Template Engines für PHP

Seminararbeit im Rahmen der Vorlesung

Web-Engineering

Studienbereich Wirtschaft

im Studiengang Wirtschaftsinformatik

an der Berufsakademie

Ravensburg

Verfasser: Cornelius Knall

Kurs:WI2005-2

Datum: 06.09.2007

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Ziele und Motivation

Ziel dieser Seminararbeit ist dem Leser einen Überblick über die Technologie von Template Engines für PHP zu vermitteln. Es soll ebenfalls erklärt werden was Templates sind, und wie die Engines funktionieren. Ziel dieser Arbeit ist es nicht dem Leser beizubringen, wie eine Template Engine programmiert wird. Viel mehr soll eine Hilfestellung gegeben werden, falls der Leser in die Situation kommt, eine Template Engine ein zusetzten. Hier soll ihm die Entscheidung erleichtert werden, ob es Sinn macht, das Rad neu zu erfinden, oder ob eine fertige Software ein gesetzt werden soll.

Bei der praktischen Arbeit an größeren Projekten, wird schnell klar, dass die Vermischung von HTML-Code und PHP-Code die Wartung und Weiterentwicklung erschwert.

Klassisch werden zur Trennung von Code und Design, Templatesysteme eingesetzt. Dabei werden der HTML-Code und der PHP-Code, mit seinen Anweisungen, in separate Dateien gelegt. Durch spezielle Befehle, werden nun die beiden Dateien zur Laufzeit miteinander verknüpft, und an der Browser zur Ausgabe gesendet. Abbildung 1 zeigt einen Code Auszug, ohne Trennung von HTML-Code und PHP-Code.

```
<HTML>
<TAG></TAG><TAG></TAG>
<TAG></TAG><TAG></TAG>
PHP();PHP()PHP();PHP()
PHP();PHP()PHP();PHP()
PHP();PHP()<TAG>;PHP()
PHP()</TAG>PHP();PHP()
<TAG></TAG><TAG></TAG>
<TAG></TAG><TAG></TAG>
PHP();PHP()<TAG></TAG>
<TAG></TAG>PHP();PHP()
PHP()</TAG>PHP()</TAG>
<TAG>;PHP()<TAG>;PHP()
PHP()</TAG>PHP()</TAG>
<TAG></TAG><TAG></TAG>
<TAG></TAG><TAG></TAG>
</HTML>
```

Abbildung 1: Ohne Template Engine

Wie hier gut zu erkennen ist, ist der Code sehr unübersichtlich und daher ist die Weiterentwicklung und die Wartung, wie schon erwähnt, nur mit großem Aufwand zu erledigen. Daher ist es sinnvoll, und in der Praxis auch üblich, den PHP-Code von dem Design zu trennen. Abbildung 2 zeigt einen Code Auszug, wobei der Code und das Design getrennt wurden.

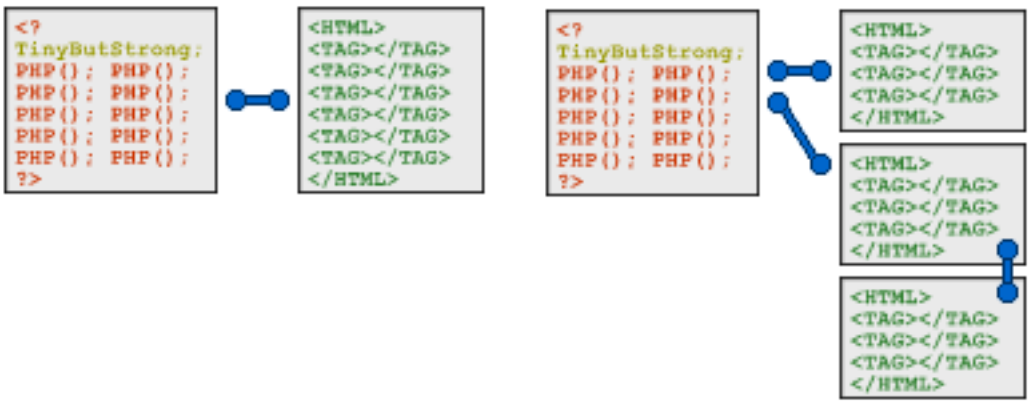

Abbildung 2: Mit Template Engine

Im Rahmen dieser Seminararbeit werden folgende Template Engines vorgestellt:

- Smarty
- EasyTemplateSystem
- Kajona
- Template Engine nur mit PHP
- Beyond The Template Engine

2 Templates und Template-Engines

2.1 Templates

Viele dynamischen Webseiten arbeiteten früher sehr ineffizient. Da die auszugebenden Daten alle in einem Skript vereint waren. Dies bedeutete, dass der Inhalt des

HTML-Dokumentes im Skript, durch simple *echo* respektive *print* Anweisungen an den Browser, zur Ausgabe weiter gegeben wurde. Um die bekannten Probleme, wie die Wartbarkeit und die Weiterentwicklung zu beheben, wurden die so genannten Templates eingeführt.

Templates werden in verschiedenen Situationen eingesetzt. Dokumentenvorlagen *(engl. templates)* werden in Textverarbeitungsprogrammen sehr häufig verwendet. Diese Templates verwenden immer die gleiche Vorlage, nur der Inhalt wird über so genannte Platzhalter dynamisch eingebunden. Daher liegt es nahe, dieses Prinzip auch auf Webseiten anzuwenden. Dies bedeutet, dass nur einmal eine HTML-Datei, mit dem Design, angelegt werden muss. Auch hier wird der Inhalt über Platzhalter dynamisch eingebunden und an den Browser zur Ausgabe gesendet.

2.2 Template-Engines

Eine Template Engine ist ein Software Modul, dass eingesetzt wird um HTML-Seiten oder andere Texte zu generieren. Das Layout einer HTML-Seite wird in einer Templatedatei fest gelegt, die wiederum durch einfache *WhatYouSeeIsWhatYouGet*-Editoren (z.B. Dreamweaver, FrontPage) erstellt, bzw. designed werden können. Zur Laufzeit, wird der Template Engine gesagt, was zu tun ist. Es muss das entsprechende Template geladen werden und die darin enthaltenen Platzhalter müssen mit Inhalt gefüllt werden.

