UNIVERSITÄT PADERBORN

Seminararbeit

Separierung der Darstellung durch Templates

Barbara Zimmermann

Seminararbeit

Separierung der Darstellung durch Templates

im Rahmen der Projektgruppe
„Generierung von Web-Anwendungen aus visuellen Spezifikationen“

Universität Paderborn

betreut von
Dr. Michael Thies

eingereicht von
Barbara Zimmermann

Paderborn, 27.05.2004

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Bei der Implementierung professioneller Webseiten ist Dynamik innerhalb dieser Projekte mittlerweile Standard. Diese dynamischen Projekte werden üblicherweise durch serverseitige Skripte erzeugt, wobei das Ergebnis dieser Abfragen in einem Webbrowser ausgegeben wird. Für die dabei involvierten Designer wird es dadurch immer schwerer, das Layout manuell zu pflegen, da in den dabei verwendeten PHP-Skripten, die die Dynamik ermöglichen, oft sowohl PHP- als auch HTML-Code verwendet wird. Erkennbar ist die Mischung dieser zwei Sprachen an dem Vorkommen von vielen Ausdrücken wie beispielsweise

```
echo „<table width=’80%’>\n\t<tr bgcolor=’$farbe’>“
```

oder einer Mischung aus solchen echo/HTML-Anweisungen und ganzen HTML-Blöcken. Das entstehende Skript ist nach der Entwicklung und auch schon währenddessen oft schwer verständlich und unübersichtlich.
Dieses Problem kann umgangen werden, indem für dynamische Ausgaben Templates verwendet werden. Die Templates bestehen aus mehreren Template-Dateien, die als eine Art Schablone zu betrachten sind und die Ausgabe der Inhalte für den Benutzer steuern. Es entsteht dabei der Vorteil, dass das in den Template-Dateien benutzte HTML vom PHP-Skriptcode getrennt wird und so auch ein nicht PHP kundiger Designer in die Lage versetzt wird, sein Layout für das Projekt zu entwickeln und später auch zu verändern.

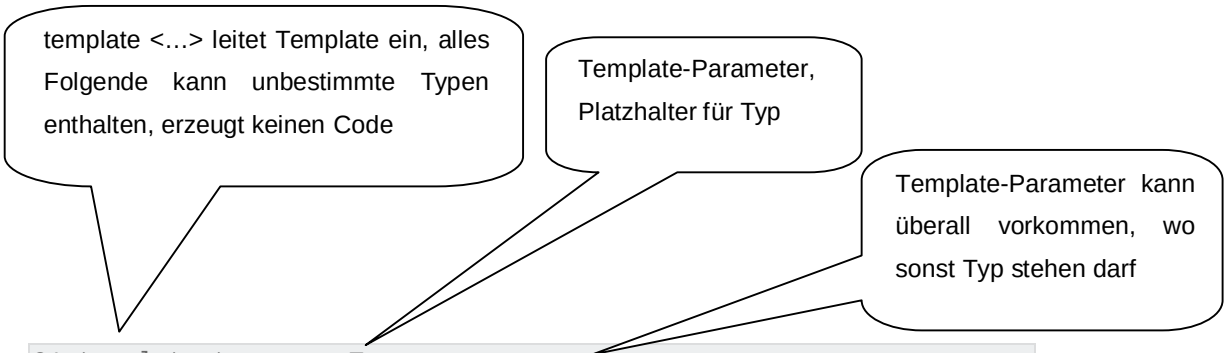
2 Template Engines

2.1 Templates

Das Problem, das zur Verwendung von Templates führt, ist nicht nur die vorher genannte Mischung von PHP- und HTML-Code, sondern auch die Codeduplizierung, die auch bei der Verwendung anderer Programmiersprachen auftritt. So ist zum Beispiel oft die Implementierung von mehreren Funktionen notwendig, die sich in ihrer Grundstruktur sehr ähnlich sind. In Abbildung 1 ist dies beispielhaft dargestellt. Es werden drei Funktionen „`max`“ implementiert, jedoch jeweils mit unterschiedlichen Typen („`float`“, „`int`“, „`string`“).

```
01 float max (float a, float b)
02 {
03      return (a<b) ? b : a;
04 }
05
06 int max (int a, int b)
07 {
08      return (a<b) ? b : a;
09 }
06 string max (string a, string b)
07 {
08      return (a<b) ? b : a;
09 }
```

Abbildung 1 Beispiel für Codeduplizierung




```
01 template<typename T>
02 const T & max (const T & a, const T & b)
03 {
```

```
04      return (a<b) ? b : a;
05 }
```

