Praktikumsbericht

Andreas Gohr
<a.gohr@web.de>

9. Juli 2002

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Rahmen meines Studiums der Angewandten Informatik an der Fachhochschule für Technik und Wirtschaft Berlin absolvierte ich im Sommersemester 2002 mein Praxissemester bei der *NetCologne Gesellschaft für Telekomunikation mbH* in Köln. Der Praktikumszeitraum umfaßte 15 Wochen (3.4.2002 – 15.7.2002). Während dieser Zeit wurde ich durch Ulrich Babiak – Systemadministrator und Webmaster von *koeln.de* – betreut.
Die NetCologne GmbH ist eine regionale Telekomunikationsgesellschaft im Großraum Köln/Bonn (sowie den umliegenden Gemeinden). Zu Ihren Dienstleistungen gehören unter anderem Telefondienste, Online-Dienste/Internet, Kabel-TV, Übertragungswege, Corporate Networks und Datenservices. Netcologne besteht seit 1994 und beschäftigt ca. 500 Mitarbeiter.
Mein Praktikum absolvierte ich in der Abteilung “Content”, der jüngsten Abteilung bei NetCologne. Sie wurde 2000 gegründet und ist für die Gestaltung der Internetpräsenzen *koeln.de*[1], *bonnXXL*[2], *koelnverkehr*[3] sowie diverser kleinerer Webangebote für Köln und Umgebung verantwortlich.
Insgesamt arbeiten in dieser Abteilung zehn festangestellte Mitarbeiter in den Bereichen Webdesign, Online-Redaktion, Technik, Verwaltung und Marketing.
Koeln.de ist das im Auftrag der Stadt Köln von NetCologne betreute Städteportal für Köln und Umgebung. Die Seite verzeichnet rund 9,2 Millionen Pageimpressions monatlich.

[1] http://www.koeln.de
[2] http://www.bonnxxl.de
[3] http://www.koelnverkehr.de

Kapitel 2

Projekte

Im Laufe meines Praktikums wurde ich mit verschiedenen kleineren und größeren Aufgaben betraut.

2.0.1 “koelnverkehr”

Eine meiner ersten Aufgaben war es, zwei Scripte zu überarbeiten, welche für *koelnverkehr.de*[1] aktuelle Verkehrs- und Parkhausinformationen in die Website einfügten. Dabei wurden die Informationen per email geliefert und über ein sendmail alias in die von mir geschriebenen Perl-Scripte gepiped.
Über *Reguläre Ausdrücke* wurde der Inhalt der Mails geparst und der entsprechende HTML als *ServerSideInclude* generiert.

2.0.2 Serverumzug

Da die Websites des *koeln.de* Netzwerkes auf einem inzwischen an seiner Grenze arbeitenden Pentium 2 liefen, stand ein Umzug auf ein Pentium 4 Dualprozessorsystem an. Dieses stand bereits mit einem vorkonfigurierten *SuSe Linux*[2] zur Verfügung.
Meine Aufgabe bestand darin, speziell an unsere Bedürfnisse angepasste Versionen des *Apache* Webservers und den benötigten *PHP3* und *PHP4* Modulen zu compilieren, installieren und zu testen.
Nach dem erfolgreichen Umzug der Webseiten wurde ich mit der Neueinrichtung des alten Servers betraut, welcher fortan unterstützende Dienste wie Logfile-Auswertungen ausführen sollte. Neben dem Austausch einiger stark belasteter

[1] http://www.koelnverkehr.de
[2] http://www.suse.de

Hardwarkomponenten (Festplatten, Lüfter) war auch die Installation einer aktuellen Linux-Distribution nötig. Meine Wahl viel dabei auf *Debian GNU/Linux 3.0*[3], welches sich vor allem durch sein einfaches und zuverlässiges Update-System und hohe Stabilität auszeichnet.

2.1 Dokusystem

2.1.1 Aufgabe

Aufgabe war es, ein möglichst einfaches System zur Verwaltung von Dokumentationen zu verschiedenen Themen zu schaffen, das folgenden Anforderungen genügt:

- Verwaltung unterschiedlichster Dateitypen
- Komfortable Suche
- Einfache Bedienung
- Vermeiden von “Versionswirrwar”
- Problemlose Weitergabe von Dokumenten an externe Abteilungen