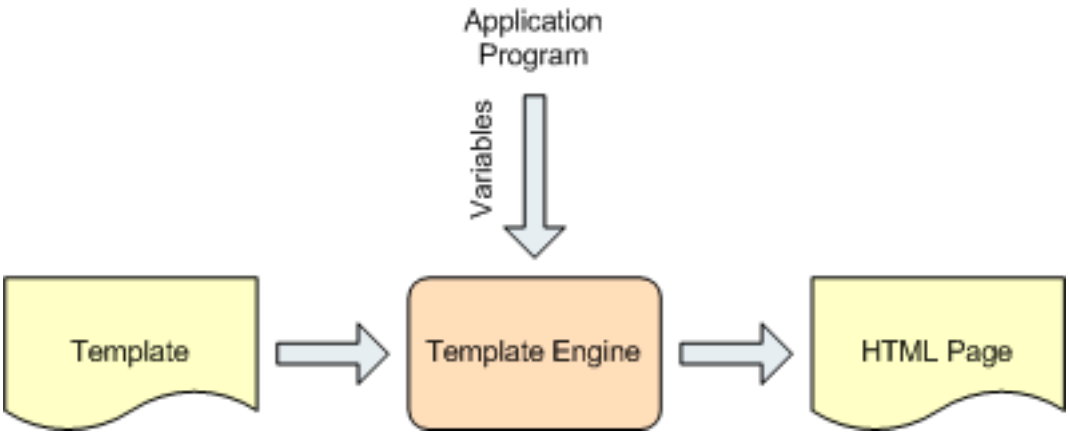

Abbildung 3: Template Engine Diagramm

Generell gibt es zwei Techniken, wie Template Engines arbeiten. Das eine Prinzip

forciert den Aufbau der Seite und die Generierung des HTML-Codes aus einem Skript heraus.
Listing 1 zeigt ein Code-Fragment, wie es die Template Engine FastTemplate verwendet.

Listing 1: FastTemplate PHP-Code

```
1 <?
2 $tpl = newFastTemplate("pfad");
3 $tpl->assign(PAGETITLE, "Musterseite");
4 $tpl->parese(MAIN, "bask");
5 $tpl->FastPrint(MAIN);
6 ?>
```

Der HTML-Code wird in eine Datei *bask.tpl* abgelegt, und könnte folgendermassen aussehen.

Listing 2: FastTemplate HTML-Code

```
1 <html>
2   <head>
3     <title>{PAGETITLE}</title>
4   </head>
5   <body>
6     <h1>Willkommen</h1>
7   </body>
8 </html>
```

Der eigentliche Code, kann natürlich auch noch so erweitert werden, dass die Variable PAGETITEL dynamisch veränderbar ist. Ruft ein Benutzer diese Seite auf, so startet FastTemple und der Code wird abgearbeitet und die Variablen werden ersetzt. Die fertige Seite wird dann direkt an den Browser weiter gegeben.

Smarty hingegen, geht noch einen Schritt weiter und generiert, teilweise ganze HTML-Codeschnippsel. Abgesehen von einigen Zusatzfunktionen, weicht Smarty kaum von dem System, wie es FastTemplate verwenden, ab.

Das andere Prinzip verlegt die Kontrolle über den Aufbau in die HTML-Vorlage. Hier rückt der PHP-Code in den Hintergrund, jedoch muss an den Stellen, an denen es nicht anders möglich ist, PHP-Code in die HTML-Vorlage geschrieben werden. Das obige Beispiel für FastTemplate könnte dann wie folgt aussehen:

Listing 3: phpTemple Code

```
<SET VARIABLE="PAGETITLE" VALUE="Musterseite" />
<html>
  <head>
    <title>{PAGETITLE}</title>
  </head>
  <body>
    <h1>Willkommen</h1>
  </body>
</html>
```

Extra Programmiercode ist an dieser Stelle nicht mehr nötig. Neu hinzugekommen ist der Tag »SET«, dass der Variablen *PAGETITLE* einen Wert zuweist. Natürlich müssen hier Datenbankabfragen, Schleifen und Verzweigungen verfügbar gemacht werden.

2.3 Vorteile

Vorteilhaft ist die einfache Trennung der Aufgabengebiete, so können Designer und Entwickler unahbhängig von einander an dem selben Projekt arbeiten. Da der Designer nicht mit PHP-Code konfrontiert wird. Ebenfalls muss sich der Entwickler nicht mit Design Problemen beschäftigen.
Wie schon erwähnt, ist die Wartbarkeit und die Weiterentwicklung komfortabler, wenn das Design vom Anwendungscode getrennt wird.

2.4 Nachteile

Ein Nachteil der Template Engines ist, es benötigt eine gewisse Einarbeitungszeit. Man muss sich auf eine Technologie festlegen, denn ein Wechsel zu einem späteren Zeitpunkt ist, wenn überhaupt, nur mit sehr großem Aufwand machbar.

3 Vostellung verschiedener Template Engines

3.1 Smarty

Smarty ist wie schon erwähnt eine Template Engine. Die Spezifikation dieser Engine begann 1999, und wurde anfangs in C geschrieben, in der Hoffnung das sie in PHP integriert wird. Nach einiger Entwicklungszeit wurde festgestellt, dass technische Probleme entstanden. Daher wurde entschieden, die Engine komplett in PHP Klassen zu realisieren, mit dem Hintergedanken, dass sie von jedermann eingesetzt und angepasst werden konnte. So entstand schließlich die Engine *SmartTemplate*. Diese Engine wurde jedoch nicht veröffentlicht. Durch Weiterentwicklung dieses Ansatzes, entstand schließlich die SmartyEngine.
Smarty bietet viele komfortable, flexible Möglichkeiten, um Templates zu erstellen, und die Applikationslogik vom Layout zu trennen[1].

Bevor nun los legen werden kann, muss die Smarty Engine installiert werden. Unter http://smarty.php.net kann die entsprechende Version geladen werden.

Smarty bringt eine ganze Menge Dateien mit. Die nicht alle gebraucht werden. Hier werden nun die Dateien vorgestellt, welche für einen effektiven Einsatz von Smarty nötig sind, beschrieben. Die wichtigsten Dateien befinden sich im Unterordner */libs*. Zu beachten ist, dass der Webserver auf diesen Ordner Leserechte hat. Um alle Features von Samrty effektiv zu nutzen, sind noch vier weitere Verzeichnisse von Bedeutung:

- templates
- configs
- templates_c
- cache

Die Namen der Verzeichnisse sind nicht zwingend, diese können je nach Bedarf umbenannt werden. Jedoch sind sie standardmäßig sehr Aussage kräftig. Diese Verzeichnisse sollten nun in einen speziellen Ordner unterhalb der Ebene, der eigentlichen Webseite gelegt werden. So sind diese Dateien sauber aufgeräumt und schnell auffindbar.