Abbildung 2 Lösung des Problems durch Templates

```
01 float f1 = 1.2, f2 = 2.3;
02 float f3 = max (f1, f2);
03
04 int i1 = 1, i2 = 2;
05 int i3 = max (i1, i2);
06
07 string s1 (“blub”), s2 (“bla”);
08 string & s3 = max( s1, s2);
```

Abbildung 3 Benutzung des Templates

Das in Abbildung 2 dargestellte Template vermittelt einen Eindruck, wie die Funktionalität der Templates im Allgemeinen erreicht wird. Es handelt sich dabei um eine Vorschrift, die eine Familie von überladenen Funktionen erzeugt [BE00]. In Zeile 1 wird das Template deklariert mit einem formalen Templateparameter `typename T.` Dieser Parameter `T` wird im Verlauf des Skriptes immer wieder verwendet und mit dem aktuellen Parameter, der den Wert enthält, gefüllt. Ist `T` beispielsweise zum Zeitpunkt der Instanziierung mit `float` belegt, entsteht die in Abbildung 1 dargestellte erste Funktion. Solche „Schablonen“ ermöglichen es also „analogen“ Code nicht mehrfach wiederholen zu müssen. Ein Compiler generiert nach Bedarf die benötigten Klassen und Funktionen anhand der Schablonen.

2.2 Was ist eine Template Engine?

Eine Template Engine ist ein Softwaremodul, das bei der Generierung von HTML-Seiten für dynamische Web-Seiten benutzt wird. Das Design des Webprojektes ist in Template-Dateien definiert. Auf der anderen Seite stehen PHP-Skripte, die die Funktionalität des Projektes implementieren. Zur Laufzeit veranlasst eine Applikation die Template Engine, das Template und die PHP-Skripte zu laden und dabei Templateparameter mit Werten

zu füllen, Blöcke des Templates wiederholt zu instanziieren und die fertige HTML-Seite zu erzeugen.

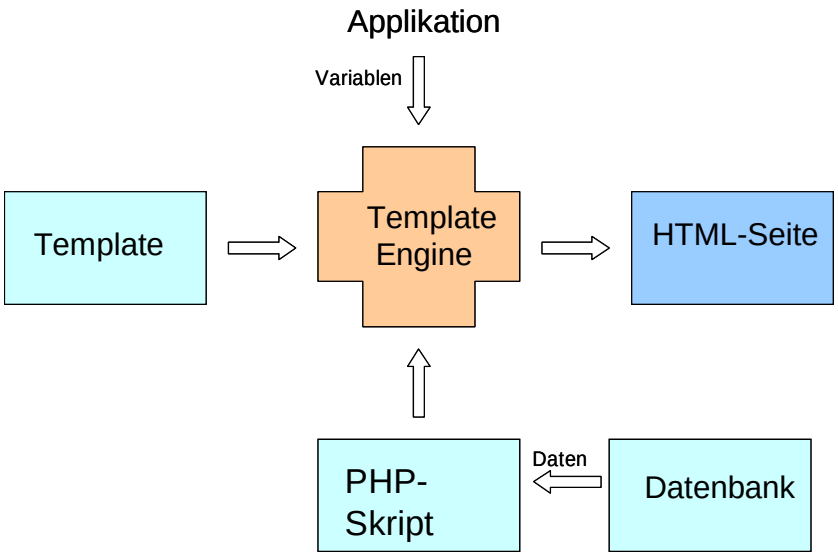


Abbildung 4 Funktionalität von Template Engines

2.3 Warum sollte eine Template Engine benutzt werden?

Der PHP-Code und das Design sind zwei separat zu betrachtende Aspekte einer Webpräsenz. Mit Hilfe einer Template-Engine können sie unabhängig voneinander entwickelt werden, da sie in getrennten Dateien und Formaten vorliegen. Softwareentwickler und Webdesigner können miteinander arbeiten und kooperieren, ohne sich gegenseitig bei der Arbeit zu behindern, da die zu bearbeitenden Bereiche des Projektes nicht in nur einer Datei liegen. So können die Verantwortlichkeiten strikt in Design und Funktionalität geteilt werden. Der Designer kann eine HTML-Datei erstellen und an den Stellen, an denen dynamische Werte verfügbar sein müssen, Variablen einfügen. Für diese Platzhalter kann der Entwickler der PHP-Skripte bei seiner Arbeit Werte einsetzen, die zum Beispiel aus einer Datenbank geladen werden, und kommt mit dem Layout nicht oder kaum

in Kontakt. Nachdem die Entwicklung der Webpräsenz beendet ist, kann das Design später noch gewartet, verändert und weiter entwickelt werden, ohne den Programmcode für die Funktionalität anpassen zu müssen. Auch stellen Templates eine sinnvolle Möglichkeit für Multi-Language-Seiten dar. Hier können mehrere Sprachen verfügbar gemacht werden und je nach Bedarf geladen werden.

