

OpenLaszlo: Programación Flash en OpenSource para la producción de material docente

Christian Moya
Tecnologia Educativa
Eurecamedia (grup UOC)
cmoyas@uoc.edu

David Trelles
Tecnologia Educativa
Universitat Oberta de Catalunya
dtrelles@uoc.edu

Julio 2007

Resumen

OpenLaszlo es una plataforma de código libre XML-nativo para la creación de aplicaciones web. Las aplicaciones resultantes son totalmente compatibles con los principales navegadores del mercado y sistemas operativos. Permite crear como archivo final un objeto Flash, formato SWF. Este framework está pensado para desarrollar aplicaciones multimedia o componentes multimedia con los mismos resultados que el software de pago de Adobe/Macromedia Flash. Con la particularidad de que la escritura de código no es la utilizada en Adobe/Macromedia Flash (ActionScript), si no que es un lenguaje de programación basado en XML-nativo y Javascript, con todas las ventajas que esto supone.

El tratamiento de XML por OpenLaszlo reduce enormemente la complejidad del tratamiento dado por Adobe/Macromedia Flash, por lo que el desarrollo de aplicaciones web y sobre todo en nuestro caso de materiales multimedia de formación, pasa a ser realmente efectivo y mucho más reutilizable por los desarrolladores de contenido. La creación de componentes y de clases reusables, que podemos usar incluyéndolas en cualquier aplicación que desarrollemos, da una nueva perspectiva a la programación de contenido para web. El resultado, como comentamos antes, es una aplicación SWF, es decir que podrá ejecutarse usando el Flash Player que cualquier navegador web lleva instalado por defecto.

1 Introducción

Desde la UOC (Universitat Oberta de Catalunya) y EurekaMedia (grupo UOC) se están desarrollando materiales de E-Learning utilizando el framework OpenLaszlo. La facilidad de desarrollo del software multimedia con esta herramienta permite que la producción de materiales didácticos sea mucho más elevada. El paso de Software de pago para desarrollar aplicaciones o componentes web en código abierto da una nueva perspectiva a la generación y modificación de materiales multimedia que con esta tecnología pasan a ser totalmente reusables, accesibles y totalmente reescalables para cualquier plataforma y navegador. La creación por parte del departamento de tecnología educativa de la UOC de numerosas clases de componentes y numerosas tipologías de ejercicios estándar de E-learning, supone un gran avance en la creación de materiales multimedia a un nivel de desarrollo que con el software de Adobe/Macromedia Flash no podría haberse llegado si no es mediante un gran coste económico y de tiempo y con muchos más programadores de ActionScript de los que para realizar estos componentes se han utilizado. Otro de los puntos fuertes por el que se optó por esta tecnología es la de desarrollar de una manera eficaz aplicaciones totalmente accesibles para personas con discapacidad. El software final es totalmente accesible añadiendo unas cuantas líneas de código, que en el caso de haber utilizado Adobe/Macromedia Flash habría tenido un coste demasiado alto como para que este desarrollo fuera viable.

Las API's que permiten crear animaciones, dibujar vectorialmente, mostrar datos, comunicar el cliente con el servidor, también facilitan la lectura de contenidos a partir de XML, que posteriormente los profesores modifican fácilmente para la creación de diferentes topologías de ejercicios.

Actualmente se está desarrollando la versión OpenLaszlo 4.0 que generará el archivo compilado en DHTML. Una aplicación OpenLaszlo estará desarrollada en OOP, por lo que el resultado final puede estar en un solo archivo o en múltiples archivos reutilizados como clases y librerías del principal.

2 Arquitectura

OpenLaszlo provee dos maneras de desplegar la aplicación:

- **SOLO:** (Standalone OpenLaszlo Output). Las aplicaciones son precompiladas por cualquier servidor web HTTP. Soporta la integración de datos mediante XML sobre HTTP y simplifica drásticamente los requisitos del centro de datos y los minimiza los costes de servicio.
- **OpenLaszlo Server:** Las aplicaciones son compiladas y puestas en caché con un J2EE o Java Servlet Container Environment. Este despliegue soporta aplicaciones que requieren la

integración de datos SOAP, XML-RPC o JAVA-RPC o requieran conexiones persistentes, entre otros. Gran escalabilidad, es decir, que tiene la habilidad para, o bien gestionar el crecimiento de trabajo de manera fluida, o bien para estar preparado para hacerse mas grande sin pérdida de calidad en los servicios que se ofrecen.

