Bitflux CMS

ein crossmedia-fähiges Web Content Management System

Proseminar von *Marcel Linnenfelser*

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Betreuer: Prof. Dr. Arnd Poetzsch-Heffter

Inhaltsverzeichnis

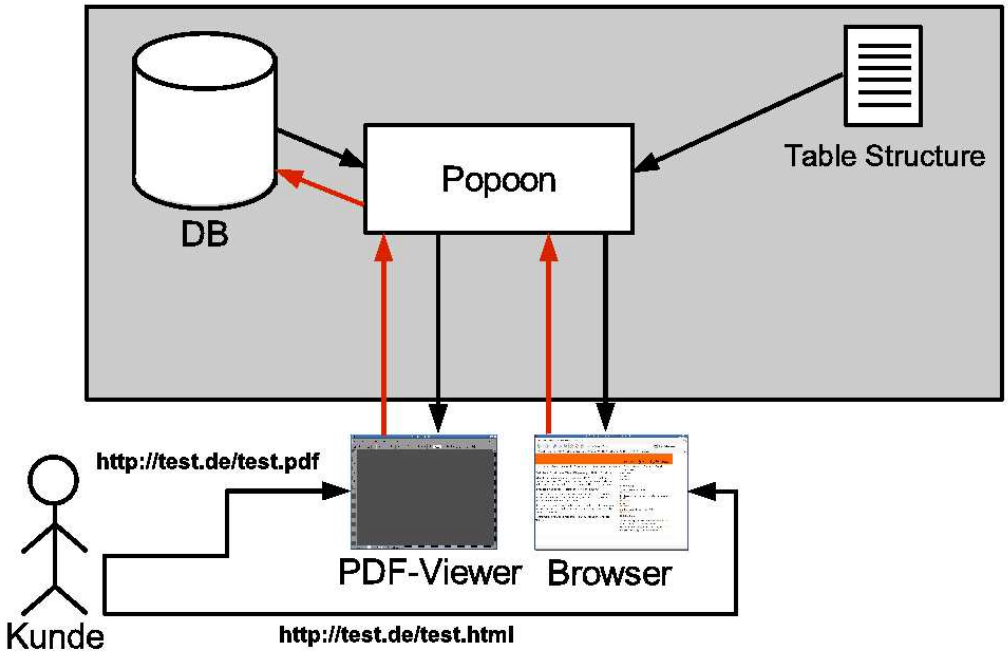

Abbildung 1: Bearbeitung einer Anfrage durch Popoon

1 Einführung

Bitflux CMS ist ein Web Content Management System, das auf den offiziellen W3C Standards XML und XSLT[1] aufsetzt. Zur Vereinfachung wird im folgenden CMS analog zu WCMS verwendet.

Die Template Engine basiert auf dem Popoon Framework, das eine nach PHP portierte Teilmenge von Apache Cocoon darstellt. Popoon ist ein komplexes Framework, das sich zur Erstellung von Webapplikationen eignet. Es basiert auf dem Prinzip der „separation of concerns", so dass mehrere Entwickler gleichzeitig an unterschiedlichen Teilen einer Applikation arbeiten können ohne Seiteneffekte befürchten zu müssen. Das Entwickeln einer Applikation gestaltet sich nach Aussagen der Cocoon Entwickler „LEGO-like", also wird eine Applikation eher gebaut als geplant.

Aufgrund der offenen Struktur und der konfigurierbaren Template Engine ist es leicht, unterschiedliche Quellen einzubinden und verschiedene Ausgabeformate zu erzeugen – auch wenn diese Ausgabeformate die Verwendung eines entsprechenden Backends erfordern.

Bearbeitung einer Anfrage

Sendet ein Client eine Anfrage an einen von einer Bitflux Installation verwalteten Web Site, so wird die URI an die Popoon Komponente weitergeleitet. Für den Client sieht es aus, als würde er eine normale Datei von einem Webserver anfordern. Betrifft die Anfrage eine Seite aus dem Bitflux System, so holt Popoon die entsprechenden Daten aus der Datenbank, generiert die Seite im entsprechenden Ausgabeformat und liefert sie an den Client zurück. Das Design von aus dem System angeforderten Seiten wird über XSLT-Templates gesteuert. Diese werden von einem XSLT-Processor interpretiert und verarbeiten die Inhalte einer Seite als DOM-Repräsentation.