2.1.2 Lösungsansatz

Die Daten werden als Dateien in einer über *Samba*[4] freigegeben Verzeichnisstruktur abgelegt. Dies ermöglicht das direkte Bearbeiten der Daten auf dem Server mit den gewohnten Werkzeugen (MS Word, Notepad, etc.) und vermeidet verschiedene Versionen auf verschiedenen Rechnern.
Zusätzlich werden zu jeder Datei Metainformationen in einer *MySQL*[5] Datenbank abgelegt, welche das Suchen nach Dokumenten ermöglichen. Außerdem wird zu jeder Datei ein Auszug des Inhalts in der Datenbank abgelegt, so dass auch ein Teil des Inhalts durchsuchbar wird. Wie dieser Auszug erstellt wird kann für jeden Dateityp festgelegt werden. Die Suche und das Bearbeiten der Metadaten erfolgt über ein PHP-Frontend im Browser.
Um eine Synchronisation zwischen den Informationen in der Datenbank und den im Dateisystem vorhandenen Dateien zu gewährleisten, existiert ein Perl-Backend. Dieses nimmt automatisch neu gefundene Dateien in die Datenbank auf, checkt ob die Informationen in dieser noch stimmen und kümmert sich um die Indizierung der Dateien.

[3] http://www.debian.org
[4] http://www.samba.org
[5] http://www.mysql.org

2.1.3 Eingesetze Software

Auf dem Server wurde folgende Software eingesetzt:

- *Perl* mit den Modulen `DBI` und `DBD::mysql` für das Backend
- *PHP 4* für das Frontend
- *mySQL* Datenbank
- *Apache*[6] Webserver
- *Samba*

Auf den Clients kommt ausschließlich Microsoft *Windows NT* sowie ein Browser der 4er Generation oder höher zum Einsatz.

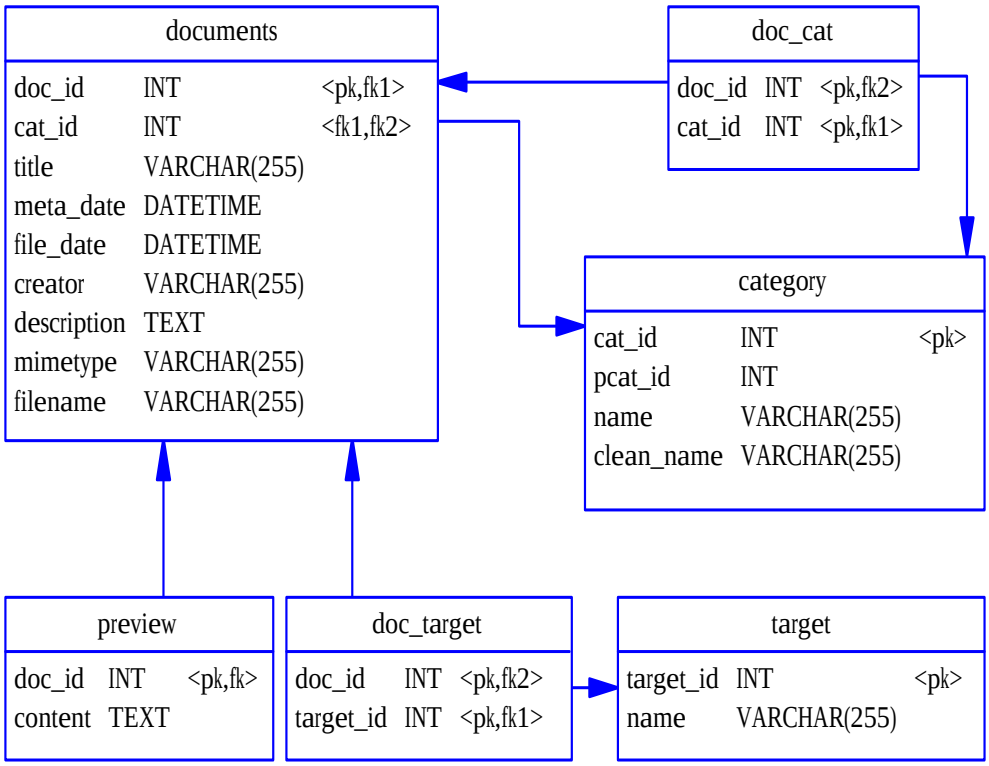

Abbildung 2.1: Datenbank Schema

[6] http://httpd.apache.org

2.1.4 Installation und Konfigurationsmöglichkeiten

Die Installation erfolgt durch das Kopieren aller Dateien in ein Verzeichnis unterhalb des Webserver Dokumentenverzeichnis.
Sowohl das PHP-Frontend als auch das Perl-Backend lässt sich über Konfigurationsdateien anpassen. Dafür habe ich einen Configparser in *PHP* und *Perl* geschrieben, welcher eine relativ einfache Syntax beherrscht: Zeilen die kein Gleichheitszeichen (=) enthalten werden ignoriert. Alles hinter einem "#" wird als Kommentar betrachtet und ebenfalls ignoriert. Jede Zeile besteht aus einer Variable auf der linken Seite, der ein Wert auf der linken Seite des Gleichheitszeichen-Zeichens zugewiesen wird. Whitespaces um Variablenname und Wert werden abgeschnitten.
Neben den Einstellungsmöglichkeiten über die Konfigurationsdateien habe ich weitere Anpassungen des Frontends über ein CSS Stylesheet, sowie zwei anpassbare Funktionen `head()` und `foot()`, welche auf jeder Seite des Frontends aufgerufen werden ermöglicht.