2.4 Die Template Engine Smarty

Um die oben genannten Probleme zu lösen wurde das OpenSource-Projekt Smarty, das in Form einer PHP-Klasse vorliegt, entwickelt. Die Besonderheit bei dieser Engine liegt darin, dass nicht, wie sonst üblich, Variablen einfach ersetzt werden, sondern aus einem Template eine neue PHP Datei generiert wird, die dann von Smarty als PHP-Include eingebunden wird. Da dieses so genannte „Kompilieren“ von Templates nicht bei jedem Aufruf ausgeführt wird, sondern nur dann, wenn sich das Template geändert hat, ergeben sich für die Ausführung große Geschwindigkeitsvorteile.

2.4.1 Eigenschaften von Smarty

Smarty unterstützt alle üblichen Ablaufstrukturen, wie beispielsweise beliebig tief geschachtelte Verzweigungen und Schleifen, Datenbankabfragen sowie benutzerdefinierte Funktionen. Die Templates bestehen aus HTML-Code, erweitert um so genannte Template-Tags. Die Notation der Template-Tags ist der von HTML-Tags sehr ähnlich, jedoch wird `{{…}}` als Begrenzung genutzt. Smarty interpretiert alle Inhalte außerhalb der Begrenzungen als statisch. Sobald Template-Tags auftreten, werden diese verarbeitet und die Ergebnisse als Ausgabe an dieser Stelle eingefügt. Die Werte von Variablen werden ausgegeben, indem der Variablenname ausgeschrieben wird, zum Beispiel `{{$nachname}}`.
Verzweigungen nutzen die Schlüsselwörter `if`, `else`, `elseif` und `/if`. Als Bedingungen dienen boolesche Ausdrücke mit den üblichen Vergleichs- und booleschen Operatoren.

Die Schlüsselwörter `section, sectionelse` und `/section` definieren Schleifen. Dabei wird der `sectionelse` Teil genau dann ausgeführt, wenn die Abbruchbedingung sofort greift, also kein Schleifendurchlauf durchzuführen ist. Schleifen iterieren grundsätzlich über die Elemente eines. So benötigen sie einen `name`-Parameter, der der Laufvariablen entspricht, und einen `loop`-Parameter, der das Array referenziert, dessen Werte durchlaufen werden müssen.

Als Beispiel für Funktionen sei der Aufruf `{{include file="header.tpl"}}` gegeben. Hier wird an der Stelle, an der der Aufruf eintritt, der Inhalt eines anderen Template eingefügt, wie bei normalen includes üblich [OZ03].

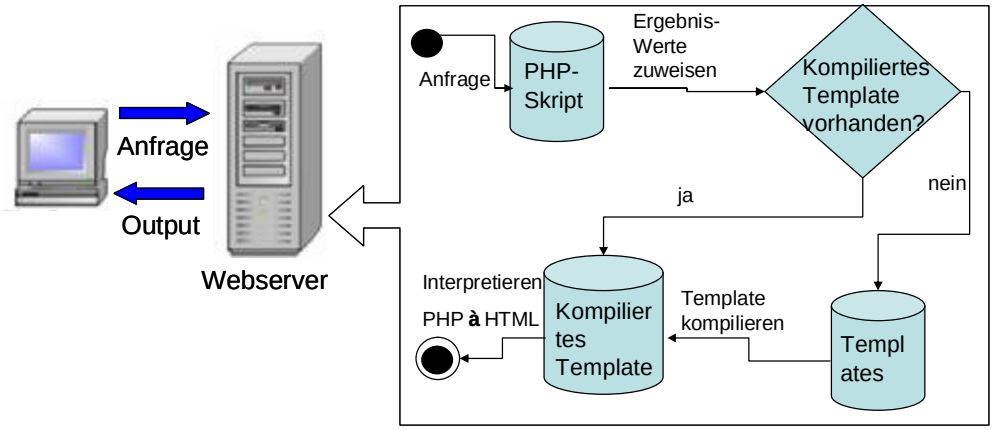


Abbildung 5 Ablauf einer Anfragbearbeitung mit Smarty

2.4.2 Beispiel

Das folgende Beispiel ist Teil der Webapplikation http://ehemalige.ratsgymnasium-bielefeld.de/abi-1999/. Innerhalb dieser werden Informationen zu auswählbaren Personen dargestellt, falls über diese Daten in der Datenbasis vorliegen. Abbildung 6 zeigt einen Auszug des PHP-Skripts, das die Anzeige der Daten steuert. Die Datei ist in drei Abschnitte zu gliedern. Bis Zeile 25 werden die nötigen personenspezifi-