[1] Für weitere Informationen zu XML und XSLT sei auf die Seiten des W3C http://www.w3c.org und die Literaturangaben verwiesen.

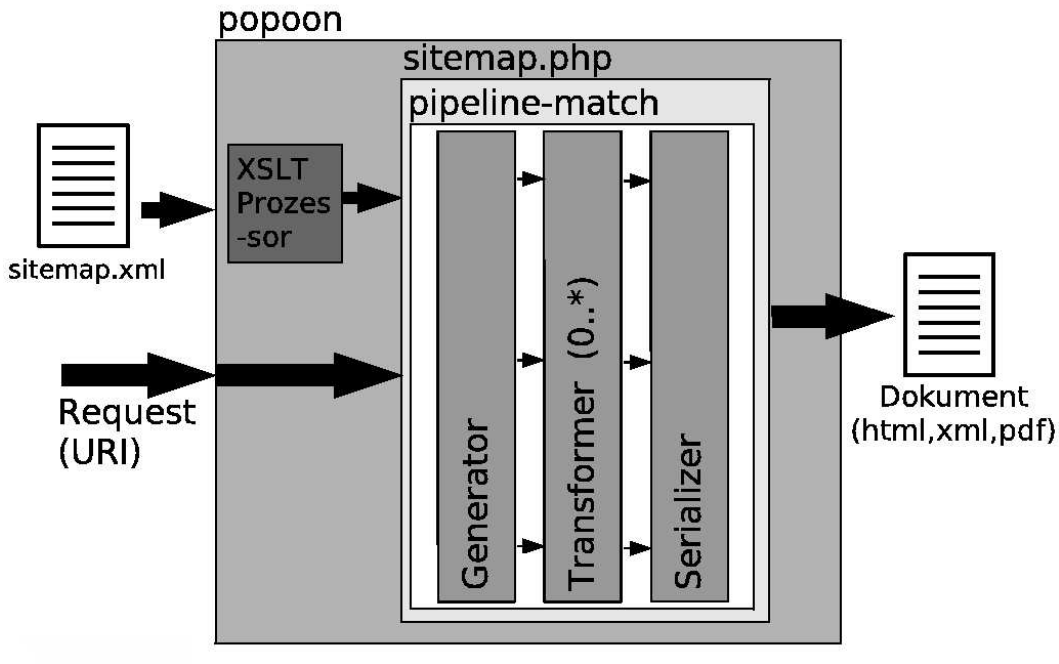

Abbildung 2: Aufbau von Popoon

Der letzte Transformer[2] einer Pipeline[3] erzeugt das Ausgabeformat direkt oder ein Eingabeformat für eine entsprechende Backend-Komponente wie etwa im Falle von PDF.

2 Architektur

Ein Redakteur gibt seine Inhalte über eine Administrationsoberfläche ein. Diese Oberfläche ist bei Bitflux CMS als Popoon-Applikation realisiert. Die Datenhaltung übernimmt eine Datenbank. Das darf zur Zeit nur eine MySQL-Datenbank sein, allerdings ist eine Portierung auf Oracle und andere in Arbeit[4].

2.1 Popoon

Zentraler Bestandteil von Popoon ist die Datei sitemap.xml[5]. Diese Datei ist das Herz von Popoon. In ihr wird genau definiert, in welchen Arbeitsschritten und durch welche Komponenten ein Dokument erzeugt wird. Eine solche Folge von Arbeitsschritten heißt in Popoon Pipeline. Eine Pipeline besteht im Wesentlichen aus den Komponenten Generator, Transformer und Serializer[6]. In den für uns relevanten Pipelines muss genau ein Generator und genau ein Serializer vorhanden sein. Die Transformer sind optional, je nach Ausgabeformat. Es können

[2] In Abschnitt 2.1 näher erklärt; Wandler, der die ursprüngliche, aus den Datenbankinhalten generierten, XML-Datei transformiert für die nächste Verarbeitungsstufe.