Um den Überblick nicht zu verlieren, werden die Templates in dem Unterverzeichnis */templates* abgelegt. Wichtig ist, dass der Webserver auf diesen Ordner *Lese-* und *Schreibrechte* hat.

Ein weiterer Ordner,auf den der Webserver *Lese-* und *Schreibrechte* benötigt ist das Verzeichnis */configs*. Hier werden zentrale Konfigurationsdateien für die Webseite hinterlegt.

In dem Ordner */templates_ c* werden die kompilierten Templates von Smarty abgelegt. Das »c« bei templates_c steht für compiled. Daraus resultiert, dass Smarty eine kompilierende Template Engines ist. Dies bedeutet, dass im Gegensatz zu anderen Template Engines, Smarty die eigenen Templates, welche in einer eigenen Syntax geschrieben sind, nach dem PHP-Code übersetzt. Der Vorteil dieser Vorgehensweise ist, dass nicht bei jedem Aufruf, das ganze Template analysiert und interpretiert werden muss.

Das letzte Verzeichnis */cache* ist wohl das interessanteste. Dieses Verzeichnis wird benötigt, wenn die Funktionalität des »cachings« eingesetzt wird. Smarty ist in der Lage, einmal statisch generierte Seiten zu »cachen«. Dies bedeutet, dass Smarty eine angeforderte Seite erstellt, und diese dann als fertige HTML-Seite abspeichert. Wird diese Seite nun wieder angefortert, muss diese Seite nicht noch einmal generiert werden, sondern kann direkt an den Browser gesendet werden. Dazu aber später mehr.

3.1.1 Erste Schitte

Für die Beispiele wird davon ausgegangen, dass die PHP-Seiten in folgendem Verzeichnis liegen:

```
/var/www/cms
```

Die anderen angesprochenen Verzeichnisse liegen alle unterhalb von */cms/smarty*. Um mit Smarty arbeiten zu können, werden die absoluten Pfadangaben der Verzeichnisse benötigt.

Um eine Seite auf Basis von Smarty zu erstellen, muss zuerst die Klasse *Smarty.class.php*, die sich im Unterverzeichnis */libs* befindet, eingebunden werden.

Da es umständlich ist, für jede Seite die erstellt wird, die Pfadangaben einzubauen, wird eine eigene Klasse generiert, die wie folgt aussehen könnte:

Listing 4: Einbinden der Smarty-Engine

```
1  <?
2  // Konstante für die Pfadangabe
3  define('PFAD', '/var/www/cms/smarty');
4
5  // Einbinden der orginalen Smarty.class.php
6  require (PFAD.'/libs/Smarty.class.php');
7
8  // Deklaration der neuen Klasse
9  class MySmarty extends Smarty
10  {
11    // Der neue Konstruktor
12    function MySmarty ()
13     {
14     // Konstruktor
15     $this->Smarty();
16
17     // Definition der Unterverzeichnisse
18     $this->template_dir = PFAD.'templates/';
19     $this->config_dir = PFAD.'configs/';
20     $this->compile_dir = PFAD.'template_c/';
21     $this->cache_dir = PFAD.'cache/';
22     }
23  }
24 ?>
```

Die Deklaration der eigenen neuen Klasse, sollte einfach in das Unterverzeichnis */cms/smarty* gespeichert werden. Nun kann die Frage gestellt werden, warum die neue Klasse nicht in das */libs* Verzeichnis gespeichert werden kann. Es ist durch aus möglich dies zu tun, kann aber bei einem Update der Smarty-Engine sehr schnell zu

Verlusten oder zu anderen Komplikationen kommen.

Da nun die Engine eingebunden wurde, kann begonnen werden mit der Engine zu arbeiten. Das nächste Beispiel soll zeigen, wie die Daten von der PHP-Seite an das Template übergeben werden.
Smarty kennt hierzu die Methode `assign`.

Listing 5: PHP-Code für Smarty

```
<?
  require 'smarty/MySmarty.class.php';

  //Instanziieren eines neuen Objektes
  $smarty = new MySmarty;

  // Initialisierung der Variablen
  $smarty->assign ('titel', 'Hallo Welt!');
  $smarty->assign ('text', 'Die erste Seite');

  // Template übergeben
  $smarty->display('eins.tpl')
?>
```

Die Methode *display* dient dazu, ein Template zu laden. Dieses Template ist eine Datei die hier *eins.tpl* heisst, und folgenden Code beinhaltet.

Listing 6: Code für das Template

```
<html>
  <head>
    <title>{$titel}</title>
  </head>
  <body>
    <b>{$text}</b>
  </body>
</html>
```

Die Smarty-Befehle innerhalb des Templates werden mit geschweiften Klammern { und } eingeschlossen. Um den Wert der Variablen ausgeben zu lassen, wird der Variablenname, eingeleitet durch ein Dollar-Zeichen »$« in geschweiften Klammer geschrieben.
Sobald die Seite im Browser auf gerufen wird, kompiliert Smarty das Template und wird ausgegeben. Auch wird nun das Template im Ordner /templates_c gespeichert.

Die Methode *assign* beherrscht nicht nur die Übergabe von Stingliteralen, sondern kann auch Variablen, Arrays und sogar Objekte behandeln.

3.1.2 Modifikatoren

Smarty unterstützt den Entwickler bei der Ausgabe und Formatierung durch sogenannte »Modifikatoren«. Um die Ausgabe eines Wertes zu steuern, wird er mit einer Pipe | an den Modifikator weitergeleitet. Die Anweisung

```
{$name|upper|spacify}
```

übergibt an den Modifikator den gesamten Text und wandelt diesen einmal in Großbuchsten um, und gibt diesen dann noch gesperrt aus. Aus »Smarty« würde dann »S M A R T Y« werden. Wie hier zu sehen ist, können Modifikatoren auch mit einander kombiniert werden. Leider ist es im Rahmen dieser Arbeit nicht möglich alle Modifikatoren zu nennen. Unter *http://smarty.php.net/docs.php* ist die komplette Liste aller Modifikatoren zu finden.

