ÍNDICE

CAPÍTULO 7. SISTEMAS DE PLANTILLAS 3

7.1 Separación de la lógica de negocio y de la presentación. 3
7.1.1 La arquitectura de tres capas. 4
7.2 Los sistemas de plantillas. 5
7.2.1 Funcionamiento general. 5
7.2.2 Composición y participantes. 6
7.2.3 Acciones habituales. 6
7.3 El paquete HTML_Template_Sigma. 8
7.3.1 Sintaxis de las plantillas. 8
7.3.2 Funcionalidad de Sigma. 9
7.3.3 Ejemplo de uso. 11
7.3.4 Documentación adicional. 13

CAPÍTULO 7. SISTEMAS DE PLANTILLAS

Durante muchos años, el software se ha desarrollado mediante arquitectura cliente-servidor para un número determinado de dispositivos. Sin embargo, la aparición y difusión de nuevos dispositivos como han sido *PDAs* o teléfonos móviles, junto con la llegada de Internet a un sector más amplio de la población utilizando tecnologías como *HTML* o *WAP* ha provocado la aparición de nuevas preguntas en relación con el desarrollo de software: ¿Es apropiado que las interfaces de usuario y lógica de negocio estén unidas? ¿Es conveniente que exista tanta dependencia entre la presentación del software y la lógica del mismo?

En términos generales, la respuesta es no. En un mundo de desarrolladores en el que los lenguajes y el software multiplataforma son un objetivo por si mismo, ¿Por qué pensar que la interfaz en estas plataformas va a ser común? Si disponemos de un software de gestión de almacenes, e introducimos *PDAs* con lectores de códigos de barra, ¿Por qué debemos volver a desarrollar un nuevo software sólo porque haya variado el dispositivo de trabajo? ¿No es más eficiente crear únicamente las interfaces apropiados para este nuevo dispositivo?

Si con esta breve introducción podemos apreciar la importancia de esta separación, démosle pues a la presentación la autonomía que se merece.

7.1 Separación de la lógica de negocio y de la presentación.

En el desarrollo de software, la separación de capas se establece en base a la arquitectura de software empleada. En la actualidad, las arquitecturas de programación más extendidas son:

- **Monolítica**, el software se estructura en grupos funcionales altamente acoplados.

- **Cliente-Servidor**, el software reparte su carga entre dos partes independientes. Este modelo no establece una ubicación para la capa de negocio, que habitualmente se encuentra dividida entre cliente y servidor.

- **Arquitectura de tres capas**, generalización de la arquitectura cliente-servidor, en la que se realiza la división en tres categorías: capa de presentación, capa de cálculo o de lógica de negocio, y la capa de almacenamiento. Siguiendo dicho modelo, cada capa únicamente tiene acceso a la capa inmediatamente posterior.

7.1.1 La arquitectura de tres capas.

Las responsabilidades de las capas software que componen esta arquitectura son:

- **Capa de presentación**: comprende las funciones de navegabilidad del sistema, validación de datos de entrada, formateo de datos de salida, internacionalización, renderizado de la presentación.

- **Capa de negocio**: integra el conjunto de reglas de negocio abstraídas del mundo real por medio del análisis funcional.

- **Capa de persistencia**, es responsable de las tareas de inserción, eliminación, actualización y demás, propias de los sistemas de gestión de bases de datos.

7.1.1.1 Consecuencias: beneficios e inconvenientes.

Los beneficios de la arquitectura de tres capas respecto del modelo arquitectónico cliente-servidor son:

- **Mayor independencia**, disponer de una capa de presentación independiente disminuye la acoplabilidad del código y, por tanto, la propagación de cambios en el código del software.

- **Disminución de costes de mantenimiento**, la separación de capas provoca que los cambios en capas inferiores no repercutan sobre el código de capas superiores, lo que reduce los costes de mantenimiento del software.

- **Facilidad para multiplataforma**, la separación de capas facilita la utilización de un mismo software en distintas plataformas, requiriendo únicamente el desarrollo de nuevas interfaces.