[3] Hintereinanderschaltung von Transformern und weiteren Komponenten. siehe Abschnitt 2.1.

[4] Diese Aufgabe sollte auch nicht allzu schwierig sein, da zur Ansteuerung der Datenbank schon die DB-Klassen aus dem PEAR (*P*HP *E*xtension and *A*pplication *R*epository) verwendet werden. Diese stellen eine datenbankunabhängige objektorientierte Schnittstelle zu unterschiedlichen Datenbanken zur Verfügung, ähnlich zu Java's JDBC oder Perl's DBI. So muss nur noch der SQL-Code von MySQL spezifischen Spracherweiterungen befreit werden.

[5] In Cocoon sitemap.xmap

[6] Es gibt noch einige weitere Komponenten, wie z.B. Actors, Readers und Aggregators, die für unsere Betrachtungen nicht weiter relevant sind.

beliebig viele definiert werden. Diese Komponenten sind alle jeweils als PHP-Klasse realisiert. Für jeden Typus gibt es eine abstrakte Superklasse.

Für die Datenbeschaffung sind in Popoon die Generators zuständig. Ein Generator ist die erste Komponente einer Pipeline und stellt das initiale XML-Dokument zur Verfügung. Er holt Daten aus verschiedenen Quellen, wie z.B. aus Dateien oder im Falle des „bitflux"-Generators, der in der Pipeline zum Generieren der CMS Dokumente genutzt wird, aus einer Datenbank nach dem Bitflux eigenen Datenbankschema. Der „bitflux"-Generator verwendet zum Datenbankzugriff die sql2xml-Klasse[7], die die Inhalte für die weiteren Bearbeitungsschritte als XML zur Verfügung stellt. Zum Erzeugen der SQL-Query wird eine XML-Datei geladen, in der die Entitäten und ihre Beziehungen abgebildet sind. Anhand der Query und der darin enthaltenen Joins erstellt eine sql2xml Instanz die Dokumentenstruktur des initialen XML-Dokuments.

Transformers sind für die Wandlung der XML-Daten zuständig. Sie erzeugen entweder das Ausgabeformat direkt oder eine Vorstufe. Die Eingabe und die Ausgabe eines Transformers sind allerdings immer XML-Daten[8]. Das entgültige Ergebnis einer Pipeline (Abbildung 4) erzeugt ein Serializer, der das Ende einer Pipeline darstellt. Bei HTML z.B. hat er allerdings nicht mehr zu tun als die interne Repräsentation in einen String zu wandeln und den „Content-Type"[9], also den MIME-Type, der Ausgabe zu setzen. Ein Serializer kann aber auch komplexere Aufgaben erledigen und ein Backend wie Apache Batik ansprechen, um z.B. von einem Transformer erzeugtes SVG etwa nach JPEG zu wandeln. Der Datenfluss in Popoon ist in Abbildung 2 dargestellt.

Es gibt verschiedene Standardtransformatoren. Der wohl am häufigsten verwendete ist der „xslt" Transformator. Er führt in einem XSLT-Prozessor ein angegebenes XSLT-Template aus, das das Eingabedokument verarbeitet und die Ausgabe für die nächste Pipelinestufe erzeugt. Es ist bei allen drei Komponentenarten[10] möglich, eigene zu definieren. Dazu muss nur eine PHP-Klasse erstellt werden, die von Generator bzw. Transformer bzw. Serializer ableitet und im entsprechenden Unterverzeichnis des „bitlib"-Verzeichnisses liegt. So können eigene Generators erstellt werden, mit denen man neue Datenquellen einbinden kann[11]. Cocoon und Popoon eignen sich also auch für EAI- und Content-Syndication-Projekte. Mit den vorhandenen und weiteren eigenen Serializers können diese Daten in alle erdenklichen Formate exportiert werden.

Um eine der definierten Pipelines auszuwählen, werden Matchers verwendet. Es wird die erste Pipeline gewählt, bei der der Matcher *wahr* liefert. Zudem ist es möglich die Ausgabe mehrerer Pipelines[12] durch Aggregators zu einem einzelnen XML-Dokument zu verbinden und dieses weiter durch die Pipeline laufen zu lassen, in der sich der jeweilige Aggregator befindet. Dabei ruft der Aggregator die anderen Pipelines auf. Aggregators sind erst seit kurzem in Popoon verfügbar.