3.1.3 Funktionen

Neben den schon besprochenen Modifikatoren gibt es in Smarty auch eine ganze Menge an Funktionen. Der Unterschied zwischen Modifikatoren und Funktionen besteht darin, dass Modifikatoren die Formatierung eines Variableninhaltes beeinflussen. Funktionen hingegen beziehen sich nicht direkt auf den Inhalt einer Variablen oder haben einen eigenen Rückgabe Wert. Des weiteren sind hier auch Kontrollstrukturen wie Schleifen und `if`-Abfragen enthalten. Wie bei der Ausgabe von Variablen, werden auch die Funktionen in geschweiften Klammer geschrieben. Jedoch gibt es hier leider etwas Problem, sobald z.B. JavaSkript im Template notwendig wird. Da JavaSkript die Anweisungsblöcke auch in geschweiften Klammer setzt, musste eine

andere Lösung gefunden werden. Um nun JavaSkript problemlos ein zubinden, muss der »fremde Code« mit hilfe der Funktion `{literal}{/literal}` eingebettet werden. Diese Funktion weist Smarty an, den eingeschlossenen Code zu ignorieren.

Eine Auflistung aller Funktionen die in Smarty bereits implementiert sind, würde den Rahmen dieser Arbeit sprengen. Unter *http://smarty.php.net/docs.php* gibt es die komplette Liste zum nachlesen. Im Rahmen dieser Arbeit werden nur ein paar wichtige Funktionen behandelt.

```
if, else, elseif
```

Die `if`-Anweisung ermöglicht eine Fallunterscheidung. Es werden die selben Möglichkeiten wie in PHP angeboten. Für jedes `{if}` muss ein entsprechendes End-tag `{/if}` angegeben werden.

```
section, foreach
```

Smarty stellt zwei Funktionen für Behandlung von Arrays zur Verfügung. Das ist zum einen die `{section}{/section}`, die normalerweise für indizierte array gedacht ist, und zum anderen die `{foreach}{/foreach}`-Funktion für assoziative Arrays. Die Funktion `{section}` ist wie folgt aufgebaut:

```
{section name=SectName loop= $name step=2 max=3}
```

Der Parameter `name`, ist der Name der Section, der zweit Parameter `loop`, ist das zu durchlaufende Array. Mit den optinalen Parametern `step` und `max` können zum einen die Schrittweite als auch die maximale Anzahl der Iterationen definiert werden.

```
{foreach from=$Id item=aktuelle_id key=key name=name}
```

Der pflicht Parameter `from` bezeichnet das zu durchlaufende Array. Der zweite pflicht Parameter `item` bezeichnet das aktuelle Element. Die beiden optinalen Parameter `name` und `key`, behandeln zum einen den Namen der Schleife, wie bei der Funktion `section`, und zum anderen den aktuellen Schlüssel.

Bei der Programmierung, kann es immer wieder zu Phänomenen kommen, die so nicht gewünscht sind. Um solchen Fehlern auf die Schliche zu kommen, bietet Smarty die Funktion `{debug}` an. Diese Funktion öffnet ein Browserfenster, in dem nun alle relevanten Varieblen etc. mit deren Inhalt ausgegeben wird.

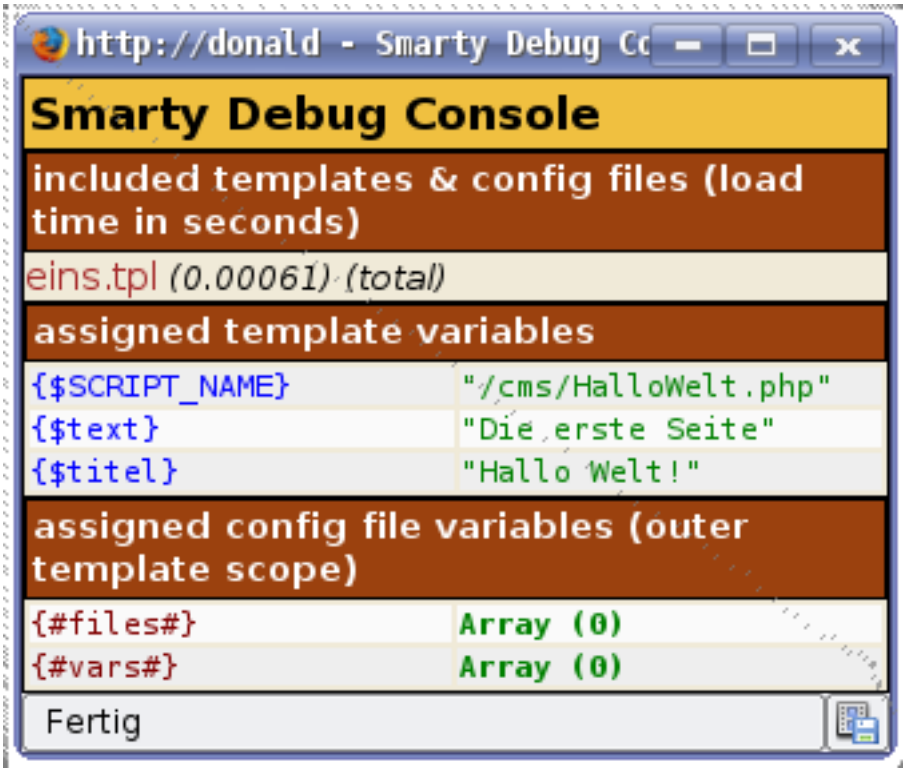

Abbildung 4: Debug_Console

3.1.4 Caching

Einer der Funktionalitäten die Smarty so interessant machen, ist die Möglichkeit Dateien zu cachen. Dies bedeutet, dass Smarty in der Lage ist eine statische Kopie einer Seite zu erstellen, die einmal aufgerufen wurde. Beim nächsten Aufruf, kann der der Server dann diese Kopie verwenden und diese dann direkt an den Browser senden. Ohne jedesmal den PHP-Code und den Smarty-Code neu zu generieren. Diese Funktion spart Systemresourcen ein. Caching kann die Performance vor allem dann deutlich verbessern, wenn Templates längere Rechenzeit beanspruchen[5]. Nun stellt sich sich die Frage, dass durch diese Funktionalität, der Inhalt nicht mehr dynamisch ist, da Änderungen nicht sofort sichtbar sind. Diese Frage ist nur bedingt

berechtigt, hier gibt es die Möglichkeit, die »Cachingdauer« frei zu definieren.