Además, ubicar las capas en equipos distintos obtenemos ventajas adicionales:

- **Balanceo de la carga**, la separación de la lógica de negocios y la lógica de aplicación en equipos puede conllevar a una mejor distribución de la carga es responsable de las tareas de inserción, eliminación, actualización y demás, propias de los sistemas de gestión de bases de datos.

Por otra parte, la utilización de la arquitectura de tres capas puede provocar algunos inconvenientes, habituales con la adición de nuevas capas de software:

- **Mayor coste inicial**, la utilización de un mayor número de capas provoca que la aplicación incluya más servicios a mantener, lo que provoca un mayor coste inicial en el desarrollo del software. Este inconveniente es especialmente notable en pequeñas aplicaciones.

- **Mayor probabilidad de errores**, la existencia de un mayor número de componentes puede conllevar un aumento en el número de errores.

- **Mayor conocimientos de programación**, la programación de un mayor número de capas y servicios, así como el establecimiento de los métodos y protocolos de comunicación entre ellos requiere de un mayor conocimiento y experiencia por parte de los programadores, especialmente en el caso de sistemas distribuidos.

7.2 Los sistemas de plantillas.

El principal objetivo de los sistemas de plantillas es lograr la separación entre la programación y la interfaz de usuario.

De este modo, es posible diferenciar más claramente entre el trabajo propio de los programadores y de los diseñadores.

7.2.1 Funcionamiento general.

Habitualmente, los sistemas de plantillas utilizan ciertas etiquetas prefijadas para marcar el diseño, y poder realizar acciones relacionadas con la programación sobre ellas.

Aunque los sistemas más sencillos únicamente permiten realizar sustituciones sobre el diseño, que normalmente se emplean para plasmar la información extraída de la base de datos sobre la interfaz

de usuario, los sistemas actuales permiten realizar acciones de inclusión, iteración e incluso invocación a funciones.

7.2.2 Composición y participantes.

Este tipo de sistemas suele tener tres componentes:

- **Fichero de plantilla**. El fichero de plantilla contiene el esqueleto o estructura general que se transformará en el resultado final. Almacena la esencia y forma que se obtendrá en el resultado final, y utiliza marcas especificas que pueden transformarse externamente para adaptarla a las necesidades específicas en cada momento.

- **Fichero de transformación**. El fichero de transformación toma como origen a un fichero de plantilla, y utiliza las marcas para especificar como se le debe dar forma para obtener el resultado final deseado.

- **Proceso de transformación**. El proceso de transformación es aquel que utilizando la información de los ficheros de plantilla y de transformación, realiza las modificaciones especificadas generando el resultado final.

En el caso de *PHP*, por lo general, estos tres componentes son los siguientes:

- **Fichero de plantilla**. Es un fichero con o sin extensión concreta, que contiene un fichero *html* semicompleto, etiquetado con las marcas específicas del sistema de plantillas empleado.

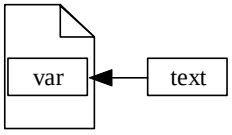
- **Fichero de transformación**. El fichero de transformación suele ser un fichero de código *PHP* que, utilizando el sistema de plantillas escogido, carga el fichero de plantilla y realiza las transformaciones obteniendo la página *html* resultante.

- **Proceso de transformación**. El proceso de transformación es el propio motor *PHP* impulsado por el servicio web.

7.2.3 Acciones habituales.

7.2.3.1 Sustitución de variables.

Como su propio nombre indica, la sustitución de variables realiza una modificación sobre una marca concreta del fichero de plantilla.

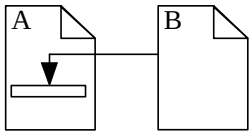


Esta acción se emplea para personalizar los resultados, y adaptar las páginas a la acción concreta. Por ejemplo, al logarse un usuario

puede utilizarse el nombre de usuario para indicar que dicho usuario ha iniciado la sesión mediante una línea del estilo “Usuario: pgomez”. También en el caso de que se estén modificando los datos de un campo, puede utilizarse para mostrar el contenido inicial de dicho campo.

7.2.3.2 Inclusión de ficheros.

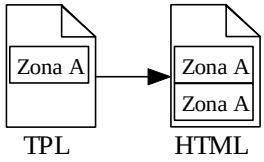
La inclusión de ficheros consiste en cargar una plantilla dentro de otra, añadiendo de esta manera el texto de la última dentro de la primera.