2.1.5 Verwalten von Dokumenten

Dokumente können grundsätzlich auf zwei Arten hinzugefügt werden: Über das Dateisystem oder via Browser-Upload.
Um ein beliebiges Dokument im Dokumentationssystem zur Verfügung zu stellen, kann es einfach an einen beliebigen Ort unterhalb des *Dokumentenverzeichnises* kopiert werden. Beim nächsten Durchlauf des Backends wird die Datei als neu erkannt und in das System aufgenommen.
Bei der automatischen Aufnahme der Datei werden keine zusätzlichen Metaangaben im System abgelegt, was das Auffinden der Information erschwert. Selbstverständlich können diese Metadaten jederzeit nachträglich eingetragen werden.
Die zweite Methode nutzt einen Dokumentenupload über den Browser. Dabei können gleich zusätzliche Metadaten angegeben werden.
Metadaten sind zusätzliche Informationen zu einem Dokument, die das Auffinden desselben erleichtern sollen. Dazu zählen die Kategorien in welche das Dokument eingeordnet werden muß, sowie Angaben wie Titel, Autor, Beschreibung und Zielgruppen.
Über eine Suche lässt sich so ein Dokument schnell auffinden und gegebenenfalls weiterbearbiten. Dabei verweisen die Links aus dem Frontend immer als `file://`-Link auf das über *Samba* freigegebene Netzlaufwerk, so daß Änderungen immer am Originaldokument auf dem Server vorgenommen werden.
Das Perlbackend `syncdoku.pl` sorgt für die Synchronisation zwischen Filesystem und Datenbank. Es sollte mindestens einmal täglich via `cron` gestartet werden.

`syncdoku.pl` verhält sich so, dass keine Daten verloren gehen können. Hier die Arbeitsweise im groben Überblick:

- Existiert eine in der Datenbank angegebene Datei nicht mehr, so wird der Verweis auf sie entfernt, die Metadaten bleiben jedoch erhalten.
- Wird eine Datei gefunden, die noch nicht in der Datenbank vorhanden ist, so wird ein Dokument für sie angelegt. (Die Metadaten bleiben leer.)
- Wird eine Datei gefunden, die bereits eine Dokumenten-ID besitzt, sich aber in einem andere Verzeichnis befindet als die Kategorie des Dokuments angibt, so wird der Primärkategorieeintrag des Dokuments geändert.
- Für neue Verzeichnisse werden automatisch entsprechende Kategorieeinträge angelegt.
- Fehlt das entsprechende Verzeichnis für eine Kategorie, so wird es angelegt.
- Ist das Datum einer Datei neuer als das in der Datenbank gespeicherte, so wird die Datei neu indiziert.

Über Kommandozeilenoptionen kann das Verhalten des Backends beeinflusst werden:

- `-f` erzwingt das Neuindizieren aller Dateien
- `-v` gibt Informationen über den aktuellen Arbeitsschritt aus

2.2 WM-Special

2.2.1 Externer Content

Pünktlich zur Fußballweltmeisterschaft 2002 sollte auf *koeln.de* und der Partnerseite *bonnXXL* ein WM-Special erscheinen. Dazu wurde bei der *Altus Media GmbH* externer Content eingekauft. Konkret sollten zu jedem Spiel der WM folgende Beiträge geliefert werden:

- ein Liveticker der während des laufenden Spiels die aktuellen Geschehnisse wiedergibt
- die Aufstellung der Mannschaften
- ein Spielbericht
- Statistiken zum Spiel (also wieviele rote und gelbe Karten, Anzahl der Tore, etc.)
- Stimmen zum Spiel (kurze Zitate von Trainern und Spielern)
- eine Bildergalerie mit 10 bis 15 Bildern pro Spiel

Mit Altus Media wurde vereinbart, daß der Content als `XML` zur Verfügung gestellt werden würde. Die Daten sollten via `HTTP` abrufbar sein. Ein großes Problem stellte die Kommunikation mit Altus und der Zeitdruck dar. Da Altus Teile des Contents selbst einkaufte, lagen Beispieldaten erst eine Woche und Informationen zu Format und Art der Lieferung der Bilder sogar erst einen Tag vor Spielbeginn vor.