Listing 7: Caching

```
1  <?
2    require_once 'smarty/MySmarty.class.php';
3    $smarty = new MySmarty;
4    $smarty->caching=true;              // Caching einschalten
5    $smarty->caching_lifetime=600;// Spaicherzeit 600 Sekunden
6
7    // Cache überprüfen
8    if (!$smarty->is_cached('eins.tpl'))
9      {
10     // Code für Datenbank etc.
11     }
12   $smarty->display('eins.tpl')
13 ?>
```

Durch den Aufruf von `$smarty->caching=true;` schaltet man die Caching Funktionalität ein. Mit dem Aufruf von `$smarty->caching_lifetime=600;` wird die Lebensdauer im Cache definiert. Hier verfällt die Datei nach 600 Sekunden, und dann wird die Datei, beim nächsten Aufruf, neu kompiliert und wieder 600 Sekunden gecached.

Es gibt noch eine weiter sehr nützliche Funktion beim Caching. Diese Funktion nennt sich `$compile_check` und bedeutet, dass nun überprüft wird, ob es seit dem letzten kompilieren der Datei Änderungen gab. Diese Funktion beeinflusst die Performance nur gering. Jedoch ist eine manuelle Einstellung der »Lifetime« sinnvoller.

Um den Cache manuell zu leeren gibt es zwei weitere Funktionen. Mit der Funktion `clear_all_cache()` wird der gesamte Template-Cache gelöscht. Mit `clear_cache()` können einzelne Templates gelöscht werden.

Da Templates dynamisch sind ist es wichtig darauf zu achten, welche Inhalte für wie lange gecached werden sollen. Wenn sich zum Beispiel die erste Seite einer Website nur sporadisch ändert, macht es Sinn die Seite für eine Stunde oder länger zu cachen. Wenn aber eine Seite sich minütlich ändert, z.B. mit Wetterinformationen,

macht es möglicherweise keinen Sinn, die Seite überhaupt zu cachen[5].

3.2 EasyTemplateSystem

3.2.1 Einleitung EasyTemplateSystem

EasyTemplateSystem ist eine Template Engine die, wie Smarty auch in PHP geschrieben wurde. Diese Engine erlaubt es Daten in ein beliebiges Dokument zu überführen. EasyTemplateSystem, kann z.B. eine Menge von Daten, die aus einer Datenbank bezogen werden in ein HTML-Dokument tranformieren. Des weitern kann diese Engine auch SQL-Statements, ASCII-Code oder XML-Dokumente erstellen. Um dies zu berwerkstelligen kennt EasyTemplateSystem zwei Funktionen [9]:

- sprintt, erzeugt einen String aus dem Template
- printt, gibt das Template aus

EasyTemplateSystem arbeitet mit zwei Elementen, dem sogenannten »datatree« und das Template. Im »datatree« werden alle relevanten Daten bereit gestellt. Das Template definiert die Darstellung der Daten aus dem »datatree«. Man könnte es mit einem XML-Dokument vergleichen, welches aus einem XLST-Template erstellt wurde. Um nun von PHP heraus mit EasytemplateSystem zu arbeiten, stellt dieses System 2 Funktionen bereit:

Listing 8: ETS Funktionen für PHP

```
1 void printt {
2   (mixed datatree, mixed containers, [, string entry]}
3 //gibt das Template aus
4
5 void sprintt {
6   (mixed datatree, mixed containers, [, string entry]}
7 //gibt das Template zurück
```

`datatree` kann ein Objekt sein, oder ein Array von Objekten.

`containers` können entweder Arrays enthalten oder nur einfache Strings.

`entry` ist der Abschnitt in einem Template, der angesprochen werden soll. Standard

ist `main`.

3.2.2 Funktionsweise

Das nächste Code-Beispiel zeigt die Zuweisung der Platzhalter und die Übergabe an die Engine.

Listing 9: ETS datatree

```
1 $page->title = 'Home page';
2 $page->lastmodified = 'Thursday, January 23, 2003';
3
4 printt($page, 'test.tpl');
```

Da im Rahmen dieser Seminararbeit kann nicht der ganze Umfang dieser Engine gezeigt werden, hier werden nur ein paar wichtige Beispiele gezeigt. Für einen tieferen Einblick wird auf die Dokumentation verwiesen, da diese ausführliche Beispiele beschreibt. Die Dokumentation findet man auf der Webseite des Herstellers[1].

Nun wird gezeigt, wie ein Template für EasyTemplateSystem ausehen kann. Zu beachten ist hier, dass ein Template mit dem Tag `{mask:main}` beginnt und mit `{/mask}` endet. Da es sich hier um ein triviales Template handelt, musste in der Funktion `printt($page, 'test.tpl')` der Parameter für den Abschnitt nicht mit angeben werden, da standardmäßig `main` eingesetzt wird.

Listing 10: ETS Template-Code

```
1 {mask:main}
2 <html>
3   <head>
4         <title>{title}</title>
5   </head>
6   <body>
7     <h1>{title}</h1>
8     <hr>
```

[1] Hersteller: http://ets.sourceforge.net/ets.pdf

```
9       <div align="center">
10        Last modified: {lastmodified}
11      </div>
12    </body>
13  </html>
14  {/mask}
```

An Hand des nächsten Beispiels soll erklärt werden, wie der Umgang mit Objekten und Arrays funktioniert. Zu Beginn werden im »datatree« die Objekte erzeugt und Inhalt zugewiesen. In diesem Falle handelt es sich um das Objekte `partner` und um das Array mit den Objekten `menu[1]` und `menu[2]`.

Listing 11: ETS datatree

```
1  $page->title = 'Home page';
2  $page->lastmodified = 'Thursday, January 23, 2003';
3  $page->partner->name = 'foobar.com';
4  $page->partner->id = 123;
5  $page->menu[1]->url = "download.php";
6  $page->menu[1]->label = "Downloads";
7  $page->menu[2]->url = "links.php";
8  $page->menu[2]->label = "Links";
9
10 printt($page, 'test.tpl',menu,partner);
```

Interessant wird es nun, wie die Darstellung der Inhalte im Template aussieht. Als erstes wird die Funktion `printt($page, 'test.tpl',menu,partner);` erklärt. Da drei Abschnitte im Template definiert wurden, müssen diese von der Template-Engine auch befüllt werden. Dies geschieht über den Optionalen Parameter `[, string entry]`. Dabei ist zu beachten, dass die einzelnen Strings durch Kommata getrennt werden.