El caso más extendido es la inclusión de cabeceras o pies de página, que deben ser uniformes en todo el sitio web. En páginas que muestran distintas zonas o frames puede utilizarse para aportar mayor independencia a cada zona.

7.2.3.3 Establecimiento de zonas o bloques.

La señalización de zonas sirve para poder realizar transformaciones específicas en dicha zona, como puede ser que no aparezca o que aparezca en múltiples ocasiones.

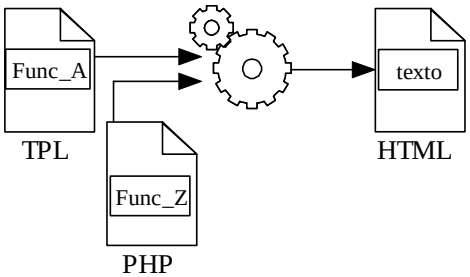


Entre los casos típicos, podemos encontrar los precios de los sitios web para mayoristas, en que normalmente si no te has logado no se permite ver los precios y se muestra un botón que redirige a la página de inscripción.

Otro caso típico es un listado, en el que las filas que muestran la información siguen el mismo patrón, y se generan una vez con la información de cada tupla de la base de datos.

7.2.3.4 Asociación de funciones.

La asociación de funciones sirve para que una etiqueta específica del fichero de plantilla se asocie con una función ejecutable del fichero de transformación de forma que el procesador pueda ejecutar una acción concreta, llegado el momento.



7.3 El paquete HTML_Template_Sigma.

El paquete *HTML_Template_Sigma* se define como la implementación de una *API* para la integración de plantillas, y su objetivo es la renderización de plantillas para la construcción de ficheros automatizados, que en nuestro caso serán páginas *HTML*.

7.3.1 Sintaxis de las plantillas.

La sintaxis de las plantillas *Sigma* define un conjunto de elementos especiales:

- **Variables o sustituciones**, conocidos como *placeholders*, consisten en una serie de etiquetas que son sustituidas por el texto deseado en el momento de la *renderización*. Conceptualmente corresponden a huecos pendientes de rellenar. La sustitución se realiza mediante las funciones *setVariable()* y *setGlobalVariable()*. En caso de no realizar ninguna sustitución, el hueco o *placeholder* quedaría vacío.

```
Sintaxis: {[0-9A-Za-z_-]+}
```

- **Definición de bloques**, la utilización de bloques es uno de los principales mecanismos de las plantillas. Toda la información de la plantilla se estructura en bloques, siendo englobada en su totalidad por el bloque virtual "__global__". Cada bloque puede *renderizarse* tantas veces como sea necesario, pudiendo no *renderizarse* ninguna vez y siendo, por tanto, eliminada. Los bloques pueden anidarse, con la única obligación de *renderizar* los bloques interiores antes que los exteriores. La función de *renderización* de los bloques es *parse().*

```
<!-- BEGIN [0-9A-Za-z_-]+ -->
... block content ...
<!-- END [0-9A-Za-z_-]+ -->
```

- **Inclusión de ficheros externos**, es posible incluir dentro de una plantilla ficheros externos, que a su vez deberán ser plantillas. El proceso de inclusión de ficheros externos se realiza antes que las sustituciones y *renderizado*, por lo que no es posible sustituir el nombre de un fichero a incluir en la plantilla. Sin embargo, la función *addBlockfile()* puede permitirnos la inclusión de ficheros que no conocemos de antemano.

```
... some content ...
<!-- INCLUDE filename.html -->
... some more content ...
```

- **Funciones de plantillas**, *Sigma* permite la utilización de funciones dentro de las mismas plantillas. Dado que las plantillas no incluyen código ejecutable propio, estas funciones se emplean como *placeholders*, que deben enlazarse con funciones *PHP* existentes durante la *renderización* de la plantilla. Para asociar una función de plantilla con una función *PHP* se emplea el método *setCallbackFunction()* teniendo en cuenta que el número de parámetros de las funciones asociadas debe coincidir, y que el texto devuelto por la función *PHP* rellenará el hueco dejado por la función de la plantilla. Los argumentos introducidos en las funciones de plantilla pueden ser a su vez *placeholders*.

```
... some content ...
func_h1("embedded in h1")
... some more content ...
```

- **Abreviaturas de funciones**, existe sintaxis alternativa a la escritura de funciones de plantillas que no se estudiará durante el curso.