Aus Performanzgründen wird die sitemap.xml in PHP-Code umgewandelt. Dies geschieht über ein XSLT-Template. Ein Neugenerieren findet nur statt, wenn sich die sitemap.xml geändert hat.

Wie in der Einführung erwähnt, verarbeiten die Generator-, Transformer- und Serializer-Klassen das zu bearbeitende Dokument als DOM-Repräsentation, was in der generierten sitemap.php zu erkennen ist. Die große Schwester Cocoon verwendet seit Version 2.0 SAX-Streams, da diese Möglichkeit wesentlich performanter ist. Allerdings ist das in PHP4 und auch PHP5 nicht möglich, da man für XSLT in PHP DOM verwenden muss. Laut Maintainer der XML-Erweiterung wird sich das auch in absehbarer Zeit nicht ändern. In Abbildung 3 ist die Definition einer Sitemap in XML zu sehen.

[7] In der Dokumentation wird sie noch als sql2xml bezeichnet, heißt aber mittlerweile db2xml
[8] Auch wohlgeformte HTML-, XHTML- und SVG-Dokumente sind valides XML
[9] So heißt die dafür zuständige Angabe im HTTP-Header
[10] Aber auch bei den Anderen, hier nicht betrachteten
[11] In Cocoon gibt es z.B. die Möglichkeit Daten aus einer SAP/R3 Installation einzubinden.
[12] genauer: Pipeline-Parts, von denen jeder eine vollständige Pipeline darstellt (also mit Generator und Serializer).

```
<map:pipelines>
    <map:pipeline>
        <map:match type="uri" pattern="/bitfluxgen/*">
           <map:generate type="bitflux"
                         src="../structure/simple.xml"
                         dsn="config://dsn"
                         redirectOnNotFound="http://bitflux.ch/"/>
           <map:transform type="phpprocessor"/>
           <map:transform type="xslt" src="globals://xslt"
                         xslparams="globals://XslParams"/>
           <map:serialize type="html" />
        </map:match>
    </map:pipeline>
</map:pipelines>
```

Abbildung 3: Die Pipeline für die HTML-Ausgabe von CMS-Dokumenten.