Listing 12: ETS Template-Code

```
{mask:main}
<html>
    <head><title>{title}</title></head>
    <body>
        <h1>{title}</h1><hr>
{mask:menu}<a href="{url}">{label}</a> {/mask}<hr>
        <div align="center">
        Last modified: {lastmodified}</div>
{mask:partner}
  <div>with our partner {name} ({id})</div>
{/mask}
    </body>
</html>
{/mask}
```

Die einzelnen Abschnitte im Template werden jeweils durch `{mask:menu}` und `{mask:partner}` eingeleitet, und durch das Tag `{/mask}` wieder geschlossen.

EasyTemplateSystem erlaub es auch PHP-Code direkt im Template zu implementieren. Damit die Engine diesen Code auch richtig interpretiert, muss der Code in den Tags `{PHP}` und `{/PHP}` eingebett sein. Ebenfalls wie bei der Smarty-Engine, sind auch Kontollstrukturen so wie Schleifen möglich. Im Anhang befindet sich eine kurze Referenz der möglichen Elemente.

3.2.3 Caching

EasyTemplateSystem ist ebenfalls, wie alle gängigen Template-Engines in der Lage Templates zu cachen. Jedoch ist das aktivieren bei dieser Engine nicht so konfortabel, wie z.B. bei Smarty oder Kajona. Um EasyTemplateSystem zum cachen auf zu fordern, sind drei Funktionen erforderlich:

Listing 13: ETS Cache-Funktionen

```
// read a compiled container and check if it?s not obsolete
function ets_cache_read_handler($id)
  {
  $content = FALSE;
  if (@filemtime("ets/$id.ets") > @filemtime("tpl/$id.html"))
    {
    if ($handle = @fopen("ets/$id.ets", 'rb'))
      {
      $size = @filesize("ets/$id.ets");
      $content = @fread($handle, $size);
      fclose($handle);
      }
    }
    return $content;
  }
// write a compiled container
function ets_cache_write_handler($id, $content)
  {
  if ($handle = @fopen("ets/$id.ets", 'wb'))
    {
    @fwrite($handle, $content);
    fclose($handle);
    }
  }

// read a source container
function ets_source_read_handler($id)
  {
  $content = FALSE;
  if ($handle = @fopen("tpl/$id.html", 'rb'))
    {
```

```
32      $size = @filesize("tpl/$id.html");
33      $content = @fread($handle, $size);
34      fclose($handle);
35      }
36      return $content;
37    }
```

3.3 Kajona

3.3.1 Einleitung

Die Template Engines die bisher vorgestellt wurden, hatten teilweise extra Code in den Templates. Wie es Smarty bevorzugt, Fallunterscheidungen oder ganze Blöcke, wie foreach-Anweisungen, direkt in das Template zuschreiben, soll nun eine Template Engine vorgestellt werden, die bis auf Platzhalter, und wenn nötig JavaScript, nur HTML-Code im Template verwendet. Diese Template Engine ist eine Eigenentwicklung und wurde im Rahmen eines Conten-Management-Systems entwickelt. Diese Engine steht unter der LGPL. Der Nachteil dieser Engine ist, dass sie speziell für das CMS entwickelt wurde, und nur sehr schwer portabel ist.

3.3.2 Funktionsweise

Eine besondere Eigenschaft dieser Engine ist, dass sich, je nach Einsatz, ein Template aus zwei anderen Templates zusammen setzt. Das Grundtemplate, im folgenden nur noch »Master-Template« genannt, ist von Großer Bedeutung. In diesem Template werden alle Element angelegt, welche auf der gesamten Seite, bzw. sehr häufig verwendet werden. Eine Navigation, Fusszeilen etc. wären solche Elemente. Diese werden einmal definiert, und können dann in anderen Templates per Platzhalter eingefügt werden. Um nun hier den Ansatz von drei Templates zuerläutern, wird vorrausgesetzt, dass auf der zu erstellenden Webseite, eine Newsverwaltung spezifiziert wurde. D. h. auf der Seite soll neben dem normalen Inhalt, auch News angezeigt werden.

Ein Master-Template kann wie folgt aussehen:

Listing 14: master.tpl

```
%%mastermainnavi_navigation%%
%%masterportalnavi_navigation%%
%%mastersearch_search%%
%%mastermodified_lastmodified%%
```

Da das Master-Template angelegt wurde, sollte nun das Template für die Darstellung der News erstellt werden:

Listing 15: master.tpl

```
<news_list>
<div class="newsList">
    <div class="newsListHeader">
        <div class="newsListTitle">
            <h3>%%news_start_date%% - %%news_title%%</h3>
        </div>
        <div class="newsListMore">%%news_more_link%%</div>
        <div class="cleaner"></div>
    </div>
    <div class="newsListTeaser">
        <div>%%news_intro%%</div>
    </div>
</div>
</news_list>


<news_detail>
<div class="newsDetail">
    <h3>%%news_title%%</h3>
    <p class="newsTeaser">%%news_intro%%</p>
    %%news_image%%
    <p>%%news_text%%</p>
```

```
23      <p>%%news_back_link%%</p>
24  </div>
25  </news_detail>
```

Das besondere an diesem Template ist, da es zwei Abschnitte beinhaltet. Der Abschnitt `<news_list></news_list` ist für die Darstellung der Listenansicht zuständig. Sollen die News in einer Liste, d.h. alle News untereinander, dargestellt werden, so wird dem Element News dieser Abschnitt, im Backend des CMS zugewiesen. Soll jedoch eine Detailansicht der News visualisiert werden, so wird der Abschnitt `<news_detail></news_detail>` zu gewiesen.

Nun fehlt nur noch das eigentliche Seiten Template. Aus Gründen der Übersicht, befindet sich dieses Listing im Anhang unter seiten.tpl.

Durch den Platzhalter `%%news_news%%` werden die News geladen und das dazu gehörige News-Template. Die Inhalte des News-Templates werden an dieser Stelle dargestellt.