En la *figura 2*, podemos observar un diagrama de la arquitectura de OpenLaszlo que nos muestra las dos formas de trabajar con esta tecnología, como hemos comentado anteriormente:

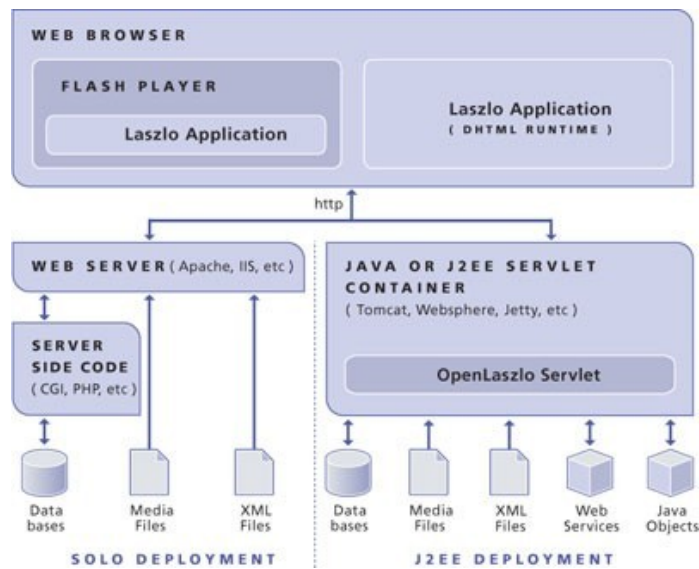


Figura 2: Arquitectura SOLO y servidor-cliente

3 Instalación

Para poder compilar nuestro código utilizamos:

- Entorno JAVA (JRE 1.6)
- Lombok (IDE de Eclipse)
- IDE4LASZLO 0.2 (plugin de Eclipse)
- OpenLaszlo 3.2 (Server, Explorer)

De esta manera el entorno de programación es mucho más eficiente y agradable para el desarrollador, ya que el Lombok unido con el IDE de Laszlo está realmente optimizado para su compilado. OpenLaszlo no tiene un editor gráfico, como tiene Adobe Flash, Microsoft Expression u otros editores de contenidos multimedia. En este caso, deberemos introducir todo el código y compilar posteriormente para saber si nuestro código es correcto y si hemos pintado como queríamos nuestro archivo final swf. Tenemos la opción de compilar nuestro archivo final junto con el debugger que lleva incorporado el propio Laszlo. De esta manera siempre podemos debuggear la aplicación sin tenerlo que

ejecutar manualmente desde Lomboz, como podemos ver en la *figura 3*:



Figura 3. Debugger OpenLaszlo para el navegador web.

Por otro lado, el OpenLaszlo permite la compilación por línea de comandos por lo que al partir de un archivo desarrollado en Laszlo (lzx) podemos compilar y automatizar mediante herramientas como *Ant* para conseguir una alta producción de materiales docentes (véase *figura 4* como ejemplo).

```
C:\WINDOWS\system32\cmd.exe
C:\>cd "Program Files"
C:\Program Files>cd "OpenLaszlo Server 3.2"
C:\Program Files\OpenLaszlo Server 3.2>cd bin
C:\Program Files\OpenLaszlo Server 3.2\bin>dir
El volumen de la unidad C es Disc Sistema
El número de serie del volumen es: 18F1-5EC7

Directorio de C:\Program Files\OpenLaszlo Server 3.2\bin
22/06/2007 13:39 <DIR> .
22/06/2007 13:39 <DIR> ..
18/06/2007 10:05          2.505 ejemplo1.lzx
24/03/2006 22:38           919 lzc
24/03/2006 22:38          759 lzc.bat
24/03/2006 22:38          986 lzdc
24/03/2006 22:38          767 lzdc.bat
24/03/2006 22:38          916 lzmc
24/03/2006 22:38          756 lzmc.bat
              7 archivos          7.608 bytes
              2 dirs      8.472.670.208 bytes libres

C:\Program Files\OpenLaszlo Server 3.2\bin>lzc ejemplo1.lzx ejemplo1.swf
```

Figura 4: Compilación por línea de comandos

4 Código de programación

Una aplicación LZX se ejecuta en un objeto visual llamado canvas (lienzo) que es básicamente un trozo de pantalla. En el canvas interactúan cajas autónomas llamadas views o vistas. Estas vistas pueden ser anidadas lógicamente y visualmente y tienen docenas de atributos programables incluyendo tamaño, posición, color de fondo, opacidad, etc... En el siguiente código podemos observar una estructura básica de un archivo LZX donde se muestra un texto de color rojo en el centro del lienzo (canvas):

```
<?xml version="1.0" encoding="UTF-8" ?>
<canvas width="100%" height="100%">
  <view align="center" y="{canvas.height}">
    <text fgcolor="red">Hola mundo, este es mi primer script</text>
  </view>
</canvas>
```

Las vistas pueden ser utilizadas para alojar recursos, como por ejemplo una imagen o un video, y pueden también ser enlazadas de forma dinámica a cualquier conjunto de datos con formato XML, bien se trate de un fichero o una URL desde la que un servicio nos proporcione datos en XML a partir de una base de datos o cualquier otra fuente.

