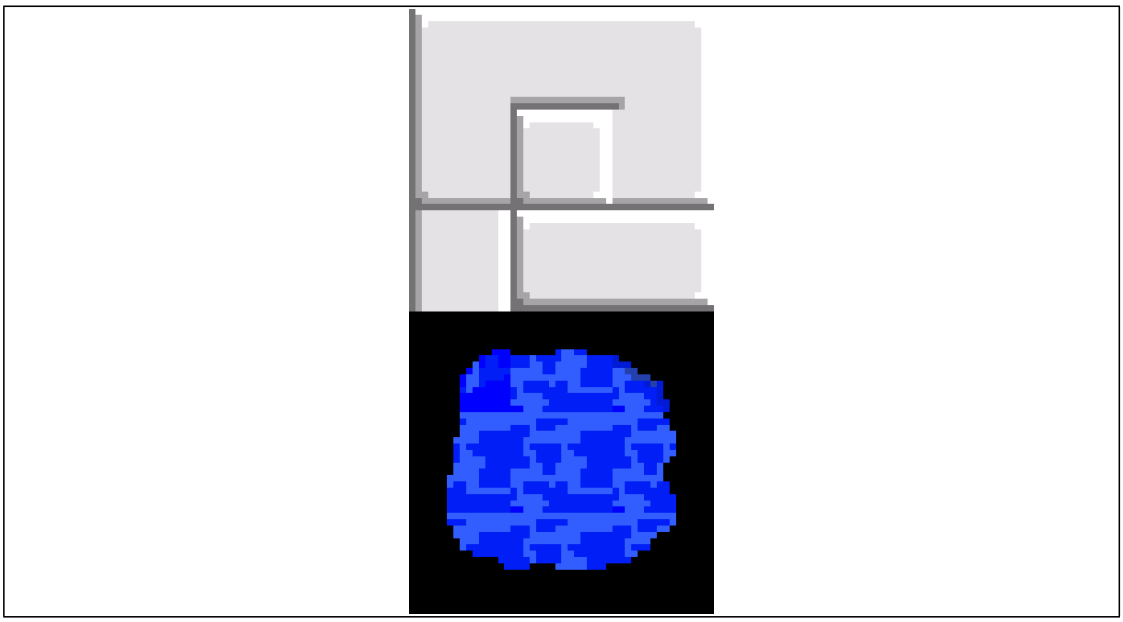


ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Como vemos, tenemos 18 imágenes independientes que en total suponen una carga de almacenamiento de 6240 bytes (algo más de 6 Kb).

El fichero con todas estas imágenes unidas, ocupa 1264 bytes (algo más de 1 Kb), por lo que el ahorro de espacio es de 5 Kb, lo que supone un ahorro de espacio del 80%. Como vemos, esta técnica es rentable en lo que a espacio se refiere y por lo tanto, nuestro objetivo ha sido cumplido. Un punto a tener en cuenta es que los *sprites* (en cualquier juego y de forma muy especial en los juegos móviles) se configuran a partir de imágenes de un tamaño relativamente pequeño, y por lo tanto la cabecera correspondiente al formato de imagen del fichero tiene un impacto importante sobre el tamaño final del fichero [37]. La imagen que contiene todas las imágenes es la siguiente:



Por supuesto, el uso de esta solución conlleva un cierto trabajo extra, tanto en tiempo de desarrollo (tendremos que unir todas las imágenes en una) como en tiempo de ejecución, ya que tendremos que dividir la imagen completa en sus componentes dentro del código del *midlet*. Si hacemos esta operación durante el periodo de inicialización del mismo, el usuario no se percatará de dicho tiempo extra y por lo tanto la solución será perfecta [38].

3.2.1 EL PROCESO DE SEPARACIÓN DE LAS IMÁGENES

Para separar las imágenes, dentro de la solución propuesta en este caso, se ha creado una clase denominada *SpritesManager*, que se encargará de cargar la imagen que contiene el conjunto de los *sprites* y extraer de la misma todas las imágenes que contiene. Estas imágenes se usarán

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

más adelante en el proceso de creación de objetos de la clase *Sprite*, de tal forma que estos objetos contengan tanto la imagen del *sprite*, así como otra información adicional, útil para la gestión de los *sprites*.

Bien, el código de la clase *SpritesManager* es el siguiente:

```
import java.io.IOException;
import javax.microedition.lcdui.Image; import javax.microedi-

tion.lcdui.Graphics;

public class SpritesManager {
    private Image[] imagenes = new Image[18];

    public SpritesManager() {

      Image img = null; try{

          img = Image.createImage("/Sprites.png");

      }catch (IOException ex){ ex.printStack-

          Trace();

      }

      int ind = 0;

      for (int i = 0; i < 3; i++) { for (int j = 0; j < 6; j++) {

            imagenes[ind] = Image.createImage(16, 16); Graphics g =

            imagenes[ind].getGraphics(); g.drawImage(img, i*-16, j*-16,

            Graphics.TOP | Graphics.LEFT);

            ind++;

          }

      }
  }
  public Image getImage (int i){ return imagenes[i];
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
    }
  }
```

En nuestro caso concreto, todas las imágenes tienen las mismas dimensiones, cuestión que simplifica bastante el proceso. De hecho, esta restricción la presenta también la solución que incluye la especificación 2.0 de MIDP para hacer exactamente lo mismo, es decir, almacenar los *sprites* en un único fichero y separar estas imágenes en tiempo de ejecución [39-43].

Siguiente dentro del ámbito de nuestro caso concreto, nuestras imágenes tienen todas 16 *pixels* de ancho por 16 *pixels* de alto. Como veíamos anteriormente, tenemos un total de 18 imágenes y ese por lo tanto la clase *SpritesManager* deberá ser capaz de manejar exactamente ese número de imágenes. Una vez carga la imagen principal desde el fichero con todas las imágenes destinadas a los *sprites* incluidas dentro de ella, dibujaremos sobre cado uno de los objetos *Image* que contiene la clase *SpritesManager,* que posteriormente será un *sprite*, sólo aquella parte de la imagen completa que corresponde al *sprite* en cuestión. Para conseguir esto, lo único que hacemos es situar el punto de dibujo de la imagen completa en el lugar adecuado.

Una vez que tenemos separadas todas las imágenes ya podemos recuperarlas a través del método *getImage*, que veremos cómo se utiliza más tarde para construir los *sprites*.

3.3 LA CLASE SPRITE

La clase *Sprite* es básicamente una abstracción de una imagen, que además nos permite controlar la situación o posición en la que debe dibujarse el *Sprite* y comprobar las colisiones con otros *Sprites*. Además, dentro de la clase *Sprite* veremos cómo pintar imágenes con transparencias. Esta funcionalidad no la proporcionan las primitivas de dibujo del objeto *Graphics* dentro de la especificación 1.0 de MIDP y es de suma importancia en el desarrollo de juegos. Como vemos, la clase *Sprite* contiene algunas de las características imprescindibles para el desarrollo de juegos. Más adelante, veremos cómo utilizar esta clase dentro de nuestro juego para obtener resultados satisfactorios [44].

Para empezar, vamos a ver cómo gestiona la clase *Sprite* la imagen que debe mostrar y la posición en la que debe dibujar la misma. Los parámetros que contiene la clase *Sprite* son:

```
private int x = 0;

private int y = 0; private Image img; pri-

vate int imgId; private int mascara[][];
```

Las dos primeras variables corresponden a la posición de la esquina superior izquierda de la imagen base del *sprite* en la pantalla. Debemos tener en cuenta, que es posible que con la transparencia, lo que realmente se vea en la pantalla del terminal no comience en este punto. Este punto definirá el marco que envuelve toda la imagen. Evidentemente, la variable *img* almacena la imagen que contiene y gestiona el *sprite*. La variable *imgId* simplemente nos permite

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

identificar la imagen asociada al *sprite*, lo que nos permitirá controlar ciertas situaciones. Es decir, sabiendo que tenemos un conjunto limitado de imágenes, en nuestro caso 18, que pueden componer un *sprite*, podemos asignar un número a cada una de las imágenes y así poder hacer alguna gestión alternativa. En el caso de que varios *sprites*

alberguen la misma imagen, todos tendrán el mismo identificador. En el caso concreto de nuestro minigolf, usamos este valor para detectar si el *sprite* contiene una imagen correspondiente a una zona de agua, o una imagen correspondiente a un obstáculo. Por último, la variable máscara define una serie de rectángulos, que serán las zonas de dibujo de la imagen que nos permite realizar y gestionar las transparencias. Esta variable la veremos en detalle un poco más adelante [45-50].

Para que la clase *Sprite* sea más flexible, se le han proporcionado tres posibles constructores que son similares, pero que difieren en la forma de obtener la imagen del *sprite*. Los constructores son los siguientes:

```
public Sprite(Image img, int mascara[][],

              int x, int y, int imgId) public Sprite(int mascara[][],

int x,

              int y, int imgId)

public Sprite(String fichero, int mascara[][], int x, int y, int imgId)
```

El primer constructor que se muestra, tiene como parámetros un objeto de tipo *Image*, una matriz de dos dimensiones que será la máscara de transparencia, el punto que define la posición inicial del *sprite* (a través de sus valores x e y) y por último, un parámetro que especifica el tipo de *sprite*, tal y cómo explicamos antes. El segundo de los constructores, no tiene imagen. En principio esto puede parecer un poco absurdo, pero puede ser útil para definir zonas, en principio invisibles, pero que nos van a permitir controlar las colisiones de otros *sprites* contra estas zonas invisibles.

El último constructor es capaz de obtener la imagen del *sprite* a partir del fichero cuyo nombre recibe como primer parámetro.

3.3.1 TRANSPARENCIAS EN SPRITES

El uso de transparencia en los *sprites* es algo básico si queremos que un juego de los que hemos denominado dinámicos sea realmente profesional y vistoso. En el caso de nuestro

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

campo de golf, hemos visto anteriormente que tenemos una serie de *sprites* que definen zonas de agua, en los que el fondo de la imagen es negro y debemos tener en cuenta que nuestro campo de golf será verde. Las siguientes imágenes muestran la diferencia entre el uso de un *sprite* con transparencia y un *sprite* sin transparencia.

Como se puede comprobar, la solución realizada sin transparencia es poco útil y nuestros *sprites* serían bastante pobres sin transparencia. En nuestro caso, usando la transparencia, la zona de agua nos permite ver el campo de golf.

Bien, una vez que vemos que las transparencias en los *sprites* son imprescindibles para desarrollar un juego y que la especificación 1.0 de MIDP no nos proporciona una implementación de la gestión de imágenes transparentes, tendremos que desarrollar nuestra propia solución. Tenemos varias opciones para implementar transparencias:

Dibujar los *sprites* a partir de las primitivas que nos proporciona la clase *Graphics*, como por ejemplo *drawLine*. Esta solución presenta dos problemas graves, que la hacen viable sólo en casos muy concretos. Por una parte es una solución con un rendimiento muy bajo en tiempo de ejecución y por supuesto, realizar *sprites* de calidad (imágenes complejas) a partir de primitivas básicas es un trabajo ímprobo y complicado.

Dibujar la imagen compleja a partir de imágenes menores que componen la primera. Es decir, partiendo de la imagen real del *sprite* que queremos usar, la dividimos en tantas imágenes como sea necesario para que al pintar todas estas imágenes por separado, consigamos la imagen original con transparencia. Esta solución también es en cierta medida compleja y cuando las imágenes tienen muchas zonas redondeadas, la solución puede llevarnos a manejar un gran número de imágenes.

Por último, podemos implementar una solución en la que definamos una serie de zonas de

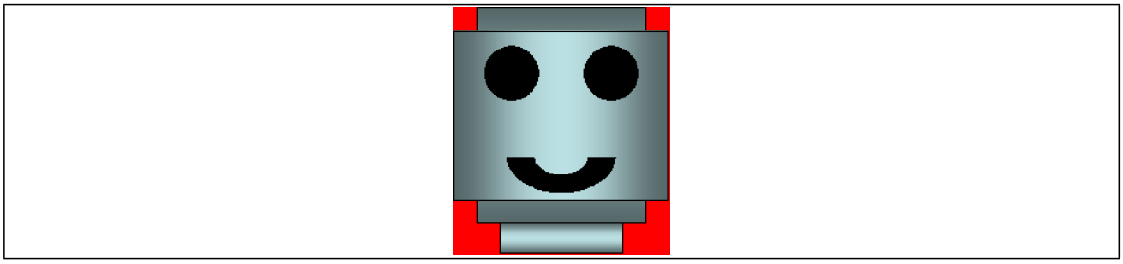
ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

dibujo rectangulares (también se podrían definir circulares) y utilizar el método *setClip* de la clase *Graphics* para que al dibujar la imagen, dicha acción sólo tenga efecto sobre la pantalla en las zonas definidas. Esta solución es un buen compromiso entre dificultad de implementación, necesidad de recursos en tiempo de ejecución y el resultado obtenido y por lo tanto, se ha optado por ella para implementar nuestro juego de ejemplo.

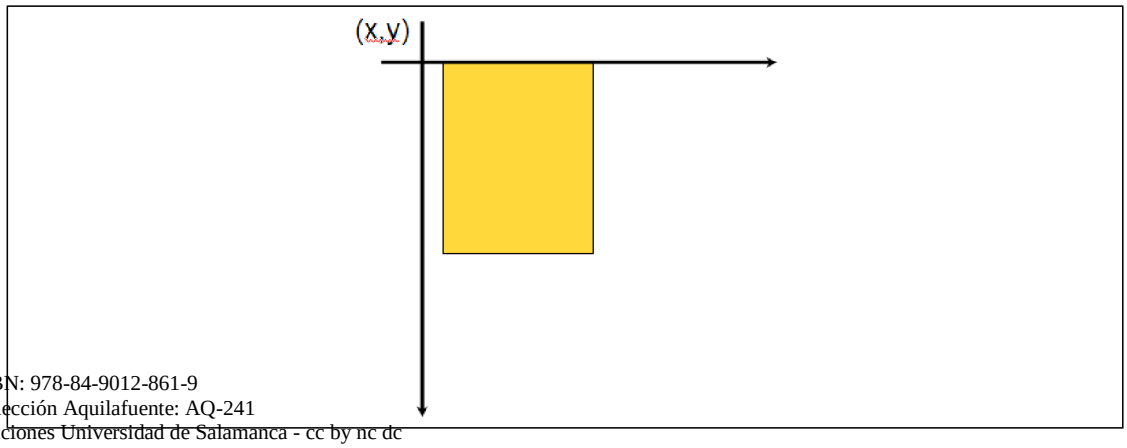
Ahora que ya tenemos una idea general de cómo implementar la transparencia de los *sprites*, vamos a desarrollar esta técnica en detalle. Debemos destacar que el hecho de que las imágenes para dispositivos móviles sean típicamente pequeñas y *pixeladas*, favorece en gran medida esta solución.

Ya hemos hablado de la variable máscara, que será una matriz de dos dimensiones de enteros. Esta variable lo que contiene es la definición de rectángulos, que compondrán aquellas zonas de la imagen que deben ser vistas. Con el siguiente ejemplo gráfico vamos a ver claramente este concepto.

Supongamos que queremos dibujar este simpático robot y la zona que en la imagen es de color rojo, debe ser transparente y debemos ver a través de ella.



La imagen anterior será la imagen *png* que tenemos almacenada dentro de la variable *img* en el objeto *Sprite* en cuestión. Para empezar, definiremos las zonas no transparentes de la imagen, es decir, lo que hemos denominado máscara. Para definir uno de estos rectángulos, lo que hacemos es configurar el punto en el que se sitúa la esquina superior-izquierda del rectángulo, y definimos su ancho y su alto. Por ejemplo, la siguiente imagen muestra un rectángulo que define una zona no transparente de la imagen.