Um nun Inhalte an die Engine zu übergeben, muss wie so oft erst einmal ein Objekt erzeugt werden, um dann über die jeweiligen Funktionen die Ausgabe zu steuern. Mit folgendem Ausdruck, werden die Daten an die Engine geschickt und ausgegeben:

Listing 16: Datenaufbereitung Modul-Template

```
1  $strOneNews .= $this->objTemplate->fillTemplate(
2    $arrOneNews, $strTemplateID);
```

In dem Array `$arrOneNews` stehen nun alle gesammelten Daten die ausgegeben werden sollen. Der nächste Parameter `$strTemplateID` beinhaltet das Template. Hier ist es das Mini-Template für die News. In einer weiteren Klasse, die für die Logik der einzelen Seiten zuständig ist, werden die Daten für die komplette Seite aufbereitet und mit den bereits fertig Daten der News kombiniert. Ist dies ohne Fehler geschehen, werden die neuen Daten wiederum an die Template-Engine gesendet und die fertige Seite wird ausgegeben.

Listing 17: Datenaufbereitung Seiten-Template

```
1 $this->strOutput = $strPageContent;
```

Im String `$strPageContent` stehen alle relevanten Daten zur Ausgabe der Seite bereit.

Und hier entsteht ein Nachteil, da die komplette Logik im PHP-Code geschieht, muss bei Arrays jedes Element nach einander an das Template übergeben werden. Dies bedeutet, dass es ein häufiges hin und her zwischen Template und Applikationslogik gibt.

Der große Vorteil hierbei ist, dass die Moduldarstellung nicht im jeweiligen Seiten Template implementiert werden muss, und die Übersichtlickeit gewahrt wird. Ein anderer Vorteil ist, wie oben schon erwähnt, dass hier keine Applikationslogik in den Templates steht, dies fördert die Bearbeitung des Layout´s. Die Applikationslogik wird ausschliesslich im PHP-Code implementiert.

3.3.3 Caching

Seit Version 2.1.0.0 der Template-Engine, ist auch hier Caching möglich. Die Caching-Funktion ist fest implementiert, und sollte eigentlich nicht deaktiviert werden. Falls es dennoch gewünscht ist, könnte dies in der Template Klasse, durch setzten einer Variablen gelöst werden. Da der Inhalt sich nicht sekündlich bzw. minütlich ändert, kann das Caching ohne Probleme aktiviert bleiben. Für den Fall das es doch eine Änderung gibt, welche sofort angezeigt werden muss, kann der Seiten-Cache manuell im Back-End leeren werden.

3.4 Templates ohne Template Engine

Bisher wurden Template Engines beschrieben, jedoch gibt es eine Möglichkeit, um auf eine komplexe Engine zuverzichten, ohne auf die gewünschten Vorteile einer Engine zuverzichten. Wie schon beschrieben ist die Trennung von Applikationslogik und Layout, ein gravierender Vorteil von Template Engines. Nun soll eine Methode vorgestellt werden, die auf eine Engine verzichtet, aber dennoch einen gewissen Grad an Flexibilität hat.

Mit *Template Engine nur mit PHP* und *Beyond the Template Engine* gibt es zwei Ansätze, die diese Idee verfolgen. Da sich *Template Engine nur mit PHP* und *Beyond the Template Engine* nicht groß von einander unterscheiden, wird hier primär auf den Ansatz von *Template Engines nur in PHP* eingegangen.

Bei diesen Ansätzen wird zwar behauptet, dass eine strickte Trennung von Applikationslogik und Layout besteht, aber dies ist nicht so. Zwar wird keine Logik in das Template übernommen, dies geschieht komplett im Applikationscode. Jedoch ist es hier nicht möglich nur Variablen im Template zu haben, denn wie aus Listing 18 zu entnehmen ist, geschieht das befüllen der Variablen im HTML-Code. Blöcke werden in den meisten Template Engines gleich behandelt. Zuerst wird ein Array erstellt, das alle Datensätze enthält, dann werden diese Datensätze ausgegeben.

Listing 18: Code für das Template

```
<body>
  <ul>
    <?php foreach ($block as $value): ?>
    <li>
      <?=$value?>
    </li>
    <?php endforeach; ?>
  </ul>
  <table width="100%" border="1">
    <?php foreach ($block as $value): ?>
    <td>
      <?=$value?>
    </td>
    <?php endforeach; ?>
  </table>
</body>
```

Listing 19: PHP-Code

```
<?
   $title_text = 'TITLE: This example contains two blocks';
   $block = array('Claus', 'Kelvin', 'Skrol', 'Daniel');

   require_once 'two_blocks.htm';
?>
```

Da dies nur ein kleines Beispiel ist, liegt die Frage nahe, wie sieht ein Template aus, wenn es sich um ein komplexeres Template mit mehrern Variablen handelt. Dies ist schnell beantwortet und liegt auf der Hand, dass dieser Code dann sehr schnell unübersichtlich wird, da kann dann auch kein Syntax-Highlighting weiter helfen. Daher ist diese Variante nicht zu empfehlen.

4 Schlussbemerkung

Wie man sieht, kann der Einsatz von einer Template-Engine einem Entwickler sehr hilfreich sein. Der größte Vorteil einer Template-Engine, ist die Trennung von Design und Applikations-Code. Dies ermöglicht, einem Team, parallel an einem Projekt zu arbeiten. Da es für gewöhnlich der Fall ist, dass Entwickler schlechte Designer sind und umgekehrt, können nun die Aufgaben getrennt werden.

Die große Herausforderung ist es, sich durch den Dschungel von Engine zu kämpfen, wenn der Einsatz einer Template-Engine gefordert wird. Die hier besprochenen Template-Engines sind nur ein Bruchteil derer, die es im Internet gibt. Aber, vor allem Smarty, gehören zu den beliebtesten und am häufigsten eingesetzten Engines. Noch zu erwähnen wäre die Engine *FastTemplate*, da aber im Rahmen dieser Arbeit auf eine Eigenentwicklung (Kajona) eingegangen wurde, die die Idee von *FastTemplate* weiterverfolgt, wurde bewusst auf diese Engine verzichtet.

Aus dieser Arbeit lässt sich schlussfolgern, dass Smarty die mächtigste, der hier vorgestellten Template-Engines ist. Sie ist leicht zu installieren und kann ohne Probleme in den eigenen Code integriert werden. Allerdings ist es, wie bei allen Engines, der Fall das es einer gewisse Einarbeitung bedarf, um die Engine auch voll zu nutzen.