schen Daten mit Hilfe einer Datenbankabfrage beschafft. Im zweiten Abschnitt (Zeile 26-45) werden die Template-Variablen mit Werten belegt. Ab Zeile 46 wird die Anzeige des Templates veranlasst. Am Anfang der Datei wird die Klasse `Ereignis` als Container für die Daten eines Ereignisses definiert. In Zeile 22ff. wird eine Verbindung zur Datenbank hergestellt, aus der dann der Datensatz der vom Benutzer ausgewählten Person abgefragt wird (Zeile 25). Sind in der Datenbasis Daten zu dieser verfügbar, werden ab Zeile 28ff. Smarty-Variablen durch die Funktion „`assign`“ mit dem Ergebnis der Abfrage initialisiert. Einen Spezialfall bildet die Smarty-Variable `ereignisse`: Hierbei handelt es sich um die Variable für ein Array (Zeile 36), das in Zeile 38 mit Werten aus der Datenbank befüllt wird. In Zeile 47 wird schließlich das Template mit dem Befehl „`display`“ angezeigt.

```
01 <?php
02  …
03
04 require (SMARTY_DIR.”Smarty.class.php”);
05
06 class Ereignis
07 {
08 var $beschreibung;
09 var $zeitpunkt;
10
11 function Ereignis ($beschreibung=””, $zeitpunkt=””)
12 {
13 $this->beschreibung=$beschreibung;
14 $this -> zeitpunkt = $zeitpunkt ;
15 }
16 }
17
18 $smarty = new Smarty ;
19
20 …
21
22 $conn = mysql_connect ($host , $user , $pass );
23 mysql_select_db ($db);
24
25 $result = mysql_query (" SELECT nachname , vorname , aktiv ,
          DATE_FORMAT ( geburtsdatum , ’%d.%m.%Y ’) , fotoAbitur ,
          fotoAktuell , anschrift1 , anschrift2 , plz , ort ,
          telefon, telefax , mobil , email , website , icqNummer ,
          aktuelleAktivitaet , UNIX_TIMESTAMP ( letzteAenderung )
          FROM person WHERE id = $id " , $conn );
26  if( $person = mysql_fetch_row ( $result ))
27 {


//Template-Variablen belegen
```

```
28 $smarty -> assign ("nachname" ,htmlentities($person [0]));
29 $smarty -> assign (" vorname " , htmlentities ( $person [1]) );
30
31 $smarty -> assign (" aktiv " , $person [2] == 'Y');
32
33 ...
34
35 $result = mysql_query(" SELECT beschreibung,DATE_FORMAT
          (zeitpunkt , '%d.%m.%Y ') FROM ereignis WHERE
           person=$id" , $conn );
36 $smarty -> assign (" ereignisse " , array ());
37 while ( $ereignis = mysql_fetch_row ( $result ))
38 $smarty -> append (" ereignisse " , new Ereignis (htmlentities
          ( $ereignis [0]) , htmlentities ($ereignis [1]) ));
39
40 $smarty -> assign (" letzteAenderung " , htmlentities (strftime
           ("%d . %B %Y" , $person [17]) ));
41
42 $smarty -> assign ("id" , $id);
43 }
44
45 mysql_close ( $conn );
46

//Template anzeigen

47 $smarty -> display ("abi -1999/ person .tpl");
48 ?>
```

Abbildung 6 Smarty-Beispiel: PHP-Skript

In Abbildung 7 ist das zum oben stehenden PHP-Code gehörige Template dargestellt. Ab Zeile 9ff. bzw. 30ff. wird je nachdem, ob der Personendatensatz aktiviert wurde, was abhängig von der Belegung der booleschen Variable „`aktiv`“ ist, entweder der Datensatz zur Person oder ein Fehlertext angezeigt. Die Tabelle zur Darstellung der Daten wird in den Zeilen 16 – 27 erstellt. In den Zeilen 14ff. und 26ff. wird die Entscheidung bezüglich des Tabellenkopfes oder -fußes getroffen, das heißt, wenn mindestens ein Datensatz vorhanden ist, wird der Kopf angezeigt und nach dem letzten Datensatz wird der Fuß dargestellt.

```
1 <html >
2
3 <head >
4 <meta http - equiv =" content - type " content =" text / html ;
    charset =iso -8859 -1 ">
5 <link rel =" stylesheet " href ="abitur -1999. css ">
6 </head >
```

```
7
8 <body class =" hintergrund ">
9 {{ if $aktiv }}
10
11 ...
12
13 {{ section name = ereignis loop = $ereignisse }}
14 {{ if % ereignis . first %}}
15      <p class =" zentriert ">Besondere Ereignisse : </p>
16      <table border ="1" width =" 100% ">
17 {{/ if }}
18      <tr >
19 {{ if $ereignisse [ ereignis ]-> zeitpunkt eq ""}}
20      <td width ="20% ">& nbsp ; </td >
21 {{ else }}
22      <td width ="20% ">{{ $ereignisse [ ereignis ]->zeitpunkt}}
         </td >
23 {{/ if }}
24      <td width ="80%">{{$ereignisse [ereignis]->beschreibung }}
         </td >
25       </tr >
26 {{ if % ereignis . last %}}
27      </table >
28 {{/ if }}
29 {{/ section }}
30 {{ else }}
31    <p class =" warnung ">Die Daten sind ( noch ) nicht zur Ver
       & ouml ; ffentlichung freigegeben </p>
32 {{/ if }}
33
34 ...
35
36 </body >
37
38 </html >
```