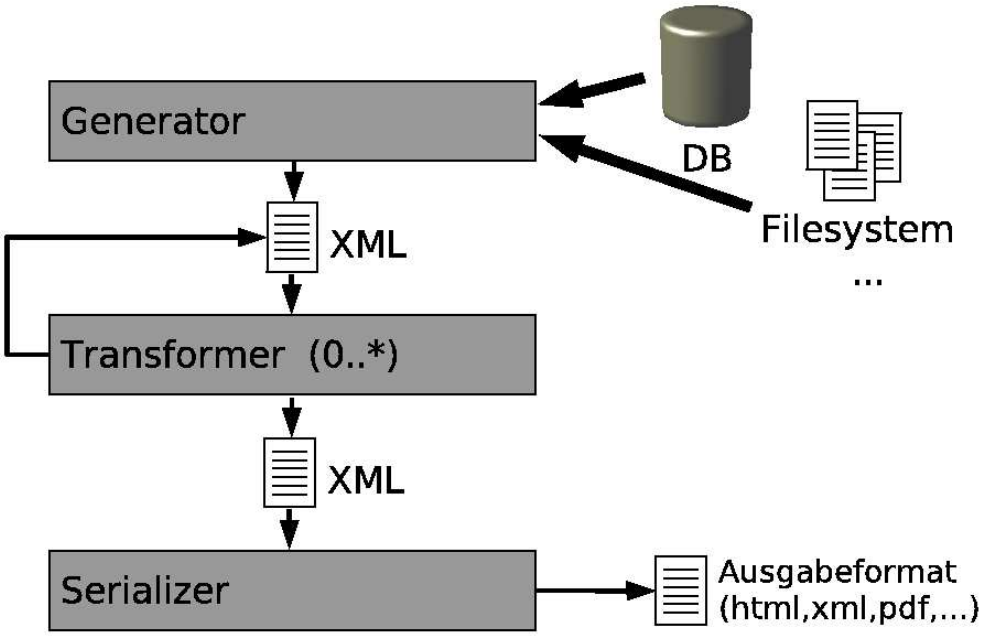

Abbildung 4: Datenelemente in Bitflux

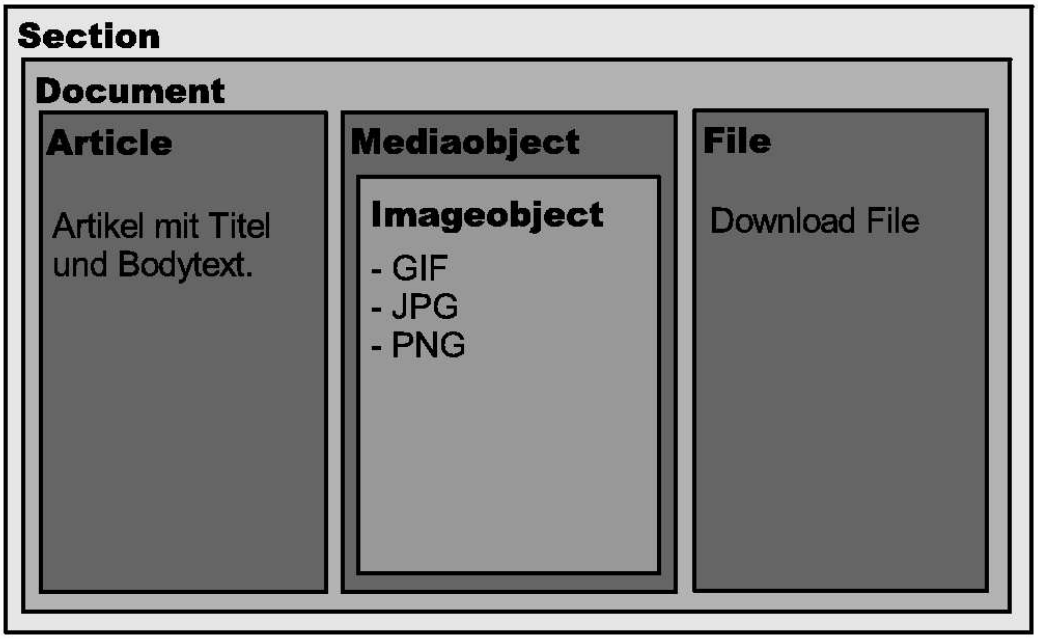

Abbildung 5: Datenelemente in Bitflux

Da die sitemap.xml, wenn Applikationen dazukommen, sehr schnell sehr groß werden kann, hat Popoon von der großen Schwester das Mounten von Subsitemaps übernommen. So kann man eine Sitemap auf mehrere Dateien aufteilen und damit die Übersichtlichkeit erhöhen. Außerdem vereinfacht es das gemeinsame Arbeiten an Projekten[13].

2.2 Bitflux Datenstruktur

Statt der Begriffe „Verzeichnis" oder „Ordner" verwendet Bitflux für die Elemente, die zur hierarchischen Strukturierung der Dokumente eines Site verwendet werden, den Begriff „Section". Dies ist schon deshalb unschön, da der Begriff Section noch einmal in der XML-Struktur des Articles auftaucht.

Eine Section kann dabei, ähnlich wie bei „hard Links" unter Unix, mehrere Vater-Sections haben. In den Sections können Documents, die Inhalts-Container in Bitflux, liegen. Auch hier kann, wie auch bei in den Documents befindlichen Articles, Imageobjects und Files, jedes Element mehrere Eltern haben.