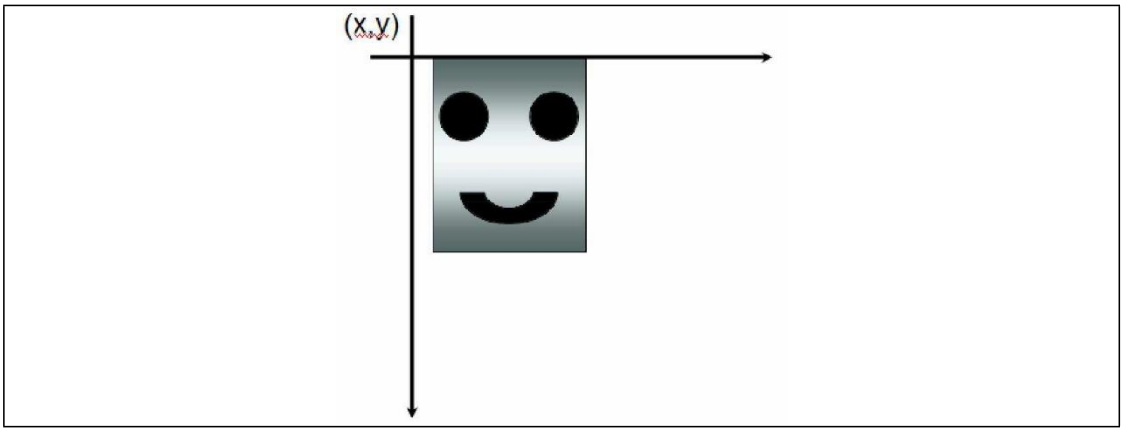


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

El rectángulo amarillo define una zona no transparente de la imagen, por lo que tendremos una primera versión de la variable máscara como:

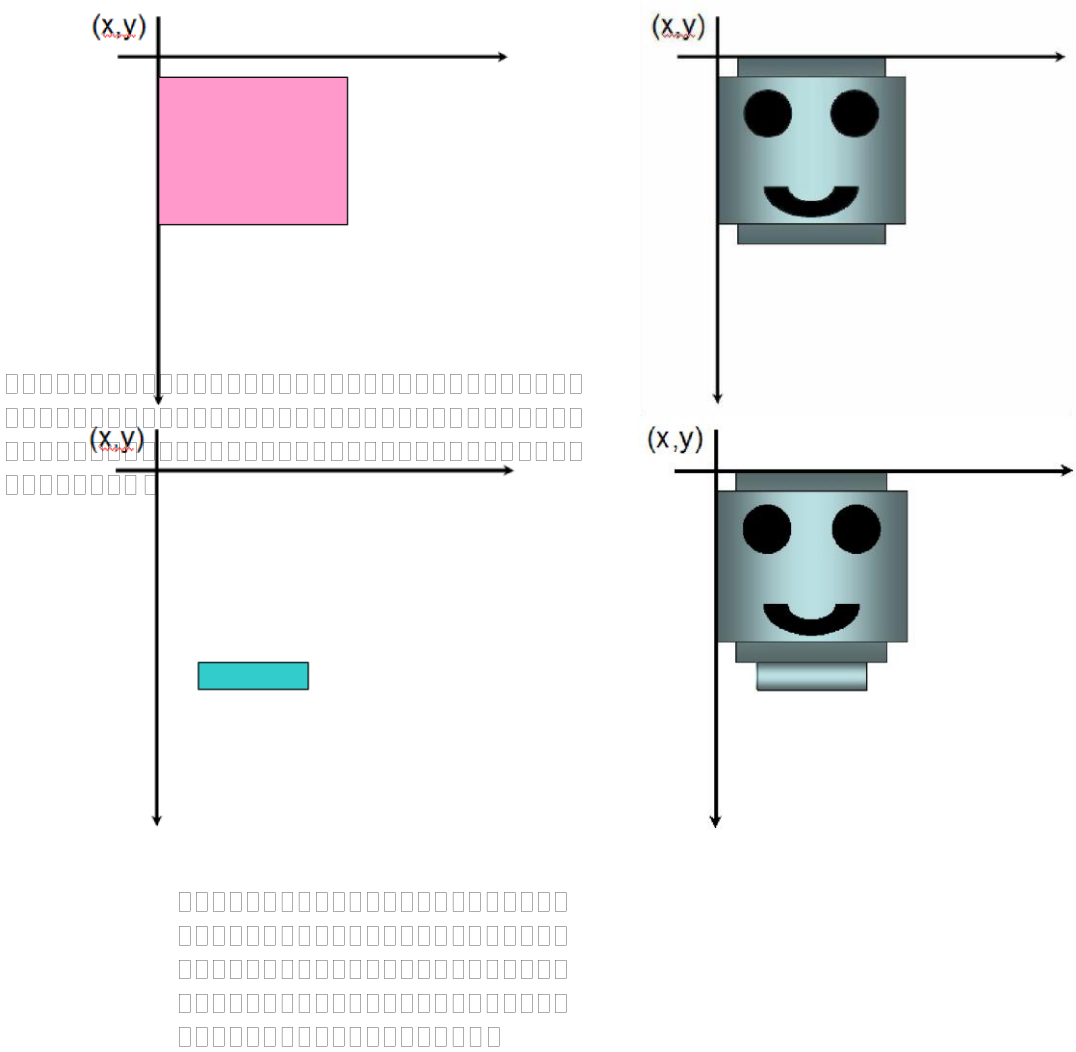
```
mascara = {{x+1,y,5,8}};
```

En el código anterior, 5 es el ancho del rectángulo amarillo y 8 su altura. Si ahora definimos esta zona como aquella única parte de la pantalla en la que tendrán efecto las operaciones de dibujo y dibujamos la imagen en el punto *(x,y)*, el resultado será el siguiente:



ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Si definimos alguna zona más y repetimos la operación, es decir, establecemos un rectángulo como zona visible a través de *setClip* y volvemos dibujar la imagen. El resultado sería el siguiente.



De esta forma, tendremos definitivamente en la pantalla toda la imagen del *sprite* y será transparente en aquellas zonas no presentes en algunos de los rectángulos de la máscara. El código final para definir la máscara sería algo como lo siguiente.

```
mascara = {{x+1,y,5,8}{x,y+1,8,5}{x+2, y+8,4,1}};
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Por supuesto, estos valores son aproximados, pero nos da una idea de cómo sería el funcionamiento. Ahora que ya tenemos claro cómo es el proceso, vamos a ver el código real que debemos escribir para que esto funcione.

Cuando una *sprite* no tiene zonas transparentes, es decir, las imágenes se deben dibujar enteras, en nuestra implementación, el valor de la máscara será *null*. Típicamente no tendremos transparencias cuando nuestro *sprite* sea cuadrado o rectangular, como lo son los obstáculos de nuestro campo de golf [51].

En nuestra clase *Sprite* hemos definido un método *paint* que recibe como parámetro un objeto *Graphics* y que dibuja la imagen del *sprite* en cuestión sobre dicho objeto *Graphics*. Este método será invocado cada vez que se quiera dibujar el *sprite*. El código del método *paint* es el siguiente.

```
public void paint (Graphics g){

   int ancho = g.getClipWidth(); int alto =

   g.getClipHeight();
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        if (mascara == null){
            g.drawImage(img, x, y, Graphics.LEFT |
                Graphics.TOP);
          }else{
            for (int i=0; i<mascara.length;i++){ g.setClip(x+mas-
                cara[i][0],y+mascara[i][1], mascara[i][2],mascara[i][3]);
                g.drawImage(img, x, y,
                  Graphics.LEFT | Graphics.TOP);
            }
            g.setClip(0,0,ancho,alto);
        }
            }
```

Como podemos comprobar, si la máscara tiene valor *null* se dibuja la imagen tal cual en el punto definido como posición del *sprite* y no hacemos nada más. Por el contrario, si tenemos zonas definidas en la variable máscara, para cada una de estas zonas (que es un rectángulo), definimos una zona de recorte con *setClip* igual a dicha zona y dibujamos la imagen en su posición. Al final del proceso, reestablezco la zona de recorte o de *clipping* a toda la pantalla, que es similar a no definir ninguna zona de recorte. De esta manera, tendremos implementado un sistema de transparencia de *sprites*.

3.3.2 DETECCIÓN DE COLISIONES ENTRE SPRITES

Otro punto esencial dentro de la programación de juegos es la detección de colisiones entre *sprites*. Es decir, debemos detectar cuando dos *sprites* chocan o colisionan en la pantalla debido a su movimiento. También es posible que sólo uno de los *sprites* esté en movimiento y choque contra otro *sprite* estático. La acción a realizar cuando dos *sprites* chocan dependerá de

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

cada juego en concreto. Por ejemplo, si un *sprite* que simula un disparo choca contra un personaje del juego, quizás debamos eliminar del juego el *sprite* del personaje [52]. En nuestro caso concreto, el simulador de minigolf tendrá colisiones entre la pelota y los obstáculos, las zonas de agua y el hoyo. En cada caso, deberemos saber contra qué elemento exactamente estamos colisionando, ya que el resultado de la colisión será distinto. Si la bola choca con un obstáculo, rebotará en el mismo y cambiará la trayectoria de su movimiento. Por el contrario si la bola choca con una zona de agua, la bola se pierde y vuelve a la posición que tenía antes de ser golpeada. Por último, si la colisión es entre el *sprite* que hace las funciones de bola y el que hace las funciones de hoyo, habremos conseguido nuestro objetivo y pasaremos al siguiente hoyo.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

La funcionalidad correspondiente a la detección de colisiones también recae sobre la clase *Sprite*. En concreto, la clase *sprite* tiene los métodos necesarios para realizar todo el proceso. Estos métodos son:

```
public int checkCollision(Sprite sprite);

public int[][] getMascaraColision();
```

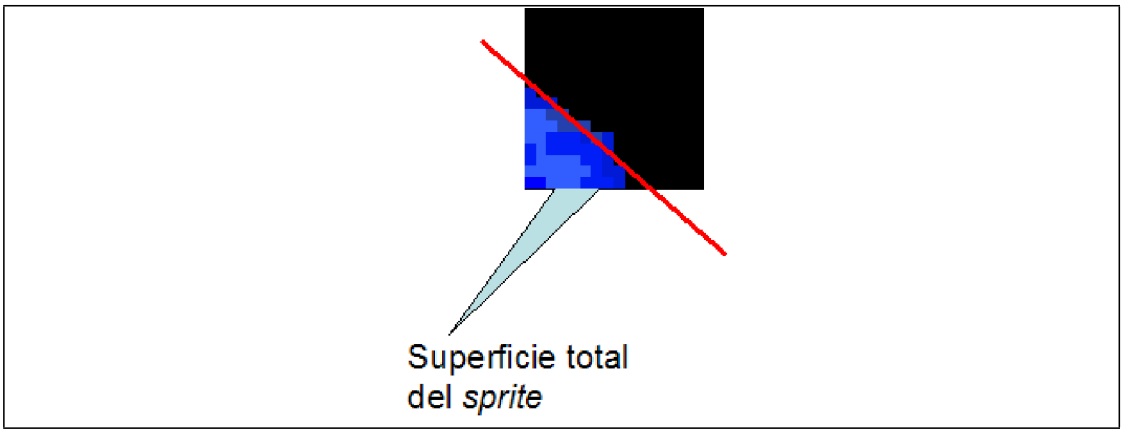
El primer método recibe como parámetro otro *sprite* y comprueba si el *sprite* en cuestión al que pertenece el método está colisionando con el *sprite* que se recibe como parámetro. El método devuelve un valor entero que en nuestro caso indica contra qué zona del *sprite* se ha chocado. Esto es especialmente útil cuando la colisión es contra un objeto cuadrado o rectangular como ocurre en alguno de nuestros casos. Otra opción sería que este método retornará simplemente *true* o *false* para indicar si hay o no colisión. En nuestra implementación, si el valor de retorno es –1, podemos asumir que no hay colisión entre los dos *sprites [53]*.

El segundo método, el método *getMascaraColision*, nos retorna una matriz bidimensional de enteros, que representa las zonas del *sprite* que corresponden con las zonas de colisión. El concepto es similar al usado para las transparencias. De hecho, en nuestra implementación, la zona no transparente del *sprite* es exactamente la zona de colisión del mismo.

Se definen zonas de colisión, por la misma razón por la que tenemos zonas transparentes y zonas no transparentes en los *sprites*. En nuestros *sprites* para las zonas de agua, existen zonas circulares. Debemos realizar la funcionalidad de tal forma que si la bola en su movimiento entra dentro de lo que es el *sprite* de agua pero no toca la zona de agua propiamente dicha del *sprite* (zona no transparente), no debemos detectar ninguna colisión. Si no lo hiciéramos de esta forma, el usuario obtendría un funcionamiento no esperado del sistema, ya que no refleja lo que ocurre en el mundo real. La siguiente imagen aclara lo que sería una zona de colisión dentro de un *sprite*.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc



Dentro de la imagen anterior, sólo deberíamos de detectar una colisión cuando esta ocurra con la zona a la izquierda de la línea roja. A la derecha de dicha línea, el *sprite* es transparente y por lo tanto, esa zona del *sprite* no existe para el usuario [54].

Como decíamos anteriormente, se definen zonas de colisión dentro del *sprite* para saber cuál es la zona inexistente para el usuario. Las zonas de colisión se definen mediante zonas rectangulares, de tal forma que la suma de las superficies de todos estos rectángulos es la superficie de colisión del *sprite*. En nuestro caso concreto y en la mayoría de los casos, la zona de colisión será la misma zona que la zona no transparente del *sprite*. Este hecho nos permite que la variable *mascara* de la clase *Sprite* de la que hemos hablado anteriormente, nos sirva para ambos propósitos, tanto para realizar las transparencias como para detectar las colisiones.

En este punto, es sencillo llegar a la conclusión de que podemos comprobar si dos *sprites* colisionan, comprobando si sus zonas de colisión se solapan. Para realizar esto, habrá que comprobar si alguno de los rectángulos que definen la zona de colisión de un *sprite*, tiene alguna superposición con alguno de los rectángulos de colisión del otro *sprite*.

El siguiente código muestra el contenido del método *getMascaraColision*.

```
public int[][] getMascaraColision(){

    int masc[][];

    if (mascara != null){

        masc = new int[mascara.length][4];
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        for (int i = 0; i < masc.length; i++) { masc[i][0] = mas-
            cara[i][0] + this.getX();
            masc[i][1] = mascara[i][1] + this.getY();
            masc[i][2] = mascara[i][2];
            masc[i][3] = mascara[i][3];
        }
    }else{
        masc = new int[1][4]; masc[0][0] = x;
        masc[0][1] = y;
        masc[0][2] = img.getWidth(); masc[0][3] =
        img.getHeight();
    }
    return masc;
}
```

Como podemos comprobar en el código anterior y como era de esperar, si la máscara de colisión es *null* (debemos recordar que esto indicaba que el *sprite* no tenía zonas transparentes) toda la superficie del *sprite* es objeto de colisión. Por lo tanto, si la variable *mascara* es *null*, se retornará un rectángulo igual a toda la superficie del *sprite*. Por el contrario,

si la máscara de colisión no es *null*, se retornarán aquellas zonas de la pantalla que son sensibles a colisiones debido al *sprite*. Este hecho es importante, ya que la zona de colisión real se calcula en cada momento, en función de la posición del *sprite*. Esto se hace gracias a las siguientes líneas de código del método *getMascaraColision*.

```
        masc[i][0] = mascara[i][0] + this.getX();
        masc[i][1] = mascara[i][1] + this.getY();
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

El método *checkColision* es, como ya hemos dicho, el que realiza realmente la detección de las colisiones entre *sprites*. El siguiente código corresponde a dicho método.

```
public int checkCollision(Sprite sprite){
    int sprt1[][] = this.getMascaraColision(); int sprt2[][] =
    sprite.getMascaraColision();

    for (int i=0; i<sprt1.length; i++){ for (int j=0;
       j<sprt2.length; j++){
          boolean colision = false;
          if (sprt1[i][0]+sprt1[i][2] == sprt2[j][0] &&
                ((sprt1[i][1] >= sprt2[j][1] &&
                sprt1[i][1] <= sprt2[j][1] + sprt2[j][3])
                    ||
               (sprt1[i][1]+sprt1[i][3] >= sprt2[j][1] && sprt1[i][1]+sprt1[i][3] <=
                           sprt2[j][1] + sprt2[j][3]))){
             //colision contra la parte izquierda del
             //sprite colision = true;
             return CalculadorTrayectoria.DERECHA;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
}
if (sprt1[i][0] == sprt2[j][0] + sprt2[j][2] &&
        ((sprt1[i][1] >= sprt2[j][1] &&
        sprt1[i][1] <= sprt2[j][1] + sprt2[j][3])
              ||
      (sprt1[i][1]+sprt1[i][3] >= sprt2[j][1] && sprt1[i][1]+sprt1[i][3] <=
              sprt2[j][1] + sprt2[j][3]))){
    //colision contra la parte derecha del
    //sprite colision = true;
    return CalculadorTrayectoria.IZQUIERDA;
}
if (sprt1[i][1] == sprt2[j][1] + sprt2[j][3] &&
        ((sprt1[i][0] >= sprt2[j][0] &&
        sprt1[i][0] <= sprt2[j][0] + sprt2[j][2])
              ||
      (sprt1[i][0]+sprt1[i][2] >= sprt2[j][0] && sprt1[i][0]+sprt1[i][2] <=
              sprt2[j][0] + sprt2[j][2]))){
    //colision contra la parte abajo del
    //sprite
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                colision = true;
                return CalculadorTrayectoria.ARRIBA;
            }

            if (sprt1[i][1]+sprt1[i][3] == sprt2[j][1] &&
                ((sprt1[i][0] >= sprt2[j][0] &&
                sprt1[i][0] <= sprt2[j][0] + sprt2[j][2])
                    ||
                (sprt1[i][0]+sprt1[i][2] >= sprt2[j][0] && sprt1[i][0]+sprt1[i][2] <=
                    sprt2[j][0] + sprt2[j][2]))){
                //colision contra la parte arriba del
                //sprite colision = true;
                return CalculadorTrayectoria.ABAJO;
            }
        }
    }
    // No hay colisión return -1;
}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

En el código anterior aparecen una serie de constantes pertenecientes a la clase *CacularTrayectoria*, que veremos más adelante. El código anterior hace exactamente lo que hemos comentado, cada rectángulo perteneciente a la zona de colisión de un *sprite*, se compara con todos los rectángulos de la zona de colisión del otro *sprite* para detectar si coinciden en algún punto de su superficie [55].

En una versión sencilla de un detector de colisiones, bastaría con esto, pero en nuestro caso, además necesitamos saber con qué zona del *sprite* estamos colisionando (arriba, abajo, derecha o izquierda). Para hacer esto, detectamos con qué zona del rectángulo de colisión del *sprite* que se recibe como parámetro, está en contacto con nuestra zona de colisión. Por último, vemos que cuando no se detecta colisión alguna, se retorna el valor –1.

Hasta este punto, tenemos una descripción detallada de la clase *Sprite*, que contiene algunas características muy importantes. A continuación tenemos un listado completo del código de dicha clase, que será utilizada masivamente en el resto del código del juego.