Inicialmente, hemos comentado que una aplicación OpenLaszlo estará desarrollada en OOP, por lo que el resultado final puede estar en un solo archivo o en múltiples archivos reutilizados como clases y librerías del principal. En el código a continuación se muestra el archivo LZX principal:

En nuestra primera línea de código, como en todo XML, declaramos el código como tal y el tipo de codificación, en este caso UTF-8.

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

Todo código LZX final deberá iniciarse con canvas y finalizarse de la misma manera:

```
<canvas>
```

Para poder trabajar con los archivos multimedia deberemos declararlos de la siguiente manera:

```
<resource name="ahorcado0" src="../../resources/ahorcado0.swf" />
<resource name="ahorcadol" src="../../resources/ahorcadol.swf" />
.
.
.
<resource name="ahorcado10" src="../../resources/ahorcado10.swf" />
<resource name="tick" src="../../resources/tick.swf" />
<resource name="wrong" src="../../resources/wrong.swf" />
<resource name="Ans01" src="../../resources/Ans01.swf"/>
<resource name="speaker" src="../../resources/louder.png"/>
<resource name="flasher" src="../../resources/flasher.jpg"/>
<resource name="fondoREC" src="../../resources/fondoREC.gif"/>
<resource name="linea" src="../../resources/lineaTxt.jpg"/>
```

En este caso, hemos creado una serie de clases que deberemos incluir en nuestro archivo principal. Los componentes predefinidos por el propio OpenLaszlo no son necesarios incluirlos, ya que el compilar los incluye de forma automática:

```
<!-- LLAMADAS A CLASES -->
<include href="../../resources/addons.lzx"/>
<include href="../../resources/checkBox.lzx"/>
<include href="../../resources/DragDrop.lzx"/>
<include href="../../resources/dropdown.lzx"/>
<include href="../../resources/feedback.lzx"/>
```

```

<include href="../../resources/gapFill.lzx"/>
<include href="../../resources/radiogroups.lzx"/>
<include href="../../resources/text.lzx"/>
<include href="../../resources/writingLayout.lzx"/>
<include href="../../resources/buttons.lzx"/>
<include href="../../resources/windows.lzx"/>
<include href="../../resources/hanged.lzx"/>
<include href="../../resources/resources.lzx"/>

```

Y como en todo tipo de programación declaramos las variables globales necesarias dentro siempre de la etiqueta `<script>`:

```

<!-- DECLARACIÓN DE VARIABLES GLOBALES -->

```

```

<script>
MAX=0;
cont=0;
cont_si=0;
clicked=false;
ended=false;
chkEnd=false;
ShownumFeedBack = 0;
feedBack=2;
clickedAns=false;
txtRadiobtn=false;
txtRadioGroup=true;
idN=0;
idT = "";
idTPass=0;

//variables del HangedGame
var finalizado = false;
var acertados = 0;
var posicio = 1;
var numLletres = 0;
var palabra = "";
var titulo_palabra = "";
var total_errores = 0;
var incremento_errores = 0;
var acierto = 3;
var total_errores = 0;
var error = 0
var errores = 0;
var entrada = "off";
var algun_acierto = false;
var num = 1;
var imatge_titol = "";
var introducidas = Array();
var incremento_letras = 0;
var no_comprobar = false;
var aciertos_totales = 0;
var preguntas_totales = 0;
var fallados = 0;
var total_palabras = "";
var tipo = "";

</script>

```

A la hora de crear métodos y funciones, lo podemos hacer tanto dentro de cada tag como de

forma externa, que en este último caso afectará a todo el canvas.

```
<method name="onidle" event="onidle" reference="LzIdle">
    <![CDATA[
        if(idT!="") {
            if(idTPass==0) {
                idTPass=1;
            }
        }
    ]]>
</method>
```

Cómo vemos, aquí podemos usar las típicas acciones que se utilizan en ActionScript, como podría ser: `OnClickEvent(enterframe){...}`

```
<!-- REGISTRO DE EVENTOS -->
<method name="registerResultEvent" args="obj">
    <![CDATA[
        obj.del = new LzDelegate( obj, "onResultEvent" );
        obj.del.register( canvas , "resultEvent" );
    ]]>
</method>
<method name="registerViewEvent" args="obj">
    <![CDATA[
        obj.del = new LzDelegate( obj, "onViewEvent" );
        obj.del.register( canvas , "viewEvent" );
    ]]>
</method>
```

Control de "eventos" usando programación en OpenLaszlo.