Abbildung 7 Smarty-Beispiel: HTML-Template

2.5 Die Template Engine Vlib

Vlib Template ist, wie Smarty, eine PHP-Klasse, welche ebenfalls die Trennung von PHP- und HTML-Code ermöglichen soll. Es sind intuitiv gewohnte Funktionen verfügbar, die durch „markup tags“, die wie HTML-Tags durch `< … >` und `</ … >` gekennzeichnet sind, benutzt werden können: `<tmpl_var>` für die Benutzung von Variablen, `<tmpl_loop>` zur Initialisierung einer Schleife, `<tmpl_include>`, um Dateien einzubinden, `<tmpl_if>` für Fallunterscheidungen [KJ03].

2.5.1 Beispiel Bedingungskonstrukte

Im Folgenden ist eine Datei form.php angegeben. In ihr ist eine Mischung von HTML und PHP zu erkennen (ab Zeile 7: PHP-Code). Zur Vermeidung dieses Problems, wird in dem folgenden Beispiel die Datei mit Hilfe der Template Engine Vlib realisiert [CB04].

```
01 <html>
02 <head>
03     <title>simple &lt;FORM&gt;</title>
04 </head>
05
06 <body>
07 <?php
08    if (isset($input))
09    {
10        echo "You typed <b>$input</b>.\n";
11    }
12    else
13    {
14        echo <<<form
15            <form action="form.php" method="post">
16                <input name="input">
17                <input type="submit">
18            </form>
19 form;
20    }
21 ?>
22 </body>
23 </html>
```

Abbildung 8 Ausgangsbeispiel Vlib mit Durchmischung von PHP und HTML

```
01 <?php
02     require_once "../vlibTemplate.php";
03
04     $tmpl = new vlibTemplate('tmpl/vlibTemplate_form.htm');
05
06     if (isset($input))
07     {
08             $tmpl->setVar('display', 1);
09             $tmpl->setVar('input', $input);
10     }
11     else
12     {
13             $tmpl->setVar('display', 0);
14     }
15
16     $tmpl->pparse();
17 ?>
```

Abbildung 9 PHP-Datei Template_form.php

In dem PHP-Skript sind drei Abschnitte zu erkennen: Zunächst wird das zugehörige Template eingelesen (Zeile 2-4), ab Zeile 6 werden Werte an Variablen gebunden, In Zeile 16 findet die Instanziierung statt.
In Zeile 1 wird die Bibliothek eingelesen, die die Vlib Engine enthält. Bei der Variablen `$tmpl` handelt es sich um eine neues Instanz der Klasse vlibTemplate. Die Fallunterscheidung überprüft, ob `$input` gesetzt wurde. Wenn ja, wird `$input` ausgeben, ansonsten das Formular (zur Eingabe von `$input`). Zum Abschluss, wird die Funktion `pparse()` ausgerufen, die das Template verarbeitet.

```
01 <html>
02 <head>
03      <title>simple &lt;FORM&gt;</title>
04 </head>
05 <body>
06 
07 <tmpl_if name='display'>
08     You typed <b>{tmpl_var name='input'}</b>.
09 <tmpl_else>
10     <form action="vlibTemplate_form.php" method="post">
11        <input name="input">
12        <input type="submit">
13    </form>
14 </tmpl_if>
15 
16 </body>
17 </html>
```

Abbildung 10 Zugehörige HTML-Datei Template_form.htm

In der oben stehenden Abbildung ist das Template zum PHP-Code dargestellt. Im Template wird eine Fallunterscheidung verwendet. Diese funktioniert folgendermaßen: Wenn `'display'` auf 1 gesetzt wurde, wird "`You typed <b>{tmpl_var name='input'}</b>`." ausgegeben, was eine Ausgabe der eingegebenen Benutzdaten darstellt, ansonsten wird durch `<tmpl_else>` das Formular mit Eingabefeld und Button angezeigt. `$tmpl->setVar('input', $input)` bedeutet, dass wenn `$input` gesetzt ist, muss `$input` dem Template durch die Funktion `setVar` auch noch übergeben werden.