```
import javax.microedition.lcdui.*;

import java.io.IOException;

public class Sprite { private int x = 0;

    private int y = 0; private Image

    img; private int imgId;


    private int mascara[][];

    public Sprite(Image img, int mascara[][],

                        int x, int y, int imgId) { this.mascara = mascara;

        this.x = x;

        this.y = y; this.img = img;

        this.imgId = imgId;

    }
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
    public Sprite(int mascara[][], int x,
                          int y, int imgId) { this.mas-
        cara = mascara; this.x = x;
        this.y = y; this.imgId = imgId;
    }

    public Sprite(String fichero, int mascara[][], int x, int y, int imgId) {
        this.mascara = mascara; this.x = x;
        this.y = y; this.imgId = imgId;
        try{
            this.img = Image.createImage(fichero);
        }catch (Exception ex){}
}

    public void moveTo (int x, int y){ this.x = x;
        this.y = y;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
}
public void paint (Graphics g){ int ancho =
   g.getClipWidth();        int        alto        =
   g.getClipHeight(); if (mascara == null){
      g.drawImage(img, x, y, Graphics.LEFT |
          Graphics.TOP);
   }else{
      for (int i=0; i<mascara.length;i++){ g.setClip(x+mas-
         cara[i][0],y+mascara[i][1], mascara[i][2],mascara[i][3]);
         g.drawImage(img, x, y,
          Graphics.LEFT | Graphics.TOP);
      }
   g.setClip(0,0,ancho,alto);
   }
}
public int getX(){ return x;
}
public int getY(){ return y;
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
}
public int[][] getMascaraColision(){ int masc[][];

   if (mascara != null){

      masc = new int[mascara.length][4];

      for (int i = 0; i < masc.length; i++) { masc[i][0] = mas-

         cara[i][0] + this.getX();

         masc[i][1] = mascara[i][1] + this.getY();

         masc[i][2] = mascara[i][2];

         masc[i][3] = mascara[i][3];
      }

   }else{

      masc = new int[1][4]; masc[0][0] = x;

      masc[0][1] = y;
      masc[0][2] = img.getWidth(); masc[0][3] =

      img.getHeight();

   }
   return masc;
}
public int checkCollision(Sprite sprite){ int sprt1[][] = this.get-

   MascaraColision();
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
int sprt2[][] = sprite.getMascaraColision();
for (int i=0; i<sprt1.length; i++){ for (int j=0;
    j<sprt2.length; j++){
        boolean colision = false;
        if (sprt1[i][0]+sprt1[i][2] == sprt2[j][0] &&
                ((sprt1[i][1] >= sprt2[j][1] &&
                sprt1[i][1] <= sprt2[j][1] + sprt2[j][3])
                    ||
               (sprt1[i][1]+sprt1[i][3] >= sprt2[j][1] && sprt1[i][1]+sprt1[i][3] <=
                    sprt2[j][1] + sprt2[j][3]))){
            //colision contra la parte izquierda del
            //sprite colision = true;
            return CalculadorTrayectoria.DERECHA;
        }
        if (sprt1[i][0] == sprt2[j][0] + sprt2[j][2] &&
                ((sprt1[i][1] >= sprt2[j][1] &&
                sprt1[i][1] <= sprt2[j][1] + sprt2[j][3])
                    ||
               (sprt1[i][1]+sprt1[i][3] >= sprt2[j][1] && sprt1[i][1]+sprt1[i][3] <=
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
            sprt2[j][1] + sprt2[j][3]))){
    //colision contra la parte derecha del
    //sprite"); colision = true;
    return CalculadorTrayectoria.IZQUIERDA;
}
if (sprt1[i][1] == sprt2[j][1] + sprt2[j][3] &&
        ((sprt1[i][0] >= sprt2[j][0] &&
        sprt1[i][0] <= sprt2[j][0] + sprt2[j][2])
            ||
      (sprt1[i][0]+sprt1[i][2] >= sprt2[j][0] && sprt1[i][0]+sprt1[i][2] <=
            sprt2[j][0] + sprt2[j][2]))){
    //colision contra la parte abajo del sprite colision = true;
            return CalculadorTrayectoria.ARRIBA;
}
if (sprt1[i][1]+sprt1[i][3] == sprt2[j][1] &&
        ((sprt1[i][0] >= sprt2[j][0] &&
        sprt1[i][0] <= sprt2[j][0] + sprt2[j][2])
            ||
                (sprt1[i][0]+sprt1[i][2] >= sprt2[j][0] &&
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                    sprt1[i][0]+sprt1[i][2] <=
                        sprt2[j][0] + sprt2[j][2]))){
                //colision contra la parte arriba del sprite colision = true;
                return CalculadorTrayectoria.ABAJO;
            }
        }
    }
    // No hay colisión return -1;
}

public int getImgId(){ return
    imgId;
}
}
```

3.4 CONSTRUCCIÓN DEL CAMPO DE JUEGO

En este apartado, veremos cómo se genera el campo de juego a partir de una matriz de datos, que indica aquello que contiene en cada una de las celdas o partes en las que podemos descomponer el campo de juego. Como es habitual en los juegos 2D (en dos dimensiones), la pantalla se divide en celdas, de tal forma que se configure el campo a partir de la definición del contenido de cada celda.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En el caso de nuestro juego de minigolf, trabajamos con un campo cuadrado de 11 celdas de ancho, por 11 celdas de alto. Nuestro juego está preparado para trabajar con un terminal cuya pantalla tiene 178 *pixels* de ancho y 180 *pixels* de alto. Por otra parte, los *sprites* que hemos definido para el juego son de forma cuadrada, con 16 *pixels* de lado. Con esta información es sencillo ver que si tenemos 11 celdas de 16 *pixels* de ancho, tenemos en total 176 *pixels* de lado en nuestro campo, que se acerca bastante a las dimensiones de la pantalla y que por lo tanto parece una dimensión razonable para nuestro campo de minigolf.

Para flexibilizar el proceso de carga de las pantallas por parte del juego y así hacer a este más potente, se ha creado una clase, que a partir de una matriz bidimesional que define los diferentes *sprites* que componen el campo de juego. La clase que se encarga de hacer esto, es

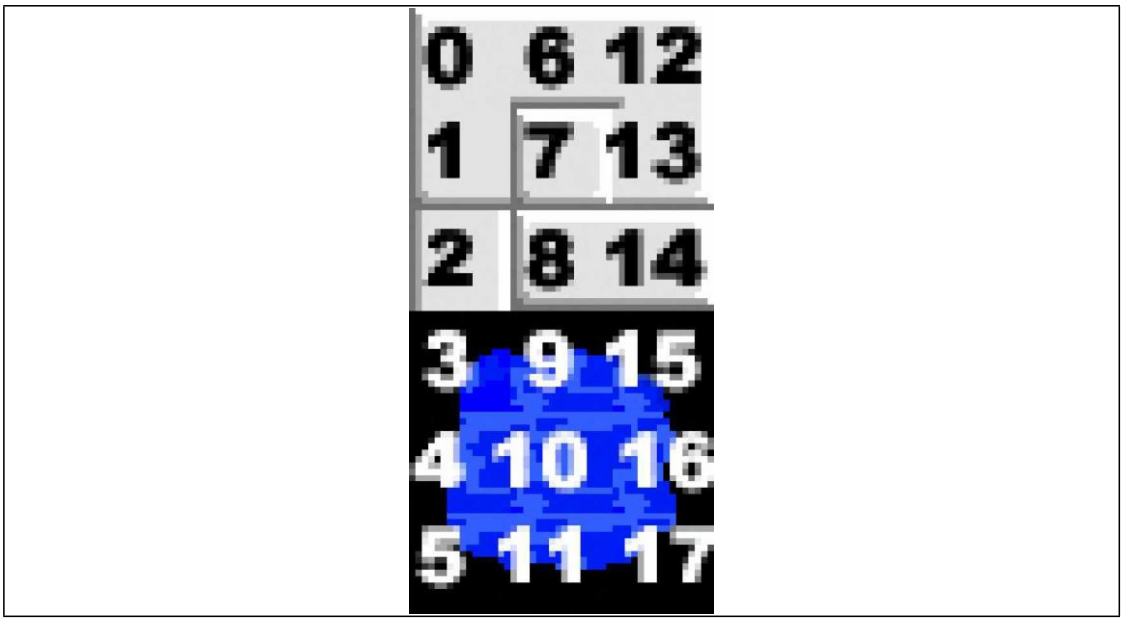
la clase *CargarCampo*. Esta clase posee un único método, que es estático y se denomina *getCampo*. Este método recibe como parámetro la matriz que define el campo y retorna una matriz con los *sprites* que componen el campo. Estos *sprites* serán los que tenemos que mantener controlados para comprobar las colisiones y dibujar el campo de juego.

Debemos tener en cuenta que evidentemente no todo el campo de juego está compuesto por *sprites* activos. Es decir, tendremos muchas zonas del campo de juego que podemos suponer vacías ya que no tienen ningún efecto sobre el juego. En nuestro caso, todas las zonas las zonas del campo que no correspondan a zonas de agua y no correspondan a obstáculos, podemos obviarlas, ya que no tendrán ningún efecto sobre nuestra bola. Para estas celdas sin efectos en el juego, que podríamos definir pasivas, no debemos comprobar las colisiones ni preocuparnos de su influencia en el juego, ya que esta no existe. Teniendo en cuenta esto, el método *getCampo* de la clase *CargarCampo*, sólo retorna en la matriz de *sprites*, aquellos *sprites* activos para el juego.

En la matriz que define el campo, los *sprites* pasivos se indican con el número –1, mientras que el resto de *sprites* se identifican con un número único. Este número se corresponde con el índice del *sprite* dentro de la clase *SpriteManager*. Este índice se asigna de manera automática durante el proceso de descomposición de la imagen que contiene todos los *sprites* necesarios para dibujar el campo. Es decir, el primer *sprite* que se extrae de la imagen contenedora tendrá el índice 0 dentro de la clase *SpriteManager*, el segundo tendrá el índice 1 y así sucesivamente. Según el código visto anteriormente para la clase *SpriteManager*, la extracción se realiza obteniendo primero todos los *sprites* de la primera columna (de arriba a abajo), después la segunda columna y así sucesivamente. Esto es importante porque determina el índice que definitivamente tendrá cada imagen.

El índice asignado a cada *sprite* dentro de la clase *SpriteManager*, será el mismo que se indicará dentro de la matriz que define el campo. La siguiente imagen muestra los índices asignados a cada *sprite* por la clase *SpriteManager* después de descomponer la imagen con todos los *sprites*.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc



A continuación vamos a ver un ejemplo del código fuente que define un campo de juego de nuestro minigolf, y la imagen que se generará en el terminal para dicho campo.

```
int tablaCampo[][]={

{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
{-1, 0,12,-1,-1, 3, 9, 9,15,-1,-1},

{-1, 1,13,-1,-1, 4,10,10,16,-1,-1},

{-1,-1,-1,-1,-1, 5,11,11,17,-1,-1},

{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},

{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},

{-1, 0, 6, 6, 6,12,-1,-1,-1,-1,-1},

{-1, 2,-1,-1,-1, 2,-1,-1,-1,-1,-1},

{-1, 2,-1,-1,-1,-1,-1,-1,-1,-1,-1},

{-1, 2,-1,-1,-1, 2,-1,-1,-1,-1,-1},

{-1, 1, 6, 6, 6,13,-1,-1,-1,-1,-1}

};
```

La imagen correspondiente para el campo definido en el anterior código nos muestra el resultado del proceso de generación del campo de juego.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Ahora que hemos explicado el proceso de generación del campo, vamos a ver el código de la clase *CargarCampo* y que hace que todo esto sea posible.

```
public class CargarCampo {
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        static int mask3[][] = {{8,10,8,7},{9,9,7,1},
            {10,8,6,1},{11,7,5,1},{13,6,3,1}};
        static int mask9[][] = {{0,7,16,9},{7,8,5,1}};
        static int mask15[][] = {{0,14,9,2},{0,11,8,3},
            {0,10,6,1},{0,9,4,1},{0,8,3,1},
            {0,7,1,1}};
        static int mask16[][] = {{0,0,8,1},{0,1,9,2},
            {0,3,10,4},{0,7,9,1},{0,8,8,3},
            {0,11,9,2},{0,13,10,3}};
        static int mask17[][] = {{0,0,10,2},{0,2,9,1},
            {0,3,7,1},{0,4,6,2},{0,6,4,1},{0,7,3,1}};
        static int mask11[][] = {{0,0,16,8},{0,8,3,1},
            {7,8,8,1}};
        static int mask5[][] = {{6,0,10,2},{7,2,9,2},
            {8,4,8,2},{10,6,6,1},{14,7,2,1},{15,8,1,1}};
        static int mask4[][] = {{6,10,10,6},{7,4,9,6},
            {8,0,8,4}};
    public static Sprite[] getCampo(int tablaCampo [][]){ int nSprites=0;
            Sprite campo[] = null;
      for (int i=0; i<tablaCampo.length; i++){
        for (int j=0; j<tablaCampo[i].length; j++){ if (tablaCampo[i][j] != -1){
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
            nSprites++;
        }
    }
}

campo = new Sprite[nSprites];

SpritesManager sprtMng = new SpritesManager(); int ind=0;

for (int i=0; i<tablaCampo.length; i++){

    for (int j=0; j<tablaCampo[i].length; j++){ if (tablaCampo[i][j] != -1){

            int c = tablaCampo[i][j]; switch (c){

                case 0:

                case 1:

                case 2:

                case 6:

                case 7:

                case 8:

                case 10:

                case 12:

                case 13:

                case 14:

                    campo[ind++] = new Sprite( sprtMng.getImage(c), null,j*16, i*16,c);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        break;
    case 3:
        campo[ind++] = new Sprite( sprtMng.getImage(c), mask3,j*16,i*16,c); break;
    case 4:
        campo[ind++] = new Sprite( sprtMng.getImage(c), mask4,j*16,i*16,c); break;
    case 5:
        campo[ind++] = new Sprite( sprtMng.getImage(c), mask5,j*16,i*16,c); break;
    case 9:
        campo[ind++] = new Sprite( sprtMng.getImage(c), mask9,j*16,i*16,c); break;
    case 11:
        campo[ind++] = new Sprite( sprtMng.getImage(c),mask11,j*16,i*16,c); break;
    case 15:
        campo[ind++] = new Sprite( sprtMng.getImage(c),mask15,j*16,i*16,c);
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                    break;
                case 16:
                    campo[ind++] = new Sprite( sprtMng.getImage(c),mask16,j*16,i*16,c);
                    break;
                case 17:
                    campo[ind++] = new Sprite( sprtMng.getImage(c),mask17,j*16,i*16,c);
                    break;
            }
        }
    }
    return campo;
  }
}
```

Al comienzo de la clase se definen una serie de variables estáticas que se corresponden con las matrices de colisión (o transparencia) de cada uno de los *sprites*. Estas variables son matrices de enteros, tal y como vimos en su día, y se llevan nombres de acuerdo al patrón *maskxx*, donde *xx* es el índice del *sprite*. Como se explicó anteriormente, algunos *sprites* no necesitan matriz de colisión ni de transparencia, por lo que la máscara que recibe el constructor del *sprite* tiene el valor *null.*

El método *getCampo*, lo primero que hace es descartar aquellas celdas cuyo índice del *sprite* es –1, ya que como hemos visto, son celdas sin importancia para el juego. Una vez hecho esto, para cada uno de las celdas a tener en cuenta, crea el *sprite* correspondiente y lo almacena en la matriz de *sprites* que retornará el método.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

3.5 OPERACIONES CON NÚMERO ENTEROS: FÍSICA DEL JUEGO

Los juegos J2ME son precisamente eso, juegos y por lo tanto suelen requerir de operaciones matemáticas más o menos complejas para simular en muchos casos ciertas leyes físicas. En Internet tenemos gran cantidad de información sobre estos temas, pensada para el desarrollo de juegos para ordenadores comunes. Toda estas información será útil y en muchos casos necesaria para el desarrollo de los juegos en J2ME, pero deberá adaptarse en mayor o menor alcance, para adaptarla a las capacidades de los entornos móviles. Las restricciones más importantes a tener en cuenta a la hora de implementar este tipo de código, es la pobre capacidad de proceso de los terminales móviles y que MIDP sólo presenta operaciones matemáticas con número enteros.

En nuestro caso, necesitamos conocer, entre otros puntos, cómo responden los objetos ante las colisiones, para simular el comportamiento de la bola de golf al rebotar contra los diferentes obstáculos. El cálculo de la trayectoria de la bola, concentra la mayor parte de las operaciones matemáticas presentes en el juego.

3.5.1 MANEJO DE SENOS Y COSENOS

Dentro de la especificación de MIDP no se contempla el cálculo de senos, cosenos o cualquier otra operación trigonométrica similar. En nuestro caso son necesarios los senos y los cosenos para el cálculo de la nueva trayectoria de la bola en caso de colisión y para la selección de la trayectoria inicial de la bola por parte del usuario.

Antes de golpear la bola, el usuario debe seleccionar la dirección inicial en la que se moverá esta. Para hacer esta operación, se le muestra al usuario una pequeña línea que parte de la bola y que se extiende en la dirección adecuada. Es decir, tendríamos una circunferencia virtual que tiene como punto central el centro de la bola (el *sprite* que representa a la bola realmente) y cada uno de los puntos de dicha circunferencia, definirá la dirección en la que se lanzará la bola. El cálculo de todos estos puntos que definen el círculo de selección de trayectoria, no se realizará en tiempo real, para optimizar el rendimiento del *midlet*. No debemos olvidar que cualquier operación, por simple que parezca, en el ámbito de los terminales móviles tiene un impacto, que debemos tener en cuenta, sobre el rendimiento de nuestro *midlet*.

Para optimizar este proceso se han precalculado todos los puntos que componen el círculo de selección de la trayectoria inicial. En concreto, se ha calculado, para cada posible punto de dicho círculo, la distancia vertical y horizontal desde el centro hasta dicho punto, de tal forma que sumando dichos valores al punto central de la bola, obtenemos el punto final de la línea que debemos dibujar para que el usuario seleccione la dirección de partida de la bola. Todos estos valores se han almacenado dentro de una variable estática que contiene 360 pares de valores. Cada uno de estos 360 elementos contiene la información referente a cada uno de los puntos del círculo, es decir, contendrá el desplazamiento vertical y el desplazamiento horizontal para el dibujado de cada punto. Estos valores están calculados de tal forma que el radio del círculo sea de 20 *pixels*.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Por otra parte, para calcular correctamente la trayectoria de la bola, debemos simular qué sucede cuando la bola choca con los límites del campo o con alguno de los obstáculos del campo. Cuando la bola colisiona con otro elemento dentro su trayectoria, la trayectoria cambia de dirección. Para calcular esta nueva dirección, debemos conocer el ángulo de entrada de la bola en la colisión, para averiguar el ángulo de salida y por lo tanto, conocer así la nueva trayectoria. Para hacer estos cálculos, necesitamos conocer los senos y los cosenos de los ángulos.

Como ya sabemos, las implementaciones estándar de J2ME no proporcionan métodos para calcular el seno o el coseno de un ángulo, por lo que tendremos que buscar una solución alternativa. La opción por la que se ha optado en nuestro caso es por precalcular los senos y cosenos de los diferentes ángulos. De esta manera, además de tener la funcionalidad que necesitamos, optimizamos el código en cierta medida.

Los valores de los senos y los cosenos están precalculados únicamente para los ángulos entre 0 y 90, ambos inclusive, ya que el valor para el resto de ángulos se puede calcular a partir de estos valores. Otro problema que nos encontramos en el manejo de los senos y los cosenos, es que estos elementos presentan valores entre 0 y 1 (ambos inclusive). Volvemos a enfrentarnos con un problema de manejo de número no enteros. Para solucionar este problema, se ha optado por tomar en el precálculo, en valor de seno multiplicado por 10.000. De esta

forma, trabajaremos siempre con número enteros (con una precisión de 4 decimales) y los resultados serán los esperados.

Todas estas operaciones serán básicas en el cálculo de la trayectoria de la bola. El siguiente fragmento de código muestra cómo se recalcula la trayectoria de la bola después de una colisión contra un obstáculo.