Una de la partes importantes será cargar correctamente nuestro XML. Para declararlo y cargarlo como un recurso siempre lo haremos de la siguiente manera:

```
<!-- CARGAMOS EL XML -->
<dataset name="exer_XML" src="../resources/data.xml"/>
```

En este caso, hemos creado una clase que carga una serie de archivos *swf* declarados anteriormente. Dentro de la librería de clases, podemos crear tantas subclases como queramos, aunque siempre es recomendable trabajar con distintos archivos de clases. El código de la clase es el siguiente:

```
<library>
<class name="load_images" defaultplacement="content" clip="true">
    <view width="200" height="200" id="imatge" visible="false">
        <method name="getResource">
            <![CDATA[
                var dtp = imatge_titol;
                var dtpSpl = dtp.split(".");
                var tipus = dtpSpl[1];
                this.setAttribute("visible", true);
                this.setAttribute("resource", dtpSpl[0].concat("_"+tipus));
            ]]>
        </method>
    </view>
</class>
</library>
```

El código a continuación sería propiamente la parte donde pintaremos nuestro *canvas* según los tags y métodos utilizados. El tag más básico e importante para pintar algo en pantalla es el `<view>`. Hay que recordar que toda clase o tag por defecto deberá iniciarse y cerrarse correctamente, en caso contrario, no se podrá compilar correctamente. En este caso simulamos dos frames disintintos, a la izquierda un ejercicio de tipo ahorcado y a la derecha mostramos los diferentes *feedbacks*.

Creamos una vista general con el ancho de todo el canvas:

```
<view width="{canvas.width}">
```

Un de los tags importantes para poder distribuir las diferentes vistas de forma sencilla es el `<simplelayout>`. En este caso, deberemos indicar el eje, es decir, si las vistas se distribuyen de en el eje de las *x* o en el eje de las *y*. Y finalmente, el espaciado entre estas:

```
<simplelayout axis="x" spacing="0"/>
  <view width="{canvas.width-lateral.width}" height="{canvas.height}">
    <view>
      <simplelayout axis="y" spacing="5"/>
        <view width="{parent.parent.width}" y="20">
          <simplelayout axis="y" spacing="10"/>
            <titulo/>
            <linea/> //Llamada a las clases o componentes
            <enunciado/>
            <linea/>
            <hangedgame />
          </view>
        </view>
      <vscrollbar/>
    </view>
  </view>
```

La siguiente vista sería el frame lateral que nos muestra los diferentes *feedbacks* y donde cargamos los archivos multimedia necesarios:

```
<view id="lateral" width="240" align="right" height="{canvas.height}"
bgcolor="#8FBBF7">
  <view resource="fondoREC" y="10" align="center">
    <view id="textFeed" name="textFeed" x="{lateral.width-465}">
      <load_images y="30" x="{lateral.width}"/>
      <alert_immediate y="150" x="{lateral.width}"/>
    </view>
  </view>
</view>
</view>
```

Cerramos el *canvas* y ya tendremos nuestro archivo preparado para el test de errores y el compilado final:

</canvas>

El archivo swf final se vería de la siguiente manera (figura 5):