2.5.2 Beispiel Schleifen

```
01 <?php
02   require_once "../vlibTemplate.php";
03
04   $tmpl = new vlibTemplate('tmpl/vlibTemplate_loops.htm');
05
06   $tmpl->setVar('title', 'This is the vlibTemplate Loops
     example...');
07
08   $people = array(
09      'David' => array(
10         'age' => 32,
11         'sex' => 'Male',
12         'height' => 175,
13         'hobby' => 'football',
14       'children' => 2),
15    'Bob' => array(
16         'age' => 65,
17         'sex' => 'Male',
18         'height' => 150,
19         'hobby' => 'rugby',
20         'children' => 0),
21      'Gloria' => array(
22         'age' => 28,
23         'sex' => 'Female',
24         'height' => 160,
25         'hobbies' => 'swimming, TV',
26         'children' => 1)
27   );
28
29
30   $peoplearr_basic = array();
31   // we need to rebuild the array into the correct format
32   foreach ($people as $name => $details)
33   {
34      array_push($peoplearr_basic,
35         array(
36            'name' => $name,
37            'age' => $details['age'],
38            'sex' => $details['sex'],
39            'height' => $details['height']
40         )
41      );
42   }
43   $tmpl->setLoop('basic_loop', $peoplearr_basic);
44   …
```

Abbildung 11 PHP-Skript zum Template mit Schleifen

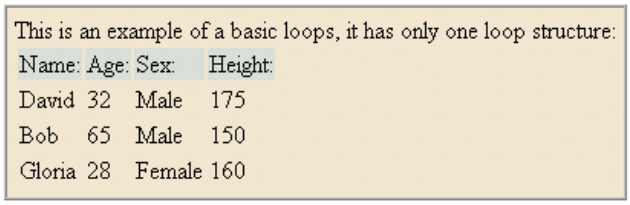
In Abbildung 11 wird gezeigt, wie Schleifen in Vlib realisiert werden [CB04]. Als Datenquelle für die Schleife im Template dient ein zweidimensionales Array. Die erste Dimension, die numerisch indiziert ist, wird von der Schleife durchlaufen. Die zweite Dimension enthält die Werte zu Di-

mension eins. Mit `$peoplearr_basic = array()` wird das Array `$peoplearr_basic` erzeugt. Durch `foreach ($people as $name => $details)` wird das vorher initialisierte Array über eine Schleife eingelesen.
`array_push($peoplearr_basic, array(...` fügt dem Array `$peoplearr_basic` ein Element hinzu. Da es sich bei diesem Element wieder um einen Array handelt, wird `$peoplearr_basic` zu dem zweidimensionalen Array. Mit der Funktion "`setLoop`" wird das Array an die Template-Datei übergeben.

```
01 <html>
02 <head>
03     <title>{tmpl_var name='title'}</title>
04 </head>
05
06 <body>
07 <table>
08    <tr bgcolor="#dddddd">
09        <td>Name:</td>
10        <td>Age:</td>
11        <td>Sex:</td>
12        <td>Height:</td>
13    </tr>
14 <tmpl_loop name="basic_loop">
15    <tr>
16        <td>{tmpl_var name='name'}</td>
17        <td>{tmpl_var name='age'}</td>
18        <td>{tmpl_var name='sex'}</td>
19        <td>{tmpl_var name='height'}</td>
20    </tr>
21 </tmpl_loop>
22 </table>
```

Abbildung 12 vlib-Template mit Schleifen

`<tmpl_loop name="basic_loop">` in dem Template zeigt den Beginn der Schleife basic_loop. Die zu belegenden Variablen werden im Template deklariert und über die Schleifenelemente dann mit Werten belegt. `{tmpl_var name='name'}` stellt einen Schlüssel zur zweiten Dimension des Arrays dar und setzt einzelne Datenelemente im Schleifenrumpf ein. Durch `</tmpl_loop>` wird die Schleife beendet.


This is an example of a basic loops, it has only one loop structure:

Name:	Age:	Sex:	Height:
David	32	Male	175
Bob	65	Male	150
Gloria	28	Female	160

Abbildung 13 Ergebnis der PHP- und HTML-Dateien

3 Formularerzeugung

3.1 HTML_QuickForm

Für professionelle Webprojekte ist die Verwendung von Formularen zur Eingabe von Daten Standard geworden. Das »PEAR::HTML_QuickForm«-Paket ermöglicht und erleichtert die Erstellung von HTML-Formularen, wobei ebenfalls Schnittstellen zu Template Engines wie zum Beispiel Smarty zur Verfügung gestellt werden [JM04]. Die Besonderheit bei diesem Projekt ist, dass die Definition der HTML-Elemente in einer PHP-Datei stattfindet und aus dieser Datei später HTML-Code generiert wird.