```
if (pared == CalculadorTrayectoria.ARRIBA ||

        pared == CalculadorTrayectoria.ABAJO){ direccionY = -direccionY;

}

if (pared == CalculadorTrayectoria.DERECHA || pared == CalculadorTrayecto-

          ria.IZQUIERDA){

                direccionX = -direccionX;

  }

if (pared == ARRIBA){ angulo = -angulo;
```

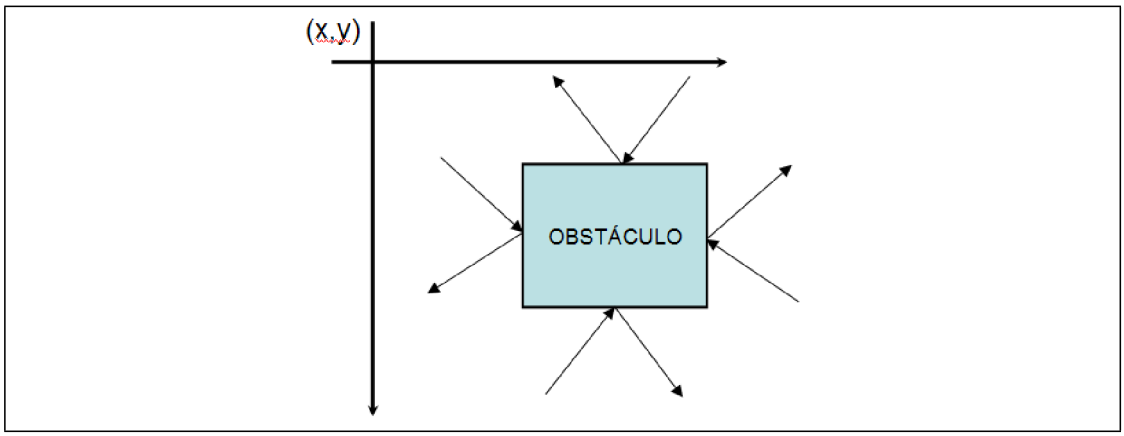
ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        yActual++;;
    }
    if (pared == ABAJO){ angulo = -
        angulo; yActual--;;
    }
    if (pared == IZQUIERDA){ if (angulo <
        0){
                    angulo = -180 - angulo;
        }else{
            angulo = 180 - angulo;
        }
        xActual++;
    }
    if (pared == DERECHA){ if (angulo <
        0){
            angulo = -180 - angulo;
        }else{
            angulo = 180 - angulo;
        }
        xActual--;
    }
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Para calcular correctamente el resultado, es necesario saber contra qué parte del bloque ha colisionado la bola. En nuestro caso, que los bloques son cuadrados, la colisión puede ser contra la parte de arriba o abajo, la izquierda o la derecha del obstáculo. Los dos primeros *if* del código anterior, comprueban si hay que cambiar la dirección de la bola. Básicamente, en una colisión con la parte inferior o superior de un bloque la dirección vertical de la bola variará, pero la horizontal permanecerá invariable. Por el contrario, en un choque con la parte izquierda o derecha de un obstáculo, la dirección horizontal variará pero la dirección vertical permanecerá inalterable. El siguiente dibujo ilustra este concepto.



El elemento esencial para recalcular la trayectoria después de una colisión, es conocer el ángulo de entrada, para calcular el ángulo de salida y en función de este ángulo de salida, podremos saber hacia qué punto se dirige la bola como resultado de la colisión. Este cálculo se hace en la segunda parte del listado de código presentado. Para calcular el ángulo de salida lo

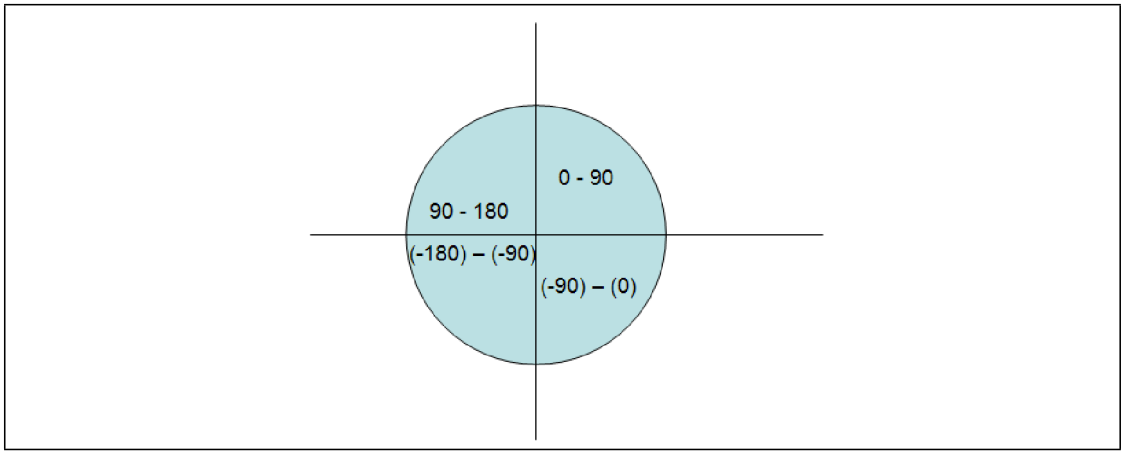
único que necesitamos es conocer el ángulo de entrada y la parte del obstáculo con el que ha colisionado la bola.

Para calcular hacia dónde se dirige la bola, una vez conocido el ángulo de salida de la colisión, tendremos que averiguar el seno y el coseno de dicho ángulo. La segunda parte del código presentado anteriormente, muestra cómo se calculan dichos valores, que como se ha mencionado anteriormente, se calculan a partir de los valore previamente conocidos del seno y el coseno de los ángulos entre 0 y 90.

En nuestro juego, usamos ángulos entre –180 y 180 grados. La siguiente imagen muestra el rango de ángulos con el que trabajamos y dónde se sitúa cada uno de los ángulos.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



En principio, el semicírculo superior se puede considerar de la forma habitual, es decir, el extremos derecho de dicho semicírculo corresponde con el ángulo 0 y el extremo izquierdo corresponde con el ángulo 180. Una vez colocados los ángulos en este semicírculo, el ángulo que podríamos definir como reflejo en el semicírculo inferior, será el valor negativo del ángulo correspondiente en la parte superior. De este manera, el extremo izquierdo del semicírculo inferior tendrá corresponderá al ángulo –180 y este valor irá creciendo hasta el valor imaginario de –0.

Una vez sistema de denominación de ángulos que se ha usado en el desarrollo del juego, vamos a mostrar el fragmento de código que muestra cómo se calculan los senos y los cosenos para cualquier ángulo.aclarado el

```
if (angulo <= 90 && angulo > 0){

    coseno = Constantes.coseno[angulo]; seno = Constantes.seno[an-

    gulo];

} else if (angulo > 0) {

    coseno = -Constantes.coseno[180-angulo]; seno = Constantes.seno[180-
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        angulo];
    } else if (angulo > -90) {
        coseno = Constantes.coseno[-angulo];
        seno = -Constantes.seno[-angulo];
    } else{
        coseno = Constantes.coseno[180+angulo]; seno = -Constan-
        tes.seno[180+angulo];
    }
```

La clase C*onstantes* contiene las variables estáticas seno y coseno, que son una matriz de valores enteros, con el valor del seno y el coseno de los ángulos entre 0 y 90.

Ahora que conocemos el seno y el coseno del ángulo de salida de una colisión, vamos a calcular el punto exacto al que se dirigirá la bola. Por lo tanto, conocido el punto de colisión (punto actual de la bola) y el punto al que se dirige, conoceremos la nueva trayectoria de la bola, ya que dos puntos definen una recta. El siguiente código muestra cómo se calcula el punto al que se dirige la bola.

```
    if ((angulo >=0 && angulo <= 180) ||
            (angulo > -91)){
        y1 = yActual - (seno / 20); x1 = xActual + (co-
        seno / 20);
    } else {
        y1 = yActual - (seno / 20); x1 = xActual - (coseno
        / 20);
    }
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

El punto destino de la bola será el punto *(x1, y1).* Vamos a explicar cómo realizamos este cálculo. A partir de un punto inicial (*x0, y0),* y de un ángulo *a,* el punto destinatario será:

- x1 = x0 + (v*cos(a))
- y1 = y0 + (v*sen(a))

En las ecuaciones anteriores, *v* determinará la longitud de la recta, es decir, la distancia entre los puntos inicial y final. En nuestro caso, hemos optado por una valor de 500 para *v,* ya que 500 puntos en la recta que define la trayectoria, nos aseguran que antes de alcanzar el último punto de dicha recta, la bola colisionará con los bordes de la pantalla y por lo tanto se recalculará una nueva trayectoria. Como habíamos dicho anteriormente, nuestras constantes para el seno y el coseno tiene dicho valor multiplicado por 10000. Por lo tanto, para obtener el valor real, tendremos que dividir por 10000. Así, tendremos que el seno y el coseno se multiplica por 500 y se divide por 10000, es decir, se divide por 20 y se optimiza el cálculo, tal y como se puede comprobar en el último fragmento de código presentado.

3.5.2 ALGORITMO DE CÁLCULO DE LÍNEA

La trayectoria que describe la bola en su movimiento, es una línea recta. Como hemos comentado en el párrafo anterior, partiendo del punto inicial y el punto final de la trayectoria, debemos calcular todos los puntos intermedios. Para hacer este cálculo usaremos uno de los algoritmos de cálculo de líneas más conocido, el algoritmo de línea de Bresenham. Este algoritmo es especialmente adecuado para nuestro caso, ya que realiza todos los cálculos con números enteros. En realidad lo que haremos será usar este algoritmo como base para crear nuestro algoritmo de cálculo de trayectorias.
El código del algoritmo de Bresenham, para rectas con pendiente positiva, es el siguiente:

```
void bres_line (int x1, int y1, int x2, int y2){
    int dx; int
    dy; int x; int
    y;
    int x_end; int p;
    int c1; int
    c2;
    dx = Math.abs(x1-x2); dy =
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        Math.abs(y1-y2); p = 2*dy-dx;

        c1 = 2*dy;
        c2 = 2* (dy - dx);
        {Se determina el punto a usar como inicial y final} if (x1 > x2){x = x2; y = y2;

            x_end = x1;

        } else { x = x1;

            y = y1;

            x_end = x2;

        }

        dibujarPixel (x,y); while

        (x<x_end){

            x = x++; if

            (p<0){

                p = p + c1;

            }else{

                y++;

                p = p + c2;

            }

            dibujarPixel(x,y);

        }

    }
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Este código nos sirve como punto de partida, pero como podemos comprobar, este algoritmo calcula los puntos intermedios de la recta sin tener en cuenta el punto inicial y el punto final que se le indican como parámetro. Es decir, dados los puntos inicial y final, puede dibujar la línea del punto inicial al final o viceversa. En nuestro caso, los puntos se deben calcular en un determinado orden, ya que serán los puntos que recorrerá la bola en su trayectoria. Por lo tanto, a partir de este algoritmo, se calcularán los valores que nos permiten calcular el siguiente punto y se calculará este, siempre teniendo en cuenta cuál es el punto inicial y cuál es el punto final.

Dentro de la clase creada en nuestro juego para calcular la trayectoria completa de la bola, tendremos un método para calcular los parámetros que nos permiten ir averiguando los puntos por los que pasa la bola. Dicho método es muy similar al algoritmo de Bresenham, ya que parte de él. A continuación se muestra el listado del método en cuestión.

```
private void calcularParametrosRecta(int x0, int y0,

                        int x1, int y1){ int dx, dy, xend;

    dx = Math.abs(x1-x0); dy = Math.abs(y1-

    y0);


    if (dx >= dy){ pendiente = 0;  p =

        2 * dy - dx; incE = 2 * dy;

        incNE = 2 * (dy - dx);

        xend = x1;

        if (direccionY == 0) { if (y0 > y1) {

                direccionY = -1;

            }

            else {

                    direccionY = 1;

            }

        }

        if (direccionX == 0) { if (x0 > x1) {

                direccionX = -1;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                }

                else {

                    direccionX = 1;

                }
            }

             }else{

            pendiente = 1;  p = 2 * dx - dy;

            incE = 2 * dx;

            incNE = 2 * (dx - dy); xend = y1;

            if (direccionY == 0) { if (y0 > y1) {

                    direccionY = -1;

                }

                else {

                    direccionY = 1;

                }

            }

            if (direccionX == 0) { if (x0 > x1) {

                    direccionX = -1;

                }

else {
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                direccionX = 1;