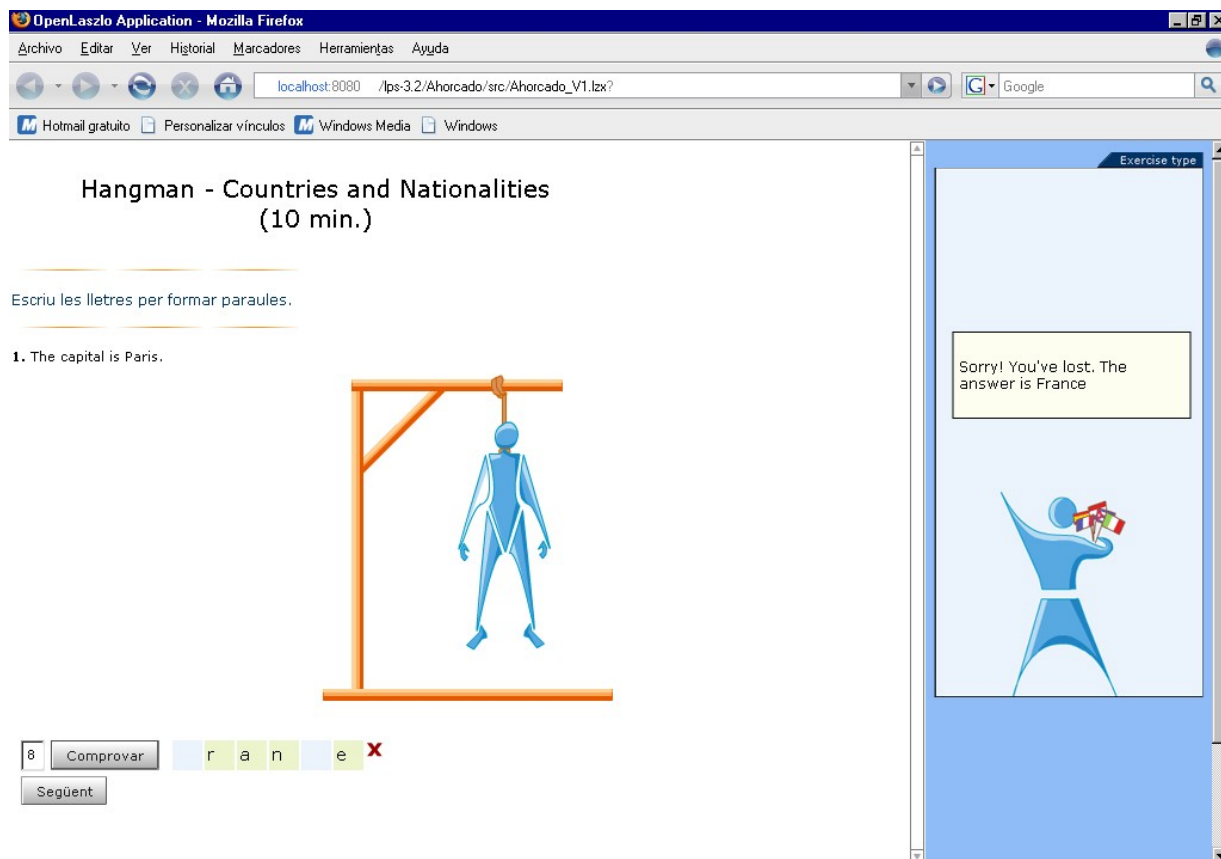


Figura 5: Arhivo final generado en formato swf

5 Conclusiones

A pesar que esta tecnología aun no ha evolucionado lo suficiente, las posibilidades son lo suficientemente óptimas para generar contenido multimedia de gran calidad. Gracias a todos los componentes que OpenLaszlo nos ofrece, en poco tiempo podemos generar un aplicación realmente impactante, tanto visualmente como en su funcionamiento de cara al usuario final. Las posibilidades son casi infinitas como podemos ver en la propia web de [OpenLaszlo](#), en las que podemos ver cómo han implementado gestores de correo electrónico, plataformas de E-commerce y demás aplicaciones.

En la UOC la producción de material de formación es continua y de una alta producción. El hecho de poder trabajar con esta tecnología ha permitido generar material de forma rápida y eficiente a pesar de los inconvenientes iniciales que puede suponer trabajar con un entorno Eclipse sin la posibilidad de tener un editor gráfico para trabajar con más comodidad. Si hemos confiado realmente en esta tecnología es por el hecho que está en constante mejoría y evolución. Las nuevas versiones, *VI Jornades de Programari Lliure*. <http://jornadespl.upc.es>

basadas en AJAX, trabajan con HTML dinámico y no es necesario el plugin de Adobe Flash, como en versiones anteriores. En nuestro caso, aun no necesitamos trabajar con HTML dinámico para generar ejercicios y material docente, aunque está contemplado para futuras aplicaciones.

En conclusión, apostar por tecnologías OpenSource para generar contenidos multimedia de cualquier tipo está a la orden del día y las posibilidades aumentan de forma exponencial. Aunque si cabe decir y admitir que tienen aun ciertas limitaciones técnicas y visuales pero con el tiempo ser irán optimizando y mejorando como hemos podido observar hasta ahora.

Referencias

- [1] OpenLaszlo: <http://www.openlaszlo.org> (Site Oficial)
- [2] Wiki OpenLazlo : wiki.openlaszlo.org (Documentación e instalación)
- [3] En castellano: openlaszlo.net (Documentación, instalación y tutoriales)
- [4] Lomboz: lomboz.objectweb.org (Entorno de trabajo)