Die Sections bilden den Pfad, unter dem die darin liegenden Documents erreichbar sind, genauso wie es bei einer Ordnerstruktur im Webspace eines Web Site wäre. Allerdings kann man einer Section mit dem Namen „foo" auch die URI[14] „bar" zuweisen oder eben keine. Wird keine URI zugewiesen, dann wird bei der Bildung des Pfades die betreffende Section einfach ausgelassen. Die zur Verarbeitung der Document-Inhalte angewendeten XSLT-Templates werden in den Sections festgelegt. Alle in einer Section befindlichen Documents verwenden das in der Eltern-Section festgelegte Template. Diese Einstellung wird an die Kind-Sections weitervererbt, solange bis einer Kind-Section wieder ein Template zugewiesen wird.

Sections bilden auch die Navigationsstruktur. Aus dem Section Baum wird über ein Navigations-Template eine Haupt- oder Subnavigation erstellt. Der Baum aus Sections ist im initialen

[13] Stichwort: „separation of concerns" (siehe Einleitung)

[14] Das ist der Name des Formularfeldes. Eigentlich ist hier der Teil einer URI gemeint, den das betreffende Element ausmacht. (z.B. in /home/**news**/news1.html den fettgedruckten Teil)

XML-Dokument, das der „bitflux"-Generator bereitstellt, enthalten. Mit den Möglichkeiten von XSLT lassen sich damit sehr individuell Navigationshilfen gestalten.

Auch Documents haben eine URI. Wird bei einem Request[15] nur ein Pfad angegeben, so wird das darin befindliche Document mit dem höchsten Rang[16] zurückgeliefert. Wird eine URI angegeben, die direkt auf ein Dokument zeigt, dann wird dieses Dokument zurückgeliefert.

Die eigentlichen Inhalte einer Bitflux Seite stellen die Elemente Imageobject, Article und File dar, wobei Imageobjects immer unter einem Mediaobject-Element eingehängt sind. Ein Document ist also ein Container für die Elementtypen Mediaobject, Article und File. Article besteht aus einem Titel und einem XML-Dokument für den Text, ein File-Element beinhaltet eine Datei und deren Namen und ein Mediaobject nur einen Namen. Die in einem Mediaobject liegenden Imageobjects, beinhalten jeweils eine Bilddatei und deren Namen[17]. Eingebunden in einen Article werden Mediaobjects und Files durch einen XML-Tag. Bei Mediaobjects wird dann beim Generieren des Documents das entsprechende Imageobject mit der richtigen Sprache ausgewählt. Das ist auch der Grund für diese Trennung. So kann einfach auf ein Mediaobject verwiesen werden, ohne sich um die Sprache eventueller Texte innerhalb des Bildes kümmern zu müssen. Bei den meisten Elementen lässt sich die Sprache einstellen. Dann werden beim Generieren, je nach Pipeline, die entsprechenden Sprachversionen der Elemente verwendet.

Die Artikel werden in der Reihenfolge des Einfügens im fertigen Dokument angezeigt. Es scheint keine Option zu geben, um diese Reihenfolge nachträglich zu ändern. Im Datenbankschema ist keine Kontrollmöglichkeit für die Reihenfolge vorgesehen. Bei Mediaobjects und Files ist die Reihenfolge des Einfügens nicht wichtig, da diese über entsprechende Tags in den Articles eingebunden werden. Dies ist allerdings auch ein Manko von Bitflux. Es überlässt dem Redakteur die gesamte Gestaltung des Content-Bereichs[18] dem Redakteur. Die Erfahrung zeigt, dass viele Redakteure (wenn nicht die meisten) damit überfordert sind. So kommt es häufig zu sehr unschönen Gestaltungen von Inhalten. Aus diesem Grund gehen viele professionelle CMS einen anderen Weg. Es gibt nicht nur Vorlagen mit einem Content-Bereich, der dann frei von den Redakteuren gestaltet werden kann, sondern die Vorlagen übernehmen auch das Platzieren von Mediadaten[19], Titeln und Textteilen. So muss sich der Redakteur darüber keine Gedanken mehr machen. Er kann sich völlig auf seine Kernkompetenzen konzentrieren und sorgt für Inhalte. Diese Möglichkeit bietet Bitflux derzeit nicht. Sicherlich wäre diese Funktionalität leicht durch konfigurierbare Formulare nachzurüsten, aus deren Eingaben der entsprechende XML-Code erzeugt werden könnte.