            }

        }

    }

}
```

Las variables *p*, *incE* y *incNE*, presentes en el método son variables globales de la clase. En este método se establecen sus valores y posteriormente se usarán para calcular el siguiente punto de la trayectoria.

El código anterior es muy sencillo, conocido el algoritmo de Bresenham. En nuestro caso, se tienen en cuenta todas la rectas, no sólo aquellas de pendiente positiva. Además, también se calcular la dirección en la que se debe mover la bola, tanto en vertical como en horizontal. Es decir, básicamente lo que se calcula son unas variables (*direccionX* y *direccionY*), que nos permite saber si en nuestra trayectoria, dado un punto (*x,y)*, debemos sumar o restar valores a *x* o *y* para calcular el siguiente.

3.6 CÁLCULO COMPLETO DE LA TRAYECTORIA DE LA BOLA

En este punto del capítulo tenemos ya los conocimientos necesarios para comprender por completo el proceso de cálculo de la trayectoria de la bola. Dentro del juego, lo que se hace es calcular un número suficientemente grande de puntos de la trayectoria de la bola, de tal forma que nos aseguremos que para el valor máximo de la fuerza que el usuario puede seleccionar para golpear la bola, el número de puntos que tenemos precalculados es suficiente.

Como veremos más adelante, el proceso de golpeo de la bola tiene los siguientes pasos, desde el punto de vista del usuario:

- Seleccionar dirección de la bola
- Seleccionar fuerza con la que se golpea la bola
- Ver la trayectoria de bola y comprobar el resultado de los puntos anteriores

En el código implementado, para que el usuario no tenga que esperar (o tenga que esperar lo mínimo posible) por el cálculo de la trayectoria, esta se calcula en un *thread* independiente una vez que el usuario selecciona la dirección inicial de la bola. Es decir, es posible que calculemos

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

500 puntos de la trayectoria y que luego el usuario seleccione golpear la bola con poca fuerza y realmente se muestren sólo 50 de esos 500 puntos. Este es uno de los puntos que se pueden mejorar del juego y vuelvo a reiterar la propuesta de la introducción, le animo a ello.

Bien, vamos a ver el fragmento de código exacto que calcula la trayectoria, por supuesto, utilizando el código que ya hemos visto para detección de colisiones, cálculo de la nueva dirección de la bola después de una colisión...

El proceso es sencillo y se resume en los siguientes pasos:

- Se calcula el siguiente punto de la trayectoria, a partir de las variables calculadas anteriormente, tal y como se explicó en el apartado anterior
- Se comprueba si dicho punto sale fuera de los límites de la pantalla. Si es así, calculamos la nueva trayectoria, ya que la bola ha colisionado con un borde de la pantalla
- Comprobamos si la bola ha colisionado con algún otro *sprite*
- Si es un *sprite* correspondiente a una zona de agua, finaliza el cálculo de la trayectoria, indicando como punto final de la misma el punto inicial. Es decir, si la bola cae al agua, se vuelve a lanzar desde el punto inicial
- Si es un *sprite* correspondiente a un obstáculo, se recalcula la nueva trayectoria de la bola, tal y como hemos visto
- Si la colisión es contra el *sprite* correspondiente al hoyo, se finaliza el cálculo de la trayectoria y se indica como punto final el punto (1000,1000), de tal forma que si al dibujar la trayectoria encontramos que esta acaba en dicho punto, sabremos que se ha embocado.

Este proceso se repite hasta que se hayan calculado el total de puntos de la trayectoria o se finalice por los eventos antes descritos.

3.7 PANTALLA PRINCIPAL DEL JUEGO

A continuación vamos a ver cómo funciona la pantalla principal del juego. Como es de esperar, esta clase extiende la clase *Canvas* e implementa un hilo de ejecución (*thread*), de tal forma que sea capaz de ir dibujando las animaciones en la pantalla. Más adelante veremos esto en detalle. La clase principal del juego es la clase *PantallaGolf*.

Para comenzar vamos a ver el constructor de esta clase. En dicho constructor se inicializan algunas variables de la clase. Entre los procesos de inicialización, debemos destacar la creación del *sprite* para la bola y el *sprite* para el hoyo. También en el constructor se carga el campo de juego. Esto es así en este caso, porque estamos desarrollando una versión no completa del juego, de tal forma que sólo tendremos un campo de golf para jugar, cosa nada normal en lo que

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

sería un juego comercial. Para realizar un juego completo, tendríamos, seguramente, una clase con una serie de campos, y sabría qué campo proporcionar a la pantalla principal en cada momento para jugar. Esto es básicamente una versión del juego con varias pantallas. Una vez, animo al lector a hacerlo. Ya vimos en su momento cómo se crea el campo de minigolf.

Por último, dentro del constructor destacaremos la creación de una imagen del mismo tamaño que la pantalla, que se utilizará en aquellas situaciones en las que el fondo de la pantalla del juego no varía, de tal forma que en lugar de dibujar todos los elementos que componen la pantalla, tendremos una imagen con la pantalla completa y la dibujaremos con una única instrucción. Este hecho ocurre por ejemplo durante el proceso de selección de la fuerza del golpe. Durante este proceso se mostrará el campo de juego, y sobrepuesta sobre este una barra de de color que nos permite seleccionar la fuerza con la que se desea golpear la bola. La siguiente imagen ilustra este proceso y vemos que varía únicamente la barra roja y no el fondo de la pantalla.

3.7.1 CONTROL GENERAL DE LOS PROCESOS

La pantalla principal del juego deberá reaccionar de forma distinta ante distintas fases del juego. Así, en función del punto del juego en el que estemos, la pantalla principal deberá mostrar el círculo de selección de la trayectoria inicial de la bola, deberá mostrar el selector de fuerza (como se muestra en la imagen anterior) o se deberá dibujar el proceso de movimiento de la bola. Este hecho se repite para el control de las pulsaciones de las teclas por parte del usuario, según la fase del juego, la pulsación de una tecla tendrá consecuencias diferentes.

Para controlar en qué fase del juego nos encontramos, tenemos una variable global a toda la clase, denominada *tarea* y que indica en qué fase del juego nos encontramos. Los posibles

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

valores que puede tomar la variable *tarea* se definen en las siguientes constantes dentro de la clase:

- PINTAR_CAMPO – Determina que la operación a realizar es pintar el campo de juego. Esta tarea se realiza al comienzo del juego, para dibujar el campo completo de juego.
- PINTAR_CIRCULO – Indica que se está seleccionando la trayectoria inicial con la que comenzará a moverse la bola
- PINTAR_SELECTOR_FUERZA – Será la tarea de dibujar el selector de fuerza para que el usuario seleccione la fuerza con la que se golpeará la bola
- PINTAR_MOVIMIENTO_BOLA – Esta tarea, una vez calculada la trayectoria, muestra dicha trayectoria. Se pintarán tantos valores como permite la fuerza seleccionada para golpear la bola
- PINTAR_FINAL – Indica que hemos embocado la bola y que debemos mostrar la pantalla de felicitación. En nuestro caso se muestra el logo del juego

Teniendo en cuenta este método de control de las operaciones que se realizan en la pantalla principal, vamos a ver cómo funcionan los métodos de esta clase.

3.7.2 EL MÉTODO PAINT

Si el método *paint* es llamado durante la tarea de dibujado del campo, que será la primera operación que se realiza, lo que se hará es dibujar sobre la imagen de la que hemos hablado que se utilizará para almacenar el fondo todos los *sprites* que componen el campo, incluyendo el hoyo y la bola. Una vez que se ha dibujado el campo, se establece como tarea activa el dibujado del círculo de selección de la trayectoria inicial. En este momento ya tenemos almacenado el campo de juego dentro de la imagen antes mencionada.

Si la tarea activa es *PINTAR_CIRCULO*, se dibujará una línea desde el punto central de la bola, hasta el punto definido por el ángulo en cada momento. Este ángulo es modificado por el usuario a través de las pulsaciones de teclas, tal y como veremos más adelante. La siguiente imagen muestra el resultado del método *paint* durante esta parte del juego. La parte importante de esta pantalla es la que se ha remarcado con un círculo rojo.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Para pintar el selector de fuerza, se muestra un rectángulo relleno de color rojo en la pantalla que varía de longitud. La parte del rectángulo que se muestra de color indicará la fuerza con la que se golpeará la bola. La longitud de esta zona de color variará de forma automática, tal y como veremos más adelante cuando se estudie en detalle el método *run*. La siguiente imagen muestra el resultado del método *paint* durante el proceso de selección de fuerza.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Una vez seleccionada la fuerza para golpear la bola, se mostrará la trayectoria de la bola. Para realizar este proceso, el método *paint* simplemente muestra el campo del juego, a través de la imagen que hemos creado y configurado anteriormente y la posición de la bola en cada momento. Para hacer esto, se dibuja la imagen y se le ordena al *sprite* bola que se dibuje. El movimiento de la bola se hace en el método *run* tal y como veremos más adelante.

Por último, tenemos la pantalla final del juego, cuando se logra embocar la bola en el hoyo. En una versión comercial del juego, tendríamos una pantalla de puntación y el paso al siguiente nivel. En nuestro caso, lo que tendremos será la pantalla de presentación del juego. Con esto simplemente se pretende ilustrar el proceso de finalización del juego

En esta última pantalla será en la única en la que no se dibuja la posición de la bola dentro del campo. A continuación, se muestra el código del método *paint* de la clase *PantallGolf*.

```
public void paint (Graphics g){

    if (tarea == PantallaGolf.PINTAR_CAMPO){ gFondo.setColor(0,200,10); gFondo.fill-

        Rect(0,0,this.getWidth(),

                                this.getHeight());

        for (int i=0; i<campo.length; i++){ campo[i].paint(gFondo);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
}

hoyo.paint(gFondo);

tarea = PantallaGolf.PINTAR_CIRCULO;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        // Elemento del círculo a pintar = ángulo
        eltoCirculo=0; g.drawImage(fondo,0,0,
                Graphics.LEFT|Graphics.TOP);
    }
    if (tarea == PantallaGolf.PINTAR_CIRCULO){ g.drawImage(fondo, 0, 0,
                        Graphics.LEFT|Graphics.TOP); g.setColor(0,0,0);
        int[] c = pintarCirculo(); g.drawLine(bola.getX()+3, bola.getY()+3,
                bola.getX()+3+c[0], bola.getY()+3+c[1]);
    }
    if (tarea == PantallaGolf.PINTAR_SELECTOR_FUERZA){ g.drawImage(fondo, 0, 0,
                Graphics.LEFT | Graphics.TOP);
        g.setColor(0,0,0); g.drawRect(xFuerzaGolpe-1,yFuerzaGolpe-1,
                        maxFuerzaGolpe+1,21); g.setColor(200,0,0); g.fi-
        llRect(xFuerzaGolpe,yFuerzaGolpe,
                        fuerzaGolpe,20);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        }
        if (tarea == PantallaGolf.PINTAR_MOVIMIENTO_BOLA){ g.drawImage(fondo, 0,
            0,
                Graphics.LEFT|Graphics.TOP);
        }
        if (tarea == PantallaGolf.PINTAR_FINAL){ g.drawImage(imagenFinal,0,0,
                Graphics.LEFT|Graphics.TOP);
        }else{
            bola.paint(g);
        }
    }
```

El método *pintarCirculo* que se utiliza en el código anterior, tiene como función retornar el punto concreto del círculo que nos servirá para trazar la línea correspondiente. Este punto se toma de la constante presentada anteriormente en función del valor del campo *elementoCirculo*, que define el ángulo del círculo seleccionado en cada momento.

3.7.3 GESTIÓN DE LOS COMANDOS DEL USUARIO

Como ya hemos comentado anteriormente y hemos visto en el apartado anterior, el juego reaccionará de forma diferente según la fase en la que se encuentre. Es decir, en función de la tarea en la que se encuentre el juego, deberemos responder de forma diferentes a las pulsaciones de las teclas por parte del usuario.

Como ya sabemos, dentro de un *canvas* tenemos en el método *keyPressed* como punto principal para la gestión de las pulsaciones de teclas. Este método es invocado cuando el usuario pulsa una tecla, pero tenemos otro método para controlar cuando el usuario suelta la tecla que ha pulsado. Este método es el método *keyReleased*. En nuestro caso, tenemos la mayoría de la funcionalidad dentro del método *keyPressed*, pero utilizaremos también el método *keyReleased* para implementar una funcionalidad que nos permite controlar pulsaciones prolongadas por parte del usuario.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

En este punto ya conocemos cómo funciona el juego, por lo que nos será sencillo comprender que únicamente tenemos que controlar pulsaciones de tecla durante el proceso de selección de la trayectoria inicial de la bola y durante la selección de la fuerza de golpeo.

Para el primer caso, el proceso de selección de la trayectoria inicial de movimiento de la bola, el juego responderá modificando el ángulo seleccionado y por lo tanto se moverá la línea que va desde el punto central de la bola hasta el punto del círculo determinado por el ángulo. Para hacer esto, el sistema responde a la pulsación de las teclas del cursor (arriba, abajo, derecha e izquierda). El número de puntos que podemos seleccionar es 360 y por lo tanto el proceso, si sólo respondemos a las pulsaciones de tecla, será muy tedioso. Por ejemplo, para pasar del punto inicial, que es el ángulo 0 y la línea es completamente horizontal, al punto determinado por el ángulo 90, es decir, una línea completamente vertical, el usuario tendría que pulsar 90 veces la tecla correspondiente. Para agilizar este proceso, controlamos las pulsaciones prolongadas de teclas.

Cuando el usuario pulsa una tecla, el ángulo empieza a variar de forma automática (controlado por el método *run* de la clase) hasta que el usuario deja de pulsar la tecla. Esto nos permite implementar una interfaz de usuario mucho más eficiente. El resultado es que mientras el usuario tenga pulsada una tecla, la línea que define la trayectoria inicial de la bola se moverá en el sentido adecuado hasta que el usuario deje de pulsar dicha tecla.

Para finalizar el proceso de selección de la trayectoria inicial, el usuario debe pulsar la tecla de fuego (*fire*). Cuando ocurre esto, se comienza a calcular la trayectoria que describirá la bola, tal y como se ha explicado anteriormente y se pasa a la tarea de selección de la fuerza con la que se golpeará la bola.

Durante el proceso de selección de la fuerza de golpeó, lo único que se debe hacer es esperar a que el usuario pulse una tecla, lo que indicará que desea golpear la bola con la fuerza que se está mostrando en ese momento. Como respuesta a esta pulsación, lo que se hará es recuperar la trayectoria calculada y pasar a la tarea en la que se dibuja la trayectoria de la bola. Como se explicó en su momento, si el cálculo de la trayectoria no ha finalizado cuando esta es requerida (cuando el usuario pulse una tecla) el método mediante el que se solicita la trayectoria devolverá *null* y por lo tanto deberemos a que esta esté disponible para comenzar la tarea de dibujo de la trayectoria en la pantalla.

A continuación se muestra el código de este método, que implementa todo lo que acabamos de exponer

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
public void keyReleased(int tecla){

    // Paramos el movimiento de la línea de selcción avanceEltoCirculo=0;

}
public void keyPressed(int tecla){ tecla = this.getGameAction(te-

    cla); switch(tecla){

        case PantallaGolf.LEFT:
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
case PantallaGolf.UP:

    if (tarea == PantallaGolf.PINTAR_CIRCULO){ avanceEltoCirculo = 1;

    }

    break;

case PantallaGolf.RIGHT: case PantallaG-

olf.DOWN:

    if (tarea == PantallaGolf.PINTAR_CIRCULO){

        avanceEltoCirculo = -1;

    }

    break;

case PantallaGolf.FIRE:

    if (tarea == PantallaGolf.PINTAR_CIRCULO){

        // Transformo el ángulo del rango 0-360

        // al rango (-180)-180, que es con el que

        // se trabaja para calcular la trayectoria if (eltoCirculo >= 180 &&

                eltoCirculo <= 270){

            eltoCirculo = -(90 + (270 - eltoCirculo));

        } else if (eltoCirculo > 270){ eltoCirculo = -(360 - eltoCirculo);

        }

        // Comienzo a calcular la trayectoria calc.setDatosIniciales(bola.getX(),
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                        bola.getY(), -eltoCirculo);
                    calc.calcularTrayectoria();
                    tarea = PantallaGolf.PINTAR_SELECTOR_FUERZA;
                } else {
                    if (tarea ==
                            PantallaGolf.PINTAR_SELECTOR_FUERZA) {
                        // Mientras la trayectoria no esté lista
                        // esperamos
                        while ((trayectoria=calc.getTrayectoria())
                                == null){ try{
                            Thread.sleep(100);
                                }catch (InterruptedException iex){}
                        }
                        tarea=PantallaGolf.PINTAR_MOVIMIENTO_BOLA; indMovimiento=0;
                    }
                }
            }
        }
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3.8 EL MÉTODO RUN, EL CORAZON DEL JUEGO

Para finalizar, vamos a estudiar el método *run* de la clase *PantallaGolf*. Como ya hemos dicho, esta clase implementa la interfaz *Runnable*, de manera que el método *run*, como en cualquier-clase de este tipo, se ejecutará constantemente para realizar las operaciones adecuadas. En nuestro caso, en función de la fase del juego en la que nos encontremos, el método *run* deberá de hacer unas operaciones u otras.

La clase *PantallaGolf* es un *Canvas* y por lo tanto tiene la responsabilidad de dibujar en la pantalla del móvil lo que corresponda en cada momento. Es decir, lo que hará el método *run* responderá al siguiente esquema:

- Hacer los calculados adecuados y hacer las operaciones correspondientes
- Repintar la pantalla
- Esperar un determinado tiempo
- Volver al punto 1

Con la espera del punto 3, conseguimos asegurarnos de que se van a mostrar un número determinado de pantallas por segundo. Este factor se conoce como *fps (frames per second)*. Para conseguir que el número de *frames* por segundo sea constante, deberemos dormir siempre un tiempo, de tal forma que el tiempo total entre cada pantalla sea constante y por lo tanto el usuario tenga una sensación de dinamismo en el juego. Para conseguir que el tiempo entre diferentes pantallas sea constante, debemos calcular el tiempo que se tarda en hacer todas las operaciones y luego dormir el tiempo restante hasta conseguir que la suma del tiempo consumido, más el tiempo que duerme el proceso sea siempre el mismo.

A continuación vamos a ver qué debe hacer el método *run* en cada una de las fases del juego. Durante el proceso de selección de la trayectoria inicial de la bola, el método *run* deberá modificar el ángulo o lo que es lo mismo, modificar la línea en la dirección adecuada si alguna de las teclas de los cursores está pulsada, tal y como vimos en el punto anterior. El siguiente fragmento de código muestra este proceso.