3.2 Beispiel

In Abbildung 14 ist der Code zur Erstellung eines Formulars dargestellt [RS03]. Es bietet Eingabemöglichkeiten für Benutzerdaten wie zum Beispiel den Namen, Vornamen und die E-Mail-Adresse. Nach Absenden des Formulars muss eine Validierung der Eingabe erfolgen, um nicht korrekte Daten, wie eine falsch aufgebaute Mail-Adresse, zu verhindern. Die Einleitung des PHP-Skriptes durch „`require_once 'HTML/QuickForm.php'`“ in Zeile 2 importiert das notwendige Pear-Package, mit dem die Funktionen für die Generierung verfügbar werden.
Um ein Formular zu erstellen ist es notwendig, dass zunächst die benötigten Formularelemente mit ihren Eigenschaften eingebunden werden (Zeile 8-27). Ab Zeile 28 werden Validierungsregeln für die Elemente festgelegt [TW04]. Ab Zeile 40 wird die Funktionalität des Formulars vor und nach Betätigen des Submit-Buttons implementiert.
In Zeile 8 ist der Konstruktor des Formulars zu finden. `EmailFormular` ist der Name des zu erzeugenden Formulars und erscheint später im HTML-Code. Der Parameter `POST` sorgt dafür, dass die Daten per HTTP POST gesendet werden. Weitere Parameter können ebenfalls angegeben werden, die sich zum Beispiel auf das Zieldokument beziehen. Durch `ad-`

dElement wird dem Formular eine Kopfzeile hinzugefügt (Zeile 9). Danach werden drei Formularfelder vom Typ »text« hinzugefügt, die als HTML-Textboxen dargestellt werden. Der zweite Parameter bestimmt den Namen des Elements, der dritte die Beschriftung. In Zeile 14 wird ein Submit-Button dem Formular hinzugefügt. Möglich sind zudem alle bekannten anderen Formularbestandteile, wie Checkboxen, Bilder, unsichtbare Textfelder, Passwortfelder, Radiobuttons, Dropdownboxen etc. Für die Elemente können zusätzliche Eigenschaften festgelegt werden, zum Beispiel um für die Eingabefelder eine Länge festzulegen, wird das Element zunächst aufgerufen mit getElement, danach kann die Länge der Felder zugewiesen werden. Ab Zeile 28 wird die Feldvalidierung hinzugefügt. Durch die Methode addRule() können jedem Element beliebig viele Regeln zugewiesen werden. In den Zeilen 29 bis 34 werden jedem Formularelement ein Fehlermeldungstext und zusätzlich der Name der zu prüfenden Bedingung zugewiesen. Die Validierungsbedingung ist vordefiniert, so signalisiert die Verwendung des Schlüsselwortes required, dass es sich um ein Pflichtfeld handelt.

Die if- Anweisung prüft nach Eingabe der Daten durch den Benutzer mit validate(), ob Fehleingaben vorliegen. Ist dies nicht der Fall, das heißt, alle Daten wurden korrekt eingegeben, gibt validate() den Wert TRUE zurück und Formular wird eingefroren. Die freeze()-Funktionsorgt dafür, dass statt der editierbaren Eingabefelder lediglich deren Inhalt ohne die Möglichkeit zur Änderung angezeigt wird. Der Aufruf der Funktion display() am Ende der Datei erzeugt HTML-Code für die Formularelemente und gibt das Formular sichtbar aus.

```
01 <?php
02 require_once 'HTML/QuickForm.php';
03
04 // QuickForm Version anzeigen
05 print 'PEAR::HTML_QuickForm Version ';
06 print HTML_QuickForm::apiVersion() . '<br/><br/>';
07
08 $myForm = new HTML_QuickForm('EmailFormular', 'POST');
09 $myForm->addElement('header', '', 'Persönliche Angaben');
10
11 $myForm->addElement('text', 'textName', 'Name:');
```

```
12 $myForm->addElement('text', 'textVorname', 'Vorname:');
13 $myForm->addElement('text', 'textEmail', 'E-Mail:');
14 $myForm->addElement('submit', 'submitButton','Daten senden');
15
16 $name =& $myForm->getElement('textName');
17 $name->setMaxLength(30);
18 $name->setSize(30);
19
20 $vname =& $myForm->getElement('textVorname');
21 $vname->setMaxLength(20);
22 $vname->setSize(30);
23
24 $email =& $myForm->getElement('textEmail');
25 $email->setMaxLength(50);
26 $email->setSize(30);
27
28 // Validierungsregeln hinzufuegen
29 $myForm->addRule('textName', 'Bitte Name angeben', 'required');
30 $myForm->addRule('textVorname','Bitte Vorname angeben',
   'required');
31 $myForm->addRule('textEmail', 'Bitte E-Mail angeben',
   'required');
32 $myForm->addRule('textEmail', 'Bitte E-Mail ungültig','email');
33 $myForm->addRule('textEmail2', 'Bitte E-Mail angeben', 're-
quired');
34 $myForm->addRule('textEmail2', 'Bitte E-Mail ungültig',
   'email');
35
36 // Auch moeglich, clientseitige Validierung mit JavaScript
37 // $myForm->addRule('textEmail', 'E-Mail ungueltig', 'email',
   NULL, 'client');
38
39 // Falls Validierung OK, Formular einfrieren
40 if ( $myForm->validate() )
41 {
42 print 'Vielen Dank! Ihre Angaben sehen wie folgt aus:';
43 $myForm->removeElement('submitButton');
44 $myForm->freeze();
45 }
46
47
48 // Formular anzeigen
49 $myForm->display();
50
51 ?>
```