Die Hierarchie unter den Elementarten ist noch einmal in Abbildung 5 dargestellt.

2.3 Caching

Popoon bietet verschiedene Caching-Konzepte. An sehr vielen Stellen werden Daten vorgehalten, um sehr aufwendige Prozesse nicht unnötig durchführen zu müssen. Der Flaschenhals in der Performanz ist dabei die Datenbankabfrage und das Bereitstellen als XML für die Pipeline, also der Generator „bitflux"[20].

2.3.1 Sitemap Caching

Diese Art des Caching wurde bereits erwähnt. Um nicht immer die Sitemap parsen und ausführen zu müssen, wird sie einfach per XSLT nach PHP gewandelt. Als normaler PHP-Code wird

[15] Anforderung an den Server.
[16] Der Rang ist ein Integer-Wert, der zur Sortierung der Documents verwendet wird.
[17] Die Angaben wurden vereinfach und nur die wichtigsten dargestellt.
[18] So wird im Allgemeinen der Bereich einer Seite genannt, in den die zentralen Inhalte der Seite eingefügt werden.
[19] Bilder, Flash, Filme, Sound-Dateien, ...
[20] In Bezug auf den für das CMS relevanten Teil von Popoon.

sie vom PHP-Interpiler[21] ausgeführt. Das bringt einen extremen Performanzschub. Außerdem profitiert die Ausführung der Sitemap dann auch noch von eventuellem Bytecode Caching[22].

2.3.2 Component Caching

Das Caching der Generators fällt in Popoon (und Cocoon) unter das Component Caching. Dabei wird das Ergebnis nach dem Lauf eines Komponenten einer Pipeline im Cache gehalten. So können die Arbeitsschritte gespart werden. Beim „bitflux"-Generator bedeutet dies, das nicht mehr auf die Datenbank zugegriffen werden muss und kein Erzeugen einer hierarchischen XML-Struktur mehr nötig ist, so lange der Cache gültig ist.

2.3.3 Output Caching

Beim Output Caching wird das komplette ausgegebene Dokument gecached. So kann der komplette Vorgang zur Erstellung eines Dokuments gespart werden und statt dessen eine Datei gelesen und ausgegeben werden. Dies bringt wohl den größten Performanzschub. Durch die Anweisungen „include" und „include_once" kann sogar das unnötige Parsen von nicht ausgeführtem PHP-Code gespart werden. Wenn der Cache gültig ist, werden die Sitemap-Klasse und die Klassen der Komponenten (Generator, ...) erst gar nicht geladen. Das Ausliefern ist also nur wenig laufzeitintensiver als bei einer Datei im Wurzelverzeichnis des Webservers.

3 GUI

Die GUI des Bitflux CMS ist zur Zeit noch eher rudimentär. Man hat zwar alle Funktionen zum Verwalten der Dokumente an der Hand, jedoch ist die Bedienung nicht sonderlich intuitiv. Der Nutzen vieler Details der einzelnen Elementformulare erschließt sich schwer. Für einen unerfahrenen Benutzer ist das Benutzen der GUI ohne Schulung fast unmöglich.

Was bei kleinen Sites noch eine hilfreiche Einrichtung sein kann ist bei mittleren bis großen Sites eine Qual. Über jedem Elementformular gibt es eine Selectbox, über die man alle anderen gleichartigen Elemente schnell erreichen kann. Man stelle sich nun nur einen Site mit 5000 Dokumenten in 3 Sprachen vor, der im Schnitt 1,2 Articles pro Document beinhaltet. Das würde bedeuten, dass in diesem Menü, falls man sich in einem Article befindet, 18000 Einträge wären. Dass diese Selectbox kein schnelles Anwählen eines Article mehr erlauben würde ist klar. Doch viel schwerer wiegt, dass dieses Verhalten die ganze GUI unbenutzbar machen würde. Die Ladezeiten würden ein Arbeiten absolut unmöglich machen.