```
if (tarea == PantallaGolf.PINTAR_CIRCULO){

    // avanceEltoCirculo será 1 o -1 eltoCirculo+=avanceElto-

    Circulo;

    if (eltoCirculo >= Constantes.circulo.length) eltoCirculo = 0;

    if (eltoCirculo < 0)
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
            eltoCirculo = Constantes.circulo.length - 1;

        }
```

El proceso que realiza el método *run* durante la selección de la fuerza de golpeo de la bola es muy sencillo. Simplemente modifica el área de la zona coloreada que se muestra en la pantalla para seleccionar la fuerza. El siguiente fragmento de código muestra dicho proceso.

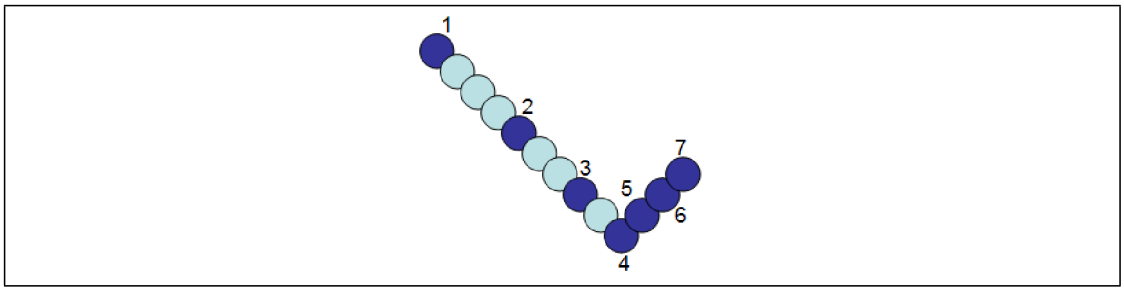
```
        if (tarea ==
```

```
                PantallaGolf.PINTAR_SELECTOR_FUERZA){

            fuerzaGolpe += incFuerzaGolpe;

            // Al llegar a los extremos, variamos la

            // dirección

            if (fuerzaGolpe == 100){ incFuerzaGolpe=-1;

            }

            if (fuerzaGolpe == 0){ incFuerzaGolpe=1;

            }

        }
```

Por último vamos a ver el código del método *run* que ejecuta para mostrar al usuario la trayectoria que describe la bola después de golpearse esta. Como sabemos, tenemos la trayectoria almacenada en una matriz bidimensional de números enteros. En el mundo real, al golpear la bola está empezará a moverse con una determinada fuerza que irá disminuyendo a medida que esta se mueve y va perdiendo la fuerza con la que se le impulsó. Para simular este hecho, nosotros

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

haremos lo siguiente. En función de la fuerza que tenga la bola en cada momento, se saltarán un determinado número de puntos de la trayectoria.



Por ejemplo, todos los puntos de la imagen anterior definen una trayectoria. A medida que la bola se mueve esta va perdiendo fuerza y por lo tanto cada vez se saltan menos puntos de la trayectoria. Siguiendo con el caso de la imagen anterior, se mostrarán únicamente por pantalla los puntos de la trayectoria de color azul oscuro. Es decir, se pinta el primer punto y se saltan tres puntos de la trayectoria. Se pinta el siguiente punto y luego se saltan dos puntos, ya que la bola va perdiendo fuerza. Este proceso se repite hasta que la bola pierde totalmente su fuerza o se llega al final de la trayectoria porque la bola ha caído al agua o porque ha embocado.

La máxima fuerza que se le puede asignar a la bola es un valor de 100. Cada vez que se muestra un punto, esta fuerza disminuye. Para calcular el número de puntos a saltar de la trayectoria y por lo tanto no hay que mostrar, se toma el número correspondiente a las decenas del valor que define la fuerza de movimiento de la bola en ese momento. Cuando el valor de dicha fuerza está por debajo 10, se avanzará siempre un punto sobre la trayectoria. Por

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

ejemplo, si el valor de la fuerza de la bola es 51, se saltarán 5 puntos de la trayectoria y se disminuirá el valor de la fuerza hasta 50. Para 50, se saltan otros 5 puntos y se disminuye la fuerza a 49, por lo que después de mostrar un punto, se saltaran otros 4.

De esta manera, determinamos el siguiente punto de la trayectoria a mostrar. Una vez que tenemos dicho punto, se mueve el *sprite* correspondiente a la bola hasta dicho punto y se actualiza la pantalla del terminal. Como se ha comentado, este proceso se repite hasta que la bola se queda sin fuerza o hasta que llegamos al final de la trayectoria. Cuando ocurre uno de estos dos hechos, se vuelve al principio del proceso y por lo tanto volvemos al proceso de selección de la trayectoria original de la bola, siempre que no se embocara la bola en el hoyo.

El siguiente fragmento de código se corresponde con el método *run* de la clase *PantallaGolf* e ilustra todo lo que acabamos de exponer.

```
public void run() {

    while (true) {

        long tiempoInicial = System.currentTimeMillis();

        if (tarea == PantallaGolf.PINTAR_CIRCULO) { eltoCirculo += avanceEltoCirculo;

            if (eltoCirculo >= Constantes.circulo.length) { eltoCirculo = 0;

            }

            if (eltoCirculo < 0) {

                eltoCirculo = Constantes.circulo.length - 1;

            }

        }

        if (tarea==PantallaGolf.PINTAR_SELECTOR_FUERZA){ fuerzaGolpe +=

            incFuerzaGolpe;

            if (fuerzaGolpe == 100) {

                incFuerzaGolpe = -1;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        }
        if (fuerzaGolpe == 0) { incFuerzaGolpe = 1;
        }
    }
    if (tarea==PantallaGolf.PINTAR_MOVIMIENTO_BOLA){ if (fuerzaGolpe > 0 &&
        indMovimiento < calc.getNumeroPuntos()) { int avance = (int) (fuerza-
        Golpe / 10); if (avance == 0) {
            avance++;
        }
        indMovimiento += avance; fuerzaGolpe--;
        if (indMovimiento < calc.getNumeroPuntos()) { bola.moveTo(trayectoria[indMovimiento][0],
                                trayectoria[indMovimiento][1]);
        }
        else {
            bola.moveTo( trayectoria[calc.getNumeroPuntos() - 1][0],
        trayectoria[calc.getNumeroPuntos() - 1][1]);
        }
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                if (trayectoria[indMovimiento][0] == 1000 &&

                        trayectoria[indMovimiento][1] == 1000) { tarea = Pantalla-

                    Golf.PINTAR_FINAL;

                }

            }

            else {

                tarea = this.PINTAR_CIRCULO; eltoCirculo = 0;

                fuerzaGolpe = 0;

                incFuerzaGolpe = 1;

            }

        }

        this.repaint(); this.serviceRepaints();

        long tiempoFinal = System.currentTimeMillis(); try {

            long r = tiempoFinal - tiempoFinal + retardo;

            if (r > 0) { thread.sleep(r);

            }

        }

        catch (InterruptedException ie) { ie.printStackTrace();}




    }

}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

4 LOS APIS ORIENTADOS A JUEGOS EN MIDP 2.0

4.1 INTRODUCCIÓN

Una de las mejoras más importantes que presenta MIDP 2.0 es su API destinado al desarrollo de aplicaciones de entretenimiento. A través de este API, podemos desarrollar interfaces de usuario más rápidas, con mejor usabilidad y con un uso de los recursos más eficiente. El API de juegos también ayuda a reducir el tamaño del fichero *jar* final que se ha de distribuir y descargar al terminal. Con MIDP 1.0, los desarrolladores debían proveer sus propias rutinas de gestión de gráficos para realizar los procesos y esto provoca el aumento del tamaño de los *jar*, al tener que añadir estas clases en la distribución.

La idea principal del API de juegos y que supone la base para todo el sistema de gráficos, es que la pantalla del juego se compone de capas. Las pantallas a mostrar se componen a través de capas. Todas estas capas pueden ser manejadas por separado y el API se encarga de dibujar las distintas capas de forma adecuada. El área de trabajo puede ser mayor que el tamaño de la pantalla del dispositivo y el movimiento o *scroll* de esta área puede resultar costoso. El API de juegos proporciona una vista de una parte del área de trabajo que compone una pantalla del juego (no una pantalla en el dispositivo).

4.2 PAQUETEJAVAX.MICROEDITION.LCDUI.GAME

El API de juegos se encuentra en el paquete *javax.microedition.lcdui.game*. A parte de este paquete, tendremos otras APIs dentro de la especificación que nos serán muy útiles en el desarrollo de juegos, como el denominado *Mobile Media API* que nos permitirá trabajar con sonidos.

Como vemos, el paquete *javax.microedition.lcdui.game* está dentro de la parte *lcdui* del API, lo que nos indica que las clases hacen referencia al interfaz de usuario, en concreto, a la gestión de los elementos que se muestran en la pantalla del dispositivo y los procesos que permiten componer dichas pantallas. Dentro del paquete *game* tenemos cinco clases:

- *GameCanvas*
- *Layer*
- *LayerManager*
- *Sprite*
- *TiledLayer*

4.2.1 La clase GameCanvas

La clase *GameCanvas* es abstracta y proporciona la funcionalidad básica para la realización de la interfaz de usuario. Esta clase presenta dos beneficios sobre la versión básica de *Canvas*, de la cual hereda. Por una parte tiene un *buffer off-screen* implementado y permite recoger el estado de las teclas del dispositivo, es decir, comprobar si las teclas están pulsadas o están liberadas.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Para cada instancia de *GameCanvas* tenemos un *buffer* dedicado, lo que nos permite tener una funcionalidad superior a la que proporciona *Canvas*, pero también debemos tener cuidado con este elemento, ya que el hecho de que cada instancia tenga un buffer dedicado, provoca que el uso masivo y sin consideraciones de objetos de este tipo, tenga un impacto importante sobre la memoria que consume nuestra aplicación. Es muy recomendable reutilizar los objetos de tipo *GameCanvas* para evitar que los requerimientos de memoria se disparen.

En un primer momento, al crear un objeto de tipo *GameCanvas*, todo el *buffer* sobre el que trabaja se llenará con *pixels* de color blanco. La clase *GameCanvas* provee una serie de constantes para averiguar qué tecla es pulsada por el usuario. El estado de las teclas, se puede recuperar a través del método *getKeyStates*, que retorna un entero en el que cada *bit* indica el estado de una de las teclas del terminal. Una ejemplo del uso de este método es el siguiente fragmento de código:

```
// Get the key state and store it

int keyState = getKeyStates();

if ((keyState & LEFT_PRESSED) != 0) { positionX--;

} else if ((keyState & RIGHT_PRESSED) != 0) {

      positionX++;

}
```

Otro método a destacar dentro de la clase *GameCanvas*, es el método *flushGraphics*. Este método, tiene dos versiones, una sin parámetros y otra con parámetros. Este método muestra el contenido del *buffer off-screen* en la pantalla del terminal. En la versión con parámetros, se especifica una parte del *buffer*, que será aquella zona del mismo que se mostrará por pantalla.

4.2.2 La clase Layer

La clase *Layer* es una clase abstracta que representa un elemento visual dentro del juego, es decir, en un juego típico de plataformas, cada uno de los dibujos que componen de las plataformas, los diferentes personajes del juego e incluso la imagen del fondo del mismo, serán objetos de tipo *Layer*. Las clases *Sprite* y *TiledLayer*, que veremos a continuación, heredan de la clase *Layer*.

Cada objeto de tipo *Layer* tiene una posición (determinada por unas coordenadas con respecto a la esquina superior-izquierda del objeto *Graphics* que recibe el método *paint*), un ancho y un alto y

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

puede ser establecida como visible o como invisible. Las subclases de *Layer* implementan el método *paint(Graphics)*, de tal forma que cualquiera de estos elementos puede ser mostrado por pantalla. Este método está definido como abstracto en la clase *Layer*.

Los métodos que contiene esta clase son básicamente los métodos de gestión de posición, las dimensiones y la visibilidad del objeto *Layer*.

4.2.3 La clase LayerManager

Los objetos de tipo *Layer* que tenemos en nuestro código son manejados por un gestor de capas, es decir, un objeto de tipo *LayerManager*. Esta clase se encarga de gestionar todo el proceso de pintado de las distintas capas (*Layers*) que componen cada pantalla del juego, dibujando en el orden adecuado los diferentes elementos. El gestor de capas conoce en todo momento las capas de las que debe encargarse. El índice de la capa dentro del gestor, indica su profundidad o posición en el eje z. Si el valor de este índice es 0, la capa será la más cercana al usuario y cuanto mayor sea el índice, mayor será la profundidad de la capa.

La clase *LayerManager* proporciona algunos métodos para el control de la visualización de la capas (*Layers*), es decir, para controlar qué es exactamente lo que se genera en la pantalla. Se puede controlar la región visible de la pantalla, lo que nos permite hacer movimientos laterales de la pantalla (*scroll*) de forma sencilla. Este efecto se denomina *ventana de visualización*, ya que la idea es exactamente esa, tenemos una ventana visible dentro de todo el gráfico que componen los diferentes objetos de tipo *Layer*.

El método *paint* de la clase *LayerManager* tiene dos parámetros de tipo entero, que indican la posición de la ventana de visualización de la que hablábamos en el párrafo anterior dentro de la pantalla del terminal. Las coordenadas de esta posición son relativas al objeto *Graphics* que recibe también como parámetro. Es decir, partiendo de una imagen compuesta de capas, de un tamaño superior al de la pantalla del terminal, definimos una región dentro de esta imagen que será la región que vamos a visualizar y además, indicamos también el punto dentro de la pantalla del terminal en la que se mostrará dicha región.

La clase *LayerManager* presenta una serie de métodos que nos permiten gestionar las diferentes capas que componen la escena, pudiendo añadir, eliminar e insertar en una determinada posición, las capas. A parte de estos métodos, cabe destacar el método *setViewWindow* que nos permitirá definir la ventana de visualización dentro la escena.

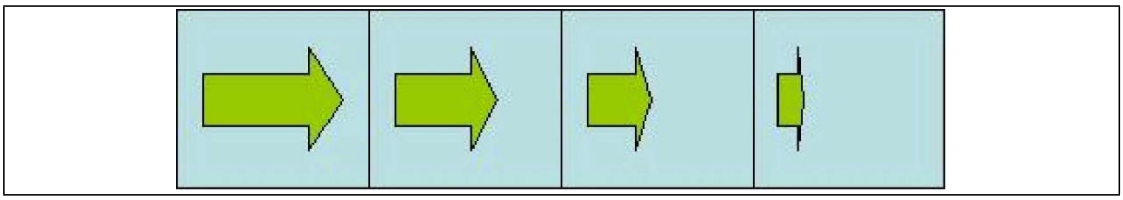
4.2.4 La clase Sprite

Los *sprites* se pueden definir como cada uno de los fotogramas de una animación para un personaje o elemento dentro de un juego. En MIDP 2.0, a partir de una única imagen se obtienen todos los *sprites* o fotogramas que componen un objeto animado. Esta imagen puede contener uno o más de estos fotogramas. En caso de que la imagen contenga más de un fotograma correspondiente a la animación de un objeto o elemento, estos fotogramas serán

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

todos iguales en cuanto al tamaño y por lo tanto se obtendrán dividiendo la imagen original en pedazos. Es decir, la imagen que contiene los fotogramas de puede concebir como la unión de dichos fotogramas.



En la imagen anterior, tenemos cuatro fotogramas de una animación en la que una flecha se encoge horizontalmente. Para extraer los fotogramas de esta imagen, simplemente dividimos la imagen en los cuadrados y todos deben tener las mismas dimensiones. El formato de la imagen puede variar y en lugar de tener la disposición de los fotogramas que tenemos en el ejemplo anterior, podemos agrupar las imágenes de forma vertical o en forma de tabla.

Los diferentes fotogramas son indexados a la hora de dividir la imagen. El fotograma más arriba y más a la izquierda será el correspondiente al índice 0 y el resto se irán numerando consecutivamente de izquierda a derecha y de arriba abajo. El orden de esta secuencia de fotogramas es el mismo orden en el que serán mostrados a la hora de realizar la animación. El programador puede modificar esta secuencia con métodos de la clase *Sprite*, de tal forma que usando los índices de los fotogramas, construya una secuencia propia de visualización. Si retomamos el ejemplo de la flecha que hemos anteriormente, tendríamos por defecto la secuencia 0,1,2,3. Esta secuencia de visualización de los fotogramas nos daría como resultado una animación en la que flecha encoge y desde el tamaño menor vuelve a pasar al más grande. Si queremos un resultado en el que la flecha crezca y encoja de forma lógica, deberíamos definir la secuencia 0,1,2,3,2,1,0.

La clase *Sprite* incluye lo que se denomina el *pixel* de referencia. El *pixel* de referencia es un punto del fotograma definido a través del método *defineReferencePixel*. Por defecto, este punto corresponde a las coordenadas (0,0), que será la esquina superior izquierda del fotograma. Hay que recordar que este punto es un punto del fotograma y no del sistema de coordenada global que define la pantalla del terminal. Este *pixel* se puede definir incluso fuera de los límites del fotograma. Con el método *setRefPixelPosition* podemos situar el *sprite* de tal forma que el *pixel* de referencia se sitúe en el punto indicado.

Otra de las funcionalidades útiles de la clase *Sprite* para manejar los fotogramas, es aquella que nos permite realizar transformaciones sobre el *sprite* original. El método para realizar estas transformaciones es el método *setTransform*, y las posibles transformaciones son:

- TRANS_NONE – Deja el *sprite* tal y como estaba originalmente

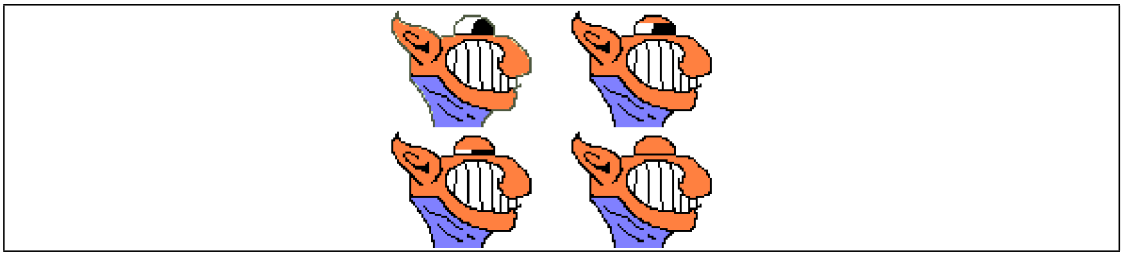
ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- TRANS_ROT180 – Gira la imagen 180 grados
- TRANS_MIRROR – Muestra el *sprite* como si se reflejara en un espejo
- TRANS_MIRROR_ROT180 – Refleja el *sprite* y lo gira 180 grados
- TRANS_ROT90 – Rota la imagen 90 grados
- TRANS_MIRROR_ROT90 – Refleja el *sprite* y lo gira 90 grados
- TRANS_ROT270 – Rota la imagen 270 grados
- TRANS_MIRROR_ROT270 – Refleja el *sprite* y lo gira 270 grados

Cuando aplicamos una transformación de estas a un *Sprite*, dicha transformación es aplicada sobre el fotograma original y siempre es aplicada con respecto al *pixel* de referencia, por lo tanto, el fotograma reposicionado, de tal forma que el *pixel* de referencia permanece inalterado con respecto al sistema de coordenadas global. Podemos considerar al *pixel* de referencia como el punto central de la transformación o el eje de giro de la transformación.
Para dibujar los *Sprites*, son los propios *Sprites* los que mantienen toda la información referente a la posición, el fotograma a visualizar... y simplemente tendremos que llamar al método *paint* de la clase *Sprite* para que el fotograma correspondiente del *sprite* sea dibujado.