7.3.2 Funcionalidad de Sigma.

La funcionalidad de *Sigma* es muy amplia, por lo que vamos a analizarla por grupos.

El primer grupo sería el manejo de variables o *placeholders*:

- ***clearVariables***, cancela la sustitución de todas las variables no globales del bloque. Se establece como mecanismo para que tras *renderizar* un bloque, las variables sustituidas no se repitan en bloques posteriores.

- ***getPlaceholderList***, devuelve la lista de los *placeholders* contenidos en el bloque indicado.

- ***placeholderExists***, comprueba si existe un *placeholder* con el nombre indicado dentro de la plantilla o de un bloque en concreto.

- ***setGlobalVariable***, establece la sustitución de una variable global. Al contrario que las normales, el valor de las variables globales no se vacía tras cada sustitución si no que permanece hasta que se indica lo contrario.

- ***setVariable***, establece la sustitución de un *placeholder* por un valor concreto.

La segunda agrupación determina el acceso y *renderización* de los bloques:

- ***blockExists***, comprueba si existe un bloque con el nombre.
- ***getBlockList***, devuelve la lista de los bloques contenidos en la plantilla o en un bloque concreto.
- ***getCurrentBlock***, devuelve el nombre del bloque actual.
- ***parse***, completa la *renderización* del bloque indicado.
- ***parseCurrentBlock***, completa la *renderización* del bloque actual. Es importante tener en cuenta que *Sigma* no mantiene una pila con los bloques que están en uso por lo que invocar dos veces seguidas a *parseCurrentBlock* no tendrá sentido, ya que *Sigma* no considerará que halla cambiado el bloque a *renderizar*.
- ***setCurrentBlock***, establece el bloque activo, es decir, el bloque sobre el que se realiza la sustitución de variables.
- ***touchBlock***, impide que el bloque desaparezca pese a no haber sustituido ninguna variable contenida en el mismo. Esta función es importante ya, habitualmente, aquellos bloques en los que no se ha realizado ninguna sustitución de variables son eliminados al *renderizarlos*.
- ***hideBlock***, su objetivo es el opuesto a *touchBlock*. En este caso, *hideBlock* oculta o elimina el bloque indicado pese a todas las sustituciones que se hubieran realizado en él.

La tercera agrupación determina las sustituciones especiales:

- ***addBlock***, sustituye un *placeholder* por un nuevo bloque. En la invocación se proporciona el nombre del nuevo bloque, y el contenido del mismo. Esto nos permite obtener una creación más dinámica de plantillas.
- ***addBlockfile***, de forma similar a *addBlock*, permite sustituir un *placeholder* por un nuevo bloque. A diferencia del caso anterior, *addBlockfile* utiliza como contenido del bloque una nueva plantilla almacenada en el fichero indicado.
- ***replaceBlock***, reemplaza el contenido de un bloque por el texto suministrado en la invocación a la función.
- ***replaceBlockfile***, reemplaza el interior de un bloque por el contenido del fichero indicado.

El cuarto grupo contiene las funciones principales de *renderización*:

- ***constructor HTML_Template_Sigma***, además de crear el objeto principal para la *renderización* de plantillas, opcionalmente

permite establecer el directorio raíz de las plantillas, así como el directorio en que se realizarán las tareas de caché.

- ***loadTemplateFile***, su objetivo es cargar el contenido del fichero de la plantilla.

- ***setTemplate***, permite cargar la plantilla directamente a través de una cadena de texto. El resultado es el mismo que *loadTemplateFile* con la diferencia del origen de la plantilla.

- ***get***, devuelve una cadena con el resultado de la *renderización*. Puede emplearse para devolver en su lugar la *renderización* de un bloque concreto.

- ***show***, envía al usuario final el resultado de la *renderización* efectuada. Es equivalente a enviar al usuario el resultado de la función *get*. Al igual que *get*, puede usarse con un bloque en concreto.