Für solche Massen an Elementen ist die Baumansicht auf der linken Seite der dreigeteilten Oberfläche schon besser gewappnet. Wird ein Element aufgeklappt, dann werden die Kinder dieses Elements nachgeladen. Hier ist nur störend, dass der Baum nicht automatisch aktualisiert wird, wenn man ein Element hinzufügt. Um das Nachladen zu erreichen muss man auf ein unscheinbares „R" in der rechten oberen Ecke klicken. Allerdings ist dann der komplette Baum wieder eingeklappt, so dass man sich wieder hindurch hangeln muss bis an die Stelle, an der sich das gesuchte Dokument befindet.

Im Datenbankschema von Bitflux ist schon die Möglichkeit zur Versionierung der Elemente vorhanden. Die Versionen werden auch schon mitgeschrieben. Jedoch gibt es noch keine Möglichkeit in der GUI, um darauf zuzugreifen. Wenn es in einer der nächsten Softwareversionen diese Möglichkeit gibt, so kann man direkt auf alle gemachten Änderungen zugreifen und diese wieder rückgängig machen.

Alles in allem ist die GUI der uninteressanteste und unspektakulärste Teil des vorgestellten Systems. Was interessant ist, ist nur wie sie programmiert wurde. Doch das Erstellen von Webapplikationen mit Popoon/Cocoon ist nicht Thema dieses Proseminars.

[21] Vor jedem Script-Lauf in Bytecode compilierender Interpreter.
[22] Z.B. mit dem kommerziellen Zend Accelerator oder dem freien APC aus dem PECL (http://pecl.php.net).

Zur Eingabe und zum Editieren von Articles kann auch der Bitflux Editor verwendet werden. Das ist ein WYSIWYG-Editor, der die Eingabe im Stile einer Textverarbeitung gestattet, und einem so die Eingabe von XML-Code erspart. Allerdings befindet er sich noch in Version 0.2 alpha, was ihm auch stellenweise deutlich anzumerken ist. Zudem ist er nur unter Mozilla ab Version 1.4 lauffähig. Das ist aber nicht so tragisch, da Mozilla auf den meisten Plattformen zu Hause ist – ganz im Gegensatz zum am stärksten verbreiteten Browser, dem Internet Explorer, den es nach der Einstellung der Mac-Version nur noch für Windows gibt.

4 Fazit

Bitflux CMS ist ein technisch sehr interessantes Projekt. Es wurden viele innovative Konzepte realisiert. Leider beschränkt sich die Innovation auf die Technik unter der Haube. Die Oberfläche ist nicht leicht und intuitiv bedienbar. Positiv sticht hier der WYSIWYG-XML-Editor heraus – mit den oben erwähnten Limitationen.

Das System ist auf fast allen unixartigen Plattformen und Windows zu Hause. Es benötigt einen Apache Webserver mit „mod_rewrite" und „mod_php" und eine MySQL Datenbank. Ein Port auf andere Datenbanken ist in Arbeit. Somit sollte niemand ein Problem haben, Bitflux auf seiner Wunschplattform zum Laufen zu bekommen.

Es gibt weder Freigabekontrolle noch Workflow. Die Benutzerverwaltung kann keine Gruppen erzeugen. Sehr schade ist, dass es keine Trennung zwischen Live- und Redaktionsdatenbank gibt. Alle Änderungen sind sofort online sichtbar.

Es bleibt zu hoffen, dass Bitflux CMS eine leistungsfähigere und intuitivere Oberfläche bekommt und die Defizite im Bereich der Mehrbenutzerfähigkeiten ausgeglichen werden, um die technischen Finessen dieses Projektes einem breiteren Publikum schmackhaft zu machen.

5 Literatur

- Bitflux CMS Web Site : http://www.bitflux.org
- Cocoon Web Site : http://cocoon.apache.org
- W3C Web Site zur XSL Spezifikation : http://www.w3.org/Style/XSL/
- PHP Web Site : http://www.php.net
- Web Site des PHP Extension und Application Repository : http://pear.php.net
- Alexander Schatten, Reinhard Pötz „Wie von Zauberhand" : iX März 2004
- Stefan Mintert „XML & Co." : Addison-Wesley August 2002
- Dirk Ammelburger „XML" : Hanser Fachbuchverlag November 2003
- Stephan Niedermeier „Cocoon 2 und Tomcat" : Galileo Press April 2004