4.2.5 EJEMPLO DEL USO DE SPRITES

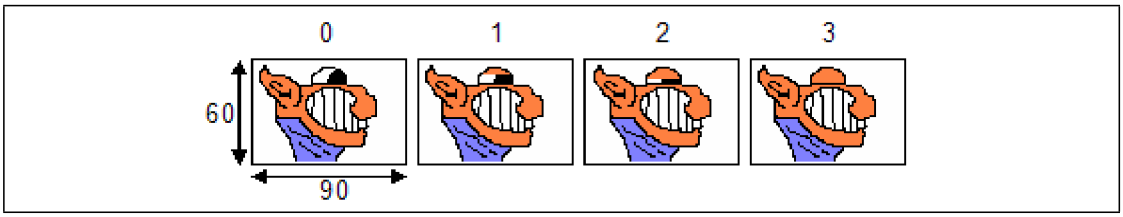
A continuación vamos a ver un pequeño ejemplo de cómo funciona la gestión de *sprites* en MIDP 2.0 y cómo se utilizan algunas de las funcionalidades que hemos visto hasta ahora. En primer lugar, partimos de una imagen *png* que contiene todos los fotogramas necesarios para implementar la animación de nuestro personaje. En nuestro caso, los fotogramas tienen unas dimensiones de 90x60 *pixels* y son cuatro fotogramas colocados en forma de tabla. El tamaño de la imagen contenedora será por tanto 180x120.



Como vemos, tenemos un personaje cuya animación en este caso consiste en el parpadeo de su único ojo. Al dividir esta imagen en fotogramas de 90x60, tendremos los siguientes fotogramas:

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



Al dividir la imagen, como ya hemos indicado anteriormente, se asignará un número de secuencia para configurar la imagen, que irán del 0 al 3 en nuestro caso. Bien, si mantenemos esta secuencia inalterada tendremos que un efecto de parpadeo un poco raro, ya que pasamos del ojo totalmente cerrado al ojo totalmente abierto. Para solucionar esta situación, definiremos una nueva secuencia para los fotogramas, en la que el orden de visualización de los mismos sea 0,1,2,3,2,1,0.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El siguiente código muestra estos detalles y algunos más que comentaremos a continuación dentro del código:

```
import javax.microedition.lcdui.*;

import javax.microedition.lcdui.game.*;

public class AnimaCiclope extends GameCanvas implements Runnable {

    // Imagen que contiene todos los sprites Image ciclopeimg = null;

    // Sprites que componen la animación Sprite sprt = null;

    // Gestor de capas

    LayerManager mng = new LayerManager(); private Thread

    thread;


    public AnimaCiclope() { super(false);

      // Cargamos la imagen contenedora try{

                    ciclopeimg = Image.createImage("/ciclope.png");

            }catch (Exception ex){ ex.printStackTrace();

            }

            // Extraemos los fotogramas de la imagen sprt = new Sprite(ciclo-

            peimg,90,60);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
            // El punto de referencia es el centro del fotograma
      sprt.defineReferencePixel(45, 30);
      // Definimos al secuencia de animación int seq[] = new int[]
      {0,1,2,3,2,1,0}; sprt.setFrameSequence(seq);
      // Insertamos el sprite en gestor de capas mng.insert(sprt, 0);

    }
    protected void showNotify() { thread = new Thread(this);
      thread.start();
    }
    public void run() {
      // Recuperamos el buffer de la pantalla Graphics g = getGraphics();
Thread mythread = Thread.currentThread();
// Ponemos el pixel de referencia del sprite en el punto
// central de la pantalla. Así tenemos la animación en
// el centro de la patanlla del terminal sprt.setRefPixelPosition((int)(this.getWidth()/2),
(int)(this.getHeight()/2));
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
    while (mythread == thread) {
            // Borramos la pantalla del terminal g.setColor(0xFFFFFF); g.fillRect(0,0,this.getWidth(),this.get-
            Height());
            // Nos movemos al siguiente fotograma de la animación sprt.nextFrame();


            // Dibujamos el sprite sprt.paint(g);
            // Mostramos el buffer en la pantalla flushGraphics();
            try {
                mythread.sleep(100);
            } catch (java.lang.InterruptedException e) {
            }
    }
}
protected void keyPressed(int keyCode) {

    // Recuperamos la tecla que se ha pulsado
    // Modificamos la posición del sprite en función de la
    // tecla que se pulse
    int accion = getGameAction(keyCode);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
    switch (accion) {
            case Canvas.LEFT: sprt.setRefPixelPosition(
                sprt.getRefPixelX()-5,sprt.getRefPixelY()); break;
            case Canvas.RIGHT: sprt.setRefPixelPosition(
                sprt.getRefPixelX()+5,sprt.getRefPixelY());
                break;
            case Canvas.DOWN: sprt.setRefPixelPosition(
                sprt.getRefPixelX(),sprt.getRefPixelY()+5); break;
            case Canvas.UP: sprt.setRefPixelPosition(
                sprt.getRefPixelX(),sprt.getRefPixelY()-5); break;
            case Canvas.FIRE:
                // Cuando se pulsa la tecla de "fuego", rotamos
                // 90 grados la imagen sprt.setTransform(Sprite.TRANS_ROT90); break;
            default:
                return;
      }
      // Mostramos la nueva posición del pixel de referencia
      // dentro del sistema de coordenadas general System.out.println("("+sprt.getRefPixelX()+
                ","+sprt.getRefPixelY());
    }
}
```

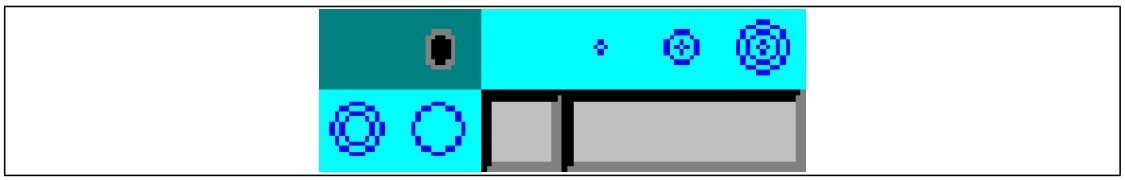

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El código anterior nos permite comprobar el funcionamiento de algunos de los puntos de los que hemos hablado a lo largo de este capítulo. El anterior ejemplo muestra como extraer los fotogramas de la imagen y muestra como gestionar los diferentes elementos a través de un objeto *LayerManager*.

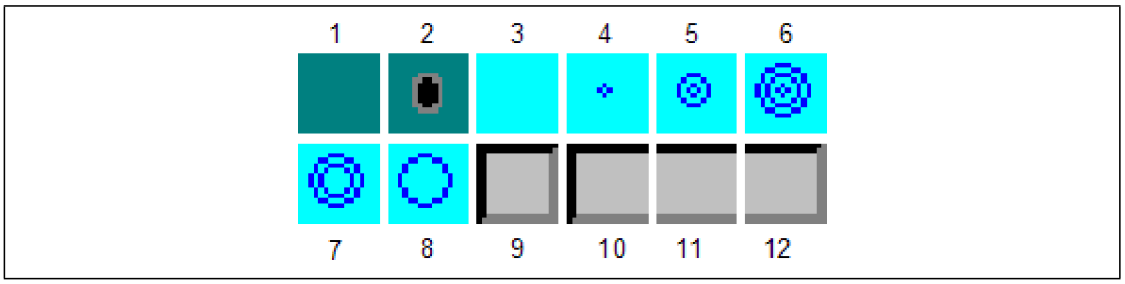
4.2.6 LA CLASE TILEDLAYER

Los objetos de tipo *TiledLayer*, representan un elemento visual compuesto por un grupo de celdas que se corresponden con una serie de imágenes. Esta clase nos permite crear dibujos a partir de pequeñas celdas, de tal forma que compondremos una imagen grande a partir de una o varias imágenes pequeñas. Se puede decir que el objetivo es crear un mosaico, es decir, a partir de pequeños elementos, repitiendo estos y colocándolos en las posiciones adecuadas conseguiremos una imagen mucho más compleja. Esta técnica es muy habitual dentro de los juegos de plataformas para conseguir escenarios complejos.

Los distintos elementos que componen el mosaico están contenidos en una única imagen, de forma similar a lo visto en el caso de los *Sprites*. Esta imagen es divida en pequeñas partes, de las mismas dimensiones cada una de las partes.



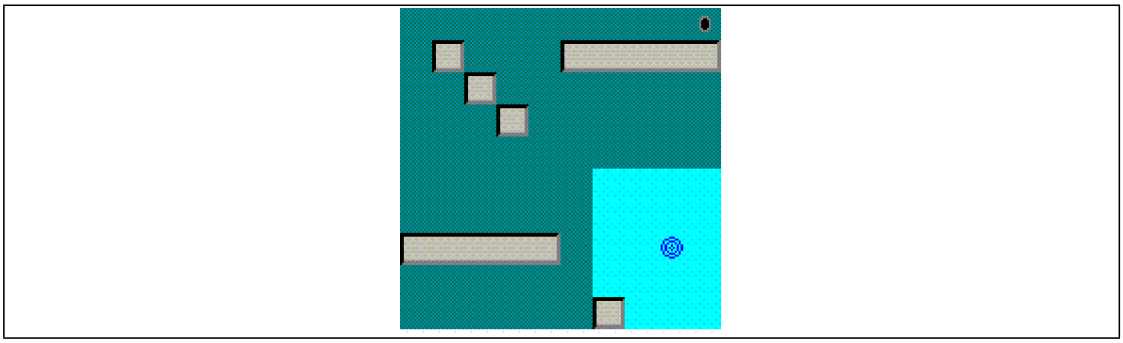
La imagen anterior contiene una serie de pequeñas imágenes que nos servirán para componer las pantallas de nuestro juego. En concreto, tiene 12 imágenes de 16 *pixels* de ancho por 16 *pixels* de alto. La siguiente figura muestra el resultado del proceso de separación de la imagen.



ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Al extraer los diferentes elementos de la imagen, se les asigna un índice automáticamente. Este índice comienza en 1. Con estas imágenes podemos componer una imagen como la siguiente, que podría corresponder a un juego de minigolf:



A la hora de componer la imagen final, también podemos usar lo que denominaremos *tile* (siguiendo con la metáfora del mosaico) o azulejo animado. Estos elementos no están presentes en la imagen original, pero se utilizan también para crear la pantalla. Los azulejos animados tendrán un determinado índice (siempre menor que 0) y son creados a través del método *createAnimatedTile*. Los azulejos virtuales son asociados dinámicamente en tiempo de ejecución con algún azulejo de los que extraídos de la imagen, y cambiando esta asociación tendremos una animación. La asociación entre el azulejo animado virtual y el azulejo estático se hace con el método *setAnimatedTile*. En la figura anterior, tenemos unos círculos dentro del lago. Esto es un elemento animado, que a través de las imágenes 4,5,6,7 y 8 extraídas de la imagen original, crea un efecto de ondas en el agua. En el ejemplo siguiente vemos el proceso de construcción de un escenario a partir de las imágenes contenidas en un fichero. En concreto, vamos a construir el escenario del minigolf que tenemos en la última figura.

```
import javax.microedition.lcdui.*;

import javax.microedition.lcdui.game.*;

public class Campo extends GameCanvas implements Runnable {

    // Imagen que contiene todos los azulejos o tiles Image campoimg = null;

    // Tiles o azulejos con los que vamos a trabajar TiledLayer tiles = null;

    // Gestor de capas

    LayerManager mng = new LayerManager(); Thread thread;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
// Variable usada para realizar la animación de la onda en

// el agua. Indica el tile a dibujar en cada momento int gota;



public Campo() { super(false);

  // Cargamos la imagen contenedora try{

                        campoimg = Image.createImage("/campo.png");

            }catch (Exception ex){ ex.printStackTrace();

            }

            // Extraemos los tiles de la imagen y creamos una rejilla

            // de 10x10 que será sobre la que compongamos la imagen tiles = new TiledLayer(10,10,cam-

  poimg,16,16);

            // Creamos el tile animado para ponerlo sobre el lago

            // y simular el efecto de onda sobre el agua gota = tiles.createAnimatedTile(3);



            // Rellenamos todo el escenario con el azulejo verde for (int i=0;i<10;i++){

            for (int j=0;j<10;j++){ tiles.setCell(i,j,1);

            }
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
}

        // Situamos algunos azulejos sobre la rejilla para componer

        // el escenario ti-

les.setCell(9,0,2); tiles.setCell(9,1,12);

tiles.setCell(8,1,11); ti-

les.setCell(7,1,11); tiles.setCell(6,1,11);

tiles.setCell(5,1,10); tiles.setCell(1,1,9);

tiles.setCell(2,2,9); tiles.setCell(3,3,9);

tiles.setCell(0,7,10); ti-

les.setCell(1,7,11); tiles.setCell(2,7,11);

tiles.setCell(3,7,11); ti-

les.setCell(4,7,12);

        // Creamos el lago

for (int i=6;i<10;i++){

        for (int j=5;j<10;j++){ tiles.setCell(i,j,3);

        }

}

tiles.setCell(6,9,9);
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                // Añadimos el tile animado en la mitad del lago
    tiles.setCell(8,7,gota);

    // Insertamos la rejilla dentro del gestor de capas mng.insert(tiles, 0);

  }

  protected void showNotify() { thread = new Thread(this);
    thread.start();
  }

  public void run() {
    // Recuperamos el buffer de la pantalla Graphics g = getGraphics();
Thread mythread = Thread.currentThread(); int indiceGota = 3;

    while (mythread == thread) {
                // Borramos la pantalla del terminal g.setColor(0xFFFFFF); g.fill-
                Rect(0,0,this.getWidth(),this.getHeight());
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                // Calculamos el tile estático a mostrar

                // dentro del tile animado if (indiceGota == 3){

                        indiceGota=4;

                }else if (indiceGota == 4){ indiceGota=5;

                }else if (indiceGota == 5){ indiceGota=6;

                }else if (indiceGota == 6){ indiceGota=7;

                }else if (indiceGota == 7){ indiceGota=8;

                }else if (indiceGota == 8){

                        indiceGota=3;

                }

                // Dibujamos el tile animado, con el elemento adecuado tiles.setAnimatedTile(gota, indiceGota);



                tile.paint(g);

                // Mostramos el buffer en la pantalla flushGraphics();

                try {

                                mythread.sleep(500);
                  } catch (java.lang.InterruptedException e) {

                  }

        }

      }

}
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

5 APÉNDICE A – COMPILACIÓN CON ANT Y ANTENNA

Como sabemos, tenemos a nuestra disposición ciertas herramientas más o menos avanzadas que nos permiten realizar el proceso de compilación. Cuando hablamos de J2ME y de *midlets*, el proceso de compilación cubre más etapas que la propia compilación, ya que primero debemos realizar la compilación propiamente dicha, luego preverificar las clases y por último crear los ficheros *jar* y *jad*, para distribuir la aplicación. Una vez hecho todo este proceso manualmente para comprender los diferentes pasos y tener claro todo el proceso, seremos capaces de comprender mejor el funcionamiento de estas herramientas y solucionar posibles problemas que se nos presenten.

Ant es una herramienta de compilación avanzada, realizada en java, que permite automatizar los procesos de compilación, empaquetado y preparación para la distribución de las aplicaciones. *Ant* se basa en ficheros XML de configuración que permiten definir todas las tareas a realizar, así como las dependencias entre las distintas tareas. El fichero de configuración de *Ant*, tiene el nombre *build.xml*. Como todos los ficheros XML, este fichero contiene una arquitectura de etiquetas, que definen los valores de los distintos elementos del fichero.

El elemento *project* del fichero XML debe estar presente en el fichero al menos una vez y es el elemento más general para un determinado proyecto, que se compondrá de una serie de tareas. El elemento *project* tiene un atributo denominado *default* que indica la tarea (*target*) a realizar cuando no se especifica ninguna otra tarea en concreto. El atributo *basedir* del elemento *project* define el camino a partir del cual se deben tomar el resto de caminos (*path*) definidos en el fichero.