Für sehr kleine Applikationen, bei denen es nicht zwingen notwendig ist eine Template Engine einzusetzen, wurden die zwei Möglichkeiten *Template Engine nur in PHP* und *Beyond the Template* beschrieben. Jedoch bei größeren Projekten soll von dem Einsatz dieser Ansätze abgeraten werden, da Teile der Applikationslogik im Template notwendig sind. Dadurch wird dieser Code sehr schnell unübersichtlich.

Schlussendlich bleibt es jedem selber überlassen, ob eine Eigenentwicklung sinnvoll ist oder nicht. Bei einer Eigenentwicklung ist darauf zu achten, dass es anfangs immer zu Problemen kommt und es eine gewisse Zeit braucht, bis alle »Kinderkrankheiten« behoben wurden. Für eine Eigenentwicklung spricht, dass die Engine natürlich an die eigenen Bedürfnisse angepasst wird, und kein unnötiger Ballast mit geschleppt werden muss.

Zu kritisieren wäre das umständliche Zusammenspiel von Applikationslogik und Layout. Werden auf einer Seite mehrere verschiedene Blöcke ausgegeben, wie zum Beispiel News, Stellenausschreibungen, Suche, Navigation, etc, muss zuerst für jeden Block der Inhalt generiert werden und zum Schluss alle gesammelten Daten noch einmal für das Endergebnis. Intern müssen alle Datensätze durch Schleifen einzeln an das Template übergeben werden.

Literatur

[1] Möhrke, Carsten (2004), Besser PHP programmieren, 1. korrigierter Nachdruck, Bonn (Galileo Press)

[2] Krause, Jörg (2004): PHP 5, Grundlagen und Profiwissen, Webserverprogrammierung unter Windows und Linux, München/ Wien (Carl Hanser Verlag)

[3] Lubkowitz, Mark (2006), Webseiten programmieren und gestalten, 4. korrigierte Nachdruck, Bonn (Galileo Press)

[4] Schröer, Michael (2005), Web Content Management mit PHP und MySQL, 1. Auflage , Bonn (Galileo Press)

[5] Smarty Template Engine, http://smarty.php.net/manual/de/ (abgerufen am 13.08.2007)

[6] Template Engine nur mit PHP, http://php-coding-standard.de/php_template_engine.php (abgerufen am 23.08.2007)

[7] Beyond The Template Engine, http://www.sitepoint.com/article/beyond-template-engine (aberufen am 25.08.2007)

[8] Contemplate, http://www.typea.net/software/contemplate/assembled/-home.html (abgerufen am 25.08.2007)

[9] Easy Template System, http://ets.sourceforge.net/ets.pdf (abgerufen am 20.08.2007)

A Anhang

A.1 Seiten Template

Listing 20: seiten.tpl

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Kajona [%%title%%]</title>
</head>
<body>

<table cellspacing="0" cellpadding="0">
  <tbody>
    <tr>
      <td colspan="2" rowspan="2" id="logo"></td>
      <td colspan="2" id="logoSpacer"></td>
    </tr>
    <tr>
      <td id="portalNavi">
            <ul>%%masterportalnavi_navigation%%</ul>
      </td>
      <td id="contentTopRightTop"></td>
        </tr>
        <tr>
          <td  id="moduleActionNaviThree"><div></div></td>
          <td id="naviContainer">
            <div id="statusBoxHeader"></div>
            <div id="statusBox">
              %%mastersearch_search%%
```

```
28                </div>
29                <div id="mainNavi">
30                  <ul>%%mastermainnavi_navigation%%</ul>
31                </div>
32                <div id="poweredBy">
33                  <img src="_webpath_/portal/pics/kajona/
34 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣kajona_poweredby.png"
35                   alt="powered␣by␣Kajona␣V3" />
36                </div>
37              </td>
38              <td  id="content">
39              %%headline_row%%
40              %%text_paragraph%%
41              %%picture1_image%%
42              %%news_news%%
43              %%gb1_guestbook%%
44              %%dl1_downloads%%
45              %%bilder_gallery%%
46              %%bilder2_galleryRandom%%
47              %%formular_form|tellafriend%%
48              %%results_search%%
49              %%sitemap_navigation%%
50              %%login_portallogin%%
51              %%switch_languageswitch%%
52              %%lm_lastmodified%%
53              %%faqs_faqs%%
54              </td>
55              <td id="contentTopRight"><div></div></td>
56              </tr>
57            <tr>
58              <td id="footerLeftCorner"></td>
59              <td id="footerLeft">
```

```
60                      _gentime_
61              </td>
62              <td id="footerRight"><div></div></td>
63              <td id="footerRightCorner"></td>
64          </tr>
65          <tr>
66              <td colspan="4" id="copyright">
67              %%mastermodified_lastmodified%%
68              %%copyright%%</td>
69          </tr>
70    </tbody>
71 </table>
72
73 </body>
74 </html>
```

A.2 ETS Referenz

Element	Description
{name}	Places an outer template or the value of a variable
{variable:template}	Places an outer template with a different name
{mask:name} ... {/mask}	Defines a template and, if inner, places it
{set:variable} ... {/set}	Places a content when a variable is set (not missing)
{set:variable:value} ... {/set}	Places a content when a variable has a specific value
{mis:variable} ... {/mis}	Places a content when a variable is missing
{mis:variable:value} ... {/mis}	Places a content when a variable has not a specific value or is missing
{php} ... {/php}	Places a content which will be parsed as PHP code
{const:template}	Places an outer template Places a content if a test is TRUE
{if:test} ... {/if}	Provides multiple conditions in conjunction with when-
{choose} ... {/choose}	test and else elements Places a content if a test is TRUE so disables other
{when:test} ... {/when}	when-test or else siblings Places a content if all when-test or when-value siblings
{else} ... {/else}	are FALSE Provides multiple conditions in conjunction with when-
{choose:variable} ... {/choose}	value and else elements Places a content if the variable of the choose parent

{when:value} ... {/when}	has the value of this element so disables other when-value or else siblings
	Places an outer template with arguments
{call:template} ... {/call}	Defines an argument of a call element
{arg:name} ... {/arg}	Places a content several times
{repeat:n} ... {/repeat}	Includes outer templates of a container
{include:container}	Places the content of a container as text
{insert:container}	Places the content of a container as a template part
{eval:container}	Places the content of a container as a template partwith restrictions
{safe:container}	Defines a size reducing behaviour
{* ... *}	Defines a comment section