Abbildung 14 Code zur Erstellung eines Formulares mit HTML_QuickForm

An dem Beispiel wird deutlich, dass die Formularerzeugung mit HTML_QuickForm relativ einfach ist. Formularaufbau und –verhalten wird in PHP direkt auf Problemebene programmiert. HTML-Code als Mittel zur technischen Umsetzung und Darstellung ist für den Programmierer nicht

sichtbar, wodurch die angesprochene Vermischung von PHP und HTML verhindert wird.

4 Fazit

Im Rahmen der Seminararbeit wurde zunächst das Problem der Mischung von HTML-und PHP-Code und der Codeduplizierung angesprochen. Nachfolgend wurde auf die Vorteile eingegangen, die sich durch die strikte Trennung von Design und Funktionalität ergeben. Die Template Engines bieten eine gute und sinnvolle Möglichkeit, dieses Vorhaben zu realisieren. Sie sind sehr funktional ausgerichtet und schränken die Entwickler nicht ein, da alle gängigen Programmierkonzepte realisiert und eingeplant wurden. Beide Template Engines spezifizieren die Templates in Form von mit zusätzlichen Tags angereicherten HTML-Code und generieren daraus, für den Entwickler sichtbar, ausführbaren PHP-Code.

Auch das Projekt HTML_QuickForm widmet sich dieser Thematik. Hierbei spezifiziert der Programmierer Formulare in Form von direkt ausführbarem PHP-Code. Die Abstraktion ist in den Methoden einer PHP-Klasse verborgen, die die angebotenen Konzepte auf Problemebene direkt technisch umsetzen. Das Projekt bietet somit eine gute Möglichkeit, vorgefertigte Funktionen zur Generierung von HTML-Code zu nutzen. Da Schnittstellen zu Template Engines vorhanden sind, ist es optimal, diese beiden Konzepte kombiniert zu verwenden, um dynamische Webprojekte besser zu entwickeln und später sinnvoll warten zu können.

Für die Arbeit in der Projektgruppe ist es sinnvoll, aus den Konzepten der Template Engines und HTML_QuickForm eine Technik zur Implementierung abzuleiten. So wurde deutlich, dass zur Arbeitsteilung in der Gruppe die Trennung der Dateien in Design und Funktionalität und der generellen Aufteilung des gesamten Projektes in Teilgebiete von Vorteil ist. Ein weiterer Punkt, der bei der Entwicklung des Projektes in nähere Betrachtung rücken sollte, ist der Aspekt des temporären Kompilierens zur Laufzeitreduzierung.

Literaturverzeichnis

[BE00] Bruce Eckel, „Thinking in C++,Volume 1, 2nd Edition, Introduction to Standard C++“, 2000, Prentice Hall, New Jersey

[CB04] Claus van Beek, “Die Template-Klasse „vlib Template“”, 2004, Active Fish, http://lamp.clausvb.de/table_of_content.html

[JM04] Martin Jansen, Alexander Merz, „PEAR Manual“, 2004, The PEAR Documentation Group, http://pear.php.net/manual/de/package.html.html-quickform.tutorial.php

[KJ03] Kelvin Jones, “vlib Template documentation “, 2003, Active Fish, http://vlib.activefish.com/docs/vlibTemplate.html

[OZ03] Monte Ohrt, Andrei Zmievski, “Smarty – die selbstkompilierende Template-Engine“, 2003, Version 2.5, ispi of Lincoln, http://smarty.php.net/download-docs.php

[RS03] Richard Samar, “Bis zur Bahre: Formulare”, erschienen in: „Linux-Magazin“, Ausgabe 09.2003, http://www.linux-magazin.de/Artikel/ausgabe/2003/09/php/php.html?print=y

[TW04] Thomas Witkowski, “Formulare für alle Fälle”, erschienen in: „PHP Magazin“, Ausgabe 02.2004, http://phpmag.de/itr/online_artikel/show.php3?nodeid=62&id=491