```
<project name="midlet1" default="run" basedir=".">
    <target name="compilar">
        <javac scrdir="${src}" destdir="${classes}"
            bootclasspath="${classpath}">
        </javac>
    </target>
</project>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En el ejemplo anterior tenemos un proyecto con una única tarea, la tarea “compilar”. Esta situación no es común dentro de los ficheros *build.xml* de *Ant*, ya que lo habitual es tener una serie de tareas. En el caso de J2ME, esta situación es aún más anómala, ya que es inútil, porque como hemos dicho anteriormente, en J2ME necesitamos realizar varias tareas antes de tener la aplicación lista para su distribución. El ejemplo anterior, por lo tanto, se puede extender de la siguiente forma para que tenga cierta utilidad:

```
<project basedir="." default="empaquetar" name="MIDlet1">
    <property name="wtk_dir" value="c:/wtk104"/>
    <property name="clases" value="classes"/>
    <property name="fuentes" value="src"/>
    <property name="recursos" value="res"/>
    <property name="distribucion" value="dist"/>
    <property name="classpath" value="${wtk_dir}/lib/midpapi.zip"/>
    <property name="nombreDist" value="Midlet1"/>
    <target depends="init" name="compilar">
        <javac srcdir="${fuentes}" destdir="${clases}" bootclasspath="${classpath}" list-
                files="true">
        </javac>
    </target>
    <target name="gestion_recursos">
        <echo message="Gestión de recursos" />
        <copy todir="${clases}">
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
            <fileset dir="${recursos}">
                <include name="**/*.png"/>
            </fileset>
        </copy>
    </target>

    <target name="preverificar" depends="compilar">
        <exec executable="${wtk_dir}/bin/preverify.exe">
            <arg line="-classpath ${classpath}"/>
            <arg line="-d ${clases}"/>
        </exec>
    </target>


    <target name="empaquetar" depends="preverificar,gestion_recursos">
        <jar destfile="${distribucion}/${nombreDist}.jar" manifest="MANIFEST.MF">
                <fileset dir="${clases}"/>
            </jar>
          <copy file="${nombreDist}.jad" tofile="${distribucion}/${nombreDist}.jad" />
    </target>

    <target name="ejecutar" depends="empaquetar">
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        <exec dir="${distribucion}"
                                executable="${wtk_dir}/bin/emulator">
            <arg line="-classpath ${nombreDist}.jar"/>
            <arg line="-Xdescriptor:${nombreDist}.jad"/>
        </exec>
    </target>

    <target name="init">
        <echo message="Limpiando...." />
        <delete dir="${clases}" />
        <delete dir="${distribucion}" />

        <mkdir dir="${clases}" />
        <mkdir dir="${distribucion}" />
    </target>

</project>
```

Con un fichero como este colocado en el directorio raiz del proyecto e invocando el comando *ant*, tendremos automatizado todo el proceso de compilación, empaquetamiento... del proyecto. Dentro del ejemplo podemos diferenciar tareas como compilar, en el que usamos ciertos elementos estándar de la herramienta (*javac*) de otras tareas como preverificar, que son propias de J2ME y deben ser indicadas más rudimentariamente, como tareas con ejecución de comandos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Si bien el uso de *Ant* supone una gran ayuda durante el proceso de desarrollo, la versión general que acabamos de ver, puede ser ampliada con nuevas tareas especificas para J2ME y que nos facilitará aún más la creación de *midlets*. El proyecto *Antenna* es desarrollado para extender *Ant* y solucionar este problema.

El fichero *build.xml* anterior, modificado para el uso de las tareas proporcionadas por el proyecto *Antenna*, será el siguiente:

```
<project basedir="." default="empaquetar" name="MIDlet1">
<!-- Información del midlet -->
  <property name="midlet" value="midlet1"/>
  <property name="nombreMidlet" value="Midlet1"/>
  <property name="claseMidlet" value="com.vitike.libroj2me.midlet1.MIDlet1"/>

  <property name="iconoMidlet" value=""/>
  <property name="perfil" value="1.0"/>
  <property name="configuracion" value="1.0"/>
  <property name="wtk_dir" value="c:/wtk104"/>
  <property name="clases" value="classes"/>
  <property name="fuentes" value="src"/>
  <property name="recursos" value="res"/>
  <property name="distribucion" value="dist"/>
  <property name="classpath" value="${wtk_dir}/lib/midpapi.zip"/>
  <!-- Configuración de las tareas de Antenna -->
  <property name="wtk.home" value="${wtk_dir}"/>
  <taskdef name="wtkjad" classname="de.pleumann.antenna.WtkJad"/>
  <taskdef name="wtkbuild"
                                    classname="de.pleumann.antenna.WtkBuild"/>
  <taskdef name="wtkpackage"
                                    classname="de.pleumann.antenna.WtkPackage"/>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
<taskdef name="wtkmakeprc"
                    classname="de.pleumann.antenna.WtkMakePrc"/>
<taskdef name="wtkrun" classname="de.pleumann.antenna.WtkRun"/>
<taskdef name="wtkpreverify"
                    classname="de.pleumann.antenna.WtkPreverify"/>
<taskdef name="wtkobfuscate"
                    classname="de.pleumann.antenna.WtkObfuscate"/>


<target depends="init" name="compilar">
    <wtkbuild srcdir="${fuentes}" destdir="${clases}" preverify="true">
    </wtkbuild>
</target>
<target name="empaquetar" depends="compilar">
    <wtkjad jadfile="${distribucion}/${midlet}.jad" jarfile="${distribucion}/${midlet}.jar"
                    name="Ejemplos del libro de j2me" vendor="Manuel J. Prieto">

        <!-- Definición de atributos propios para el jad-->
        <attribute name="Propiedad" value="Valor de propiedad"/>

        <midlet name="${nombreMidlet}"
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                    icon="${iconoMidlet}"
                    class="${claseMidlet}">
        </midlet>
      </wtkjad>
      <wtkpackage basedir="${clases}"
                    jarfile="${distribucion}/${midlet}.jar" jadfile="${distribucion}/${mid-
                    let}.jad" config="${configuracion}" profile="${perfil}">
          <fileset dir="${recursos}"/>
        </wtkpackage>

      <copy file="${midlet}.jad" tofile="${distribucion}/${midlet}.jad"/>
  </target>

  <target name="ejecutar" depends="empaquetar">
    <wtkrun jadfile="${distribucion}/${midlet}.jad" />
  </target>

  <target name="init">
    <echo message="Limpiando...." />
    <delete dir="${clases}" />
      <delete dir="${distribucion}" />
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        <mkdir dir="${clases}" />

        <mkdir dir="${distribucion}" />

    </target>

</project>
```

Se puede ver que las tareas nuevas, facilitan el proceso de construcción de los ficheros *jad* y *jar*. Un ejemplo de esta nueva situación, es el hecho de que el fichero *jad* es actualizado automáticamente con el tamaño del fichero *jar*, por lo que no debemos preocuparnos más por este valor. En el ejemplo anterior, vemos cómo añadir atributos propios al fichero *jad* (inserción del campo *propiedad* en el ejemplo anterior), que nos serán muy útiles como valores de configuración de los *midlets*.

6 APÉNDICE B – LUGARES EN INTERNET DE REFERENCIA

- wireless.java.sun.com – Página oficial de la compañía Sun Microsystems dedicada a J2ME y todo lo relacionado con dicha tecnología. Este es el lugar esencial para mantenerse informado sobre las nuevas especificaciones dentro de J2ME. También son de gran utilidad lo artículos que se publican en dicha página y los denominadas *Tech Tip*. Algunos de los artículos publicados en esta página son la base de conocimiento de algunos fragmentos de este documento. En este web también se pueden descargar las herramientas necesarias para el desarrollo de aplicaciones J2ME.
- www.nokia.com – La parte de la web oficial de la compañía Nokia dedicada a los creadores de aplicaciones, contiene una gran cantidad de información de una altísima calidad sobre la mayoría de las tecnologías móviles en el mercado. Por supuesto, hay una zona dedicada a J2ME. Destacable dentro de esta web son los foros de usuarios y los artículos y guías desarrolladas por Nokia y publicadas en dicha web. Algunos de estos artículos, como se ha mencionado anteriormente, son punto de referencia de este libro.
- www.j2me.org – Esta dirección contiene una serie de foros dedicados a tratar temas relacionados con J2ME. Uno de estos foros será especialmente interesante en nuestro caso, es el caso del foro dedicado al desarrollo de juegos con J2ME.
- www.microjava.com – Página web con un gran número de secciones dedicadas en exclusiva a J2ME. Cabe destacar la sección dedicada a los desarrolladores y la sección dedicada a las descargas, donde podemos encontrar desde aplicaciones completas hasta APIs concretas que usar en nuestros desarrollos.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Adriana Fernández-Fernández, Cristina Cervelló-Pastor, Leonardo Ochoa-Aday (2016). Energy-Aware Routing in Multiple Domains Software-Defined Networks. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 3
2. Alberto Fernández-Isabel, Rubén Fuentes-Fernández (2015). Simulation of Road Traffic Applying Model-Driven Engineering. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 2
3. Amir Hosein Keyhanipour, Behzad Moshiri (2013). Designing a Web Spam Classifier Based on Feature Fusion in the Layered Multi-Population Genetic Programming Framework. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
4. Ana Oliveira Alves, Bernardete Ribeiro (2015). Consensus-based Approach for Keyword Extraction from Urban Events Collections. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 2
5. Ángel Martín del Rey, F. K. Batista, A. Queiruga Dios (2017). Malware propagation in Wireless Sensor Networks: global models vs Individual-based models. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
6. Anna Závodská, Veronika Šramová, Anne-Maria AHO (2012). Knowledge in Value Creation Process for Increasing Competitive Advantage. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
7. Antonio Pinto, Ricardo Costa (2016). Hash-chain-based authentication for IoT. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 4
8. Casado-Vara, R., & Corchado, J. (2019). Distributed e-health wide-world accounting ledger via blockchain. Journal of Intelligent & Fuzzy Systems, 36(3), 2381-2386.
9. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
10. Casado-Vara, R., de la Prieta, F., Prieto, J., & Corchado, J. M. (2018, November). Blockchain framework for IoT data quality via edge computing. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (pp. 19-24). ACM.
11. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
12. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
13. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
14. Céline Ehrwein Nihan (2013). Healthier? More Efficient? Fairer? An Overview of the Main Ethical Issues Raised by the Use of Ubicomp in the Workplace. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 1
15. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
16. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
17. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
18. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
19. Corchado, J. A., Aiken, J., Corchado, E. S., Lefevre, N., & Smyth, T. (2004). Quantifying the Ocean's CO2 budget with a CoHeL-IBR system. In Advances in Case-Based Reasoning, Proceedings (Vol. 3155, pp. 533–546).
20. Corchado, J. M., & Fyfe, C. (1999). Unsupervised neural method for temperature forecasting. Artificial Intelligence in Engineering, 13(4), 351–357. https://doi.org/10.1016/S0954-1810(99)00007-2

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

21. Corchado, J. M., Borrajo, M. L., Pellicer, M. A., & Yáñez, J. C. (2004). Neuro-symbolic System for Business Internal Control. In Industrial Conference on Data Mining (pp. 1–10). https://doi.org/10.1007/978-3-540-30185-1_1
22. Corchado, J. M., Pavón, J., Corchado, E. S., & Castillo, L. F. (2004). Development of CBR-BDI agents: A tourist guide application. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 3155, pp. 547–559). https://doi.org/10.1007/978-3-540-28631-8
23. Corchado, J., Fyfe, C., & Lees, B. (1998). Unsupervised learning for financial forecasting. In Proceedings of the IEEE/IAFE/INFORMS 1998 Conference on Computational Intelligence for Financial Engineering (CIFEr) (Cat. No.98TH8367) (pp. 259–263). https://doi.org/10.1109/CIFER.1998.690316
24. Costa, Â., Novais, P., Corchado, J. M., & Neves, J. (2012). Increased performance and better patient attendance in an hospital with the use of smart agendas. Logic Journal of the IGPL, 20(4), 689–698. https://doi.org/10.1093/jigpal/jzr021
25. Ester Martinez-Martin, Maria Teresa Escrig, Angel P. Del POBIL (2013). A Qualitative Acceleration Model Based on Intervals. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
26. Felicitas Mokom, Ziad Kobti (2013). Interventions via Social Influence for Emergent Suboptimal Restraint Use. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
27. Fyfe, C., & Corchado, J. M. (2001). Automating the construction of CBR systems using kernel methods. International Journal of Intelligent Systems, 16(4), 571–586. https://doi.org/10.1002/int.1024
28. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
29. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
30. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
31. Heli Koskimaki, Pekka Siirtola (2016). Accelerometer vs. Electromyogram in Activity Recognition. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 3
32. K. S. Jasmine, Gavani Prathviraj S., P Ijantakar Rajashekar, K. A. Sumithra Devi (2013). Inference in Belief Network using Logic Sampling and Likelihood Weighing algorithms. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
33. Karel Macek, Jiri Rojicek, Georgios Kontes, Dimitrios V. Rovas (2013). Black-Box Optimization for Buildings and Its Enhancement by Advanced Communication Infrastructure. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
34. Laza, R., Pavn, R., & Corchado, J. M. (2004). A reasoning model for CBR_BDI agents using an adaptable fuzzy inference system. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 3040, pp. 96–106). Springer, Berlin, Heidelberg.
35. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). Random finite set-based Bayesian filters using magnitude-adaptive target birth intensity. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637788&partnerID=40&md5=bd8602d6146b014266cf07dc35a681e0
36. Marisol García-Valls (2016). Prototyping low-cost and flexible vehicle diagnostic systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 4
37. Méndez, J. R., Fdez-Riverola, F., Díaz, F., Iglesias, E. L., & Corchado, J. M. (2006). A comparative performance study of feature selection methods for the anti-spam filtering domain. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4065 LNAI, 106–120. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-33746435792&partnerID=40&md5=25345ac884f61c182680241828d448c5
38. Muñoz, M., Rodríguez, M., Rodríguez, M. E., & Rodríguez, S. (2012). Genetic evaluation of the class III dentofacial in rural and urban Spanish population by AI techniques. Advances in Intelligent and Soft Computing (Vol. 151 AISC). https://doi.org/10.1007/978-3-642-28765-7_49

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

39. Nadia Alam, Munira Sultana, M.S. Alam, M. A. Al-Mamun, M. A. Hossain (2013). Optimal Intermittent Dose Schedules for Chemotherapy Using Genetic Algorithm. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
40. Naoufel Khayati, Wided Lejouad-Chaari (2013). A Distributed and Collaborative Intelligent System for Medical Diagnosis. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
41. Pablo Chamoso, Fernando De La Prieta (2015). Swarm-Based Smart City Platform: A Traffic Application. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 2
42. Pérez, A., Chamoso, P., Parra, V., & Sánchez, A. J. (2014). Ground Vehicle Detection Through Aerial Images Taken by a UAV. In Information Fusion (FUSION), 2014 17th International Conference on.
43. Prieto, J., Mazuelas, S., Bahillo, A., Fernandez, P., Lorenzo, R. M., & Abril, E. J. (2012). Adaptive data fusion for wireless localization in harsh environments. IEEE Transactions on Signal Processing, 60(4), 1585–1596.
44. Prieto, J., Mazuelas, S., Bahillo, A., Fernández, P., Lorenzo, R. M., & Abril, E. J. (2013). Accurate and Robust Localization in Harsh Environments Based on V2I Communication. In Vehicular Technologies - Deployment and Applications. INTECH Open Access Publisher.
45. Rodriguez-Fernandez J., Pinto T., Silva F., Praça I., Vale Z., Corchado J.M. (2018) Reputation Computational Model to Support Electricity Market Players Energy Contracts Negotiation. In: Bajo J. et al. (eds) Highlights of Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection. PAAMS 2018. Communications in Computer and Information Science, vol 887. Springer, Cham
46. Rodríguez, S., De La Prieta, F., Tapia, D. I., & Corchado, J. M. (2010). Agents and computer vision for processing stereoscopic images. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6077 LNAI). https://doi.org/10.1007/978-3-642-13803-4_12
47. Rodríguez, S., Gil, O., De La Prieta, F., Zato, C., Corchado, J. M., Vega, P., & Francisco, M. (2010). People detection and stereoscopic analysis using MAS. In INES 2010 - 14th International Conference on Intelligent Engineering Systems, Proceedings. https://doi.org/10.1109/INES.2010.5483855
48. Rodríguez, S., Tapia, D. I., Sanz, E., Zato, C., De La Prieta, F., & Gil, O. (2010). Cloud computing integrated into service-oriented multi-agent architecture. IFIP Advances in Information and Communication Technology (Vol. 322 AICT). https://doi.org/10.1007/978-3-642-14341-0_29
49. Saadi Bin Ahmad Kamaruddin, Nor Azura Md Ghanib, Choong-Yeun Liong, Abdul Aziz Jemain (2012). Firearm Classification using Neural Networks on Ring of Firing Pin Impression Images. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
50. Sandrine Mouysset, Ronan Guivarch, Joseph Noailles, Daniel Ruiz (2013). Segmentation of cDNA Microarray Images using Parallel Spectral Clustering. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 1
51. Sebastián Romero, Habib Moussa Fardoun, Victor Manuel Ruiz Penichet, José Antonio Gallud (2013). Tweacher: New proposal for Online Social Networks Impact in Secondary Education. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 1
52. Sérgio Matos, Hugo Araújo, José Luís Oliveira (2013). Biomedical Literature Exploration through Latent Semantics. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
53. Sumit Goyal, Gyanendra Kumar Goyal (2013). Machine Learning ANN Models for Predicting Sensory Quality of Roasted Coffee Flavoured Sterilized Drink. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
54. Tiago Oliveira, José Neves, Paulo Novais (2012). Guideline Formalization and Knowledge Representation for Clinical Decision Support. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
55. Tiancheng Li, Shudong Sun (2013). Online Adapting the Magnitude of Target Birth Intensity in the PHD Filter. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

ISBN: 978-84-9012-861-9

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

800 AÑOS
VNiVERSiDAD D SALAMANCA
1218 ~ 2018

Ediciones Universidad Salamanca