El último grupo comprende las opciones y configuración:

- ***errorMessage***, dado un código de error, devuelve el texto de error asociado.

- ***setOption***, permite establecer las opciones de *renderización*.

- ***setCacheRoot***, establece el directorio de trabajo de caché, también conocido como directorio de plantillas "preparadas". Este valor puede definirse directamente mediante el constructor de la clase.

- ***setRoot***, permite establece la carpeta raíz de las plantillas. Este valor puede definirse directamente mediante el constructor de la clase.

Además, nos queda una última función que no corresponde a ninguna de las agrupaciones anteriores:

- ***setCallbackFunction***, establece una relación entre una función de plantilla y una función *PHP*, sustituyendo la primera por el texto devuelto por esta segunda.

7.3.3 Ejemplo de uso.

Como se puede observar en el siguiente fichero de script, en primer lugar se crea un objeto del tipo HTML_Template_Sigma, y en segundo lugar se procede a cargar el fichero de plantilla.

Posteriormente se realiza una asociación de funciones, y se procede con la sustitución de variables creando una fila distinta en cada ocasión, garantizando mediante la función *parse* que se ha completado el bloque.

Finalmente, se invoca el método *show* que procederá a mostrar por pantalla el resultado final.

```php
<?php

require_once 'HTML/Template/Sigma.php';

function toggle($item1, $item2) {
    static $i = 1;
    return $i++ % 2? $item1: $item2;
}

$data = array (
    array("Stig", "Bakken"),
    array("Martin", "Jansen"),
    array("Alexander", "Merz") );

$tpl =& new HTML_Template_Sigma('.');
$tpl->loadTemplateFile('table.html');
$tpl->setCallbackFunction('bgcolor', 'toggle');

foreach ($data as $name) {
    $tpl->setVariable(array(
                'first_name'  => $name[0],
                'last_name' => $name[1] ));

    $tpl->parse('table_row');
}

// Saca por pantalla el resultado
$tpl->show();
?>
```

Podemos introducir el siguiente texto en el fichero "*table.html*":

```html
<html>
  <body>
    <table cellpadding="2" cellspacing="0" border="1">
<!-- INCLUDE table_header.html -->
<!-- BEGIN table_row -->
      <tr>
        <td bgcolor="func_bgcolor('#CCCCCC', '#F0F0F0')">{first_name}</td>
        <td bgcolor="func_bgcolor('#CCCCCC', '#F0F0F0')">{last_name}</td>
      </tr>
<!-- END table_row -->
    </table>
  </body>
</html>
```

En su interior podemos ver que se incluye el fichero "*table_header.html*", y que posteriormente se crea el bloque *table_row* que contendrá las distintas filas de la tabla. Dentro de este bloque, tenemos la función *bgcolor* que el fichero de script ha asociado con

una función *php*, así como dos variables: *first_name* y *last_name*; que se sustituirán por su valor adecuado en cada ocasión.

Y finalmente, tenemos el fichero "*table_header.html*" referenciado en el anterior fichero de plantilla, que contendrá el siguiente texto:

```
    <tr>
      <th>First name</th>
      <th>Last name</th>
    </tr>
```

Aplicando todo lo anterior, obtendríamos como resultado el siguiente fichero HTML.

```
<html>
  <body>
    <table cellpadding="2" cellspacing="0" border="1">
      <tr>
        <th>First name</th>
        <th>Last name</th>
      </tr>

      <tr>
        <td bgcolor="#CCCCCC">Stig</td>
        <td bgcolor="#CCCCCC">Bakken</td>
      </tr>
      <tr>
        <td bgcolor="#F0F0F0">Martin</td>
        <td bgcolor="#F0F0F0">Jansen</td>
      </tr>
      <tr>
        <td bgcolor="#CCCCCC">Alexander</td>
        <td bgcolor="#CCCCCC">Merz</td>
      </tr>

    </table>
  </body>
</html>
```

7.3.4 Documentación adicional.

La documentación oficial podemos encontrarla en la dirección: pear.php.net/manual/en/package.html.html-template-sigma.php.