2.2.2 Aufgabe

Um trotz der knappen Zeit noch ein attraktives und stets aktuelles Angebot auf die Beine zu stellen, sollten alle Daten automatisch verarbeitet und in das Seitenlayout von *koeln.de* bzw. *bonnXXL.de* einpasst werden.
Meine Aufgabe war es nun, die Daten von Altus zu holen, zu verarbeiten und unter Verwendung von Templates entsprechende HTML Seiten zu generieren.

2.2.3 Datenverarbeitung

Als Programmiersprache kam *Perl* zum Einsatz, welches sich durch Zuhilfenahme diverser Module hervorragend zur Verarbeitung von XML-Daten eignet.

Leider lag uns von Altus Media keine XML Document Definition vor, so daß ich gezwungen war, anhand der gelieferten Beispieldaten zu arbeiten, was sich jedoch als nicht weiter schwierig erwies, da die Struktur der Daten relativ einfach aufgebaut war. Es gab nur zwei Formate: Eines für den Liveticker und eines für die restlichen Inhalte.
Ich schrieb also zwei Scripte, deren Aufbau prinzipiell gleich war, sich jedoch in der speziellen Verarbeitung der Daten unterschied.
Hier der grobe Ablauf:

1. Holen der als Kommandozeilenparameter angegebenen XML Datei über das HTTP-Protocol (mittels `LWP::Simple`)

2. Parsen der Daten in eine mehrdimensionale Datenstruktur (Parsing via `XML::Simple`)

3. Dekodieren von Umlauten und Sonderzeichen soweit nötig (`UML::String`)

4. Einlesen des auf der Kommandozeile spezifizierten Templates und Ersetzen der darin enthaltenen Platzhalter durch die Werte aus dem XML-File (Via Regular Expressions)

5. Schreiben des Outputs in die ebenfalls als Kommandozeilenparameter gegebene Outputdatei

2.2.4 Controller

Nun mußte noch ein Weg gefunden werden, diesen Vorgang zu aktualisieren, das heißt, die Daten zu den richtigen Zeitpunkten zu holen, so daß beispielsweise der Ticker auch wirklich während des laufenden Spiels und der Spielbericht nach dem Spiel aktualisiert wird.
Die Spielzeiten standen mir von einer anderen Anwendung her bereits in einer MySQL-Datenbank zur Verfügung. Desweiteren hatte jedes Spiel eine eindeutige ID von Altus bekommen unter der die Daten zur Verfügung gestellt wurden.
Ich erweiterte also die Datenbank um die Information der AltusID und konnte so über eine SQL-Abfrage die aktuellen Spiele erfahren.
Als nächstes schrieb ich erneut zwei Scripte, die während der WM rund um die Uhr laufen und die zwei XML-Konverter steuern sollten. Auch hier gleicht sich der Ablauf beider Scripte wieder und hätte hier sicherlich über Parametrisierung sogar zu einem zusammengefasst werden können. Aufgrund des Zeitdrucks entschieden wir uns jedoch für eine “Copy’n’Paste”-Lösung, die die individuelle Steuerung des Tickers und der restlichen Beiträge erlaubte.

Um sicherzustellen, daß das Controllerscript auch bei unvorhergesehenen Problemen weiterläuft, konzipierte ich es in zwei Schichten. Gleich nach dem Start wird eine Endlosschleife gestartet, in der ein *Child-Prozess* geforked und auf dessen Beendigung gewartet wird (`wait`). Erst innerhalb dieses Childs wird die eigentliche Arbeit getan. Diese Methode sichert ab, daß selbst wenn das Programm aus irgendeinem Grund abstürzen sollte sofort ein neuer Prozess gestartet wird.
Innerhalb des *Child*s läuft dann eine Schleife, die die aktuellen Spiele abfragt und für jedes Spiel die entsprechenden Datenverarbeitungsscripte mit den richtigen Parametern aufruft. Dieser Vorgang wird gesteuert über einen `sleep` Timer, ebenfalls endlos wiederholt.
Diese Scripte arbeiteten während der gesamten WM ohne größere Probleme, lediglich die Sonderzeichenkonvertierung bereitete mir anfangs einiges Kopfzerbrechen, bis ich das Problem über das `Unicode::String` Modul lösen konnte.

2.2.5 Bilderscript

Für die Verarbeitung der Bilder musste ich mir etwas gänzlich anderes einfallen lassen, da hier keine XML-Daten vorlagen.
Die Daten wurden über das Unix-Tool `wget` in ein vorgegebenes Verzeichnis auf unserem Server geladen, wobei die von Altus vorgegebene Verzeichnisstruktur beibehalten wurde. Dabei wurde für jedes Spiel ein der eindeutigen AltusID entsprechendes Unterverzeichnis erstellt.
Die Bilder kamen in zwei Größen: Thumbnails, welche durch ein vorangestelltes `t_` gekennzeichnet wurden, sowie normal große Bilder beginnend mit einem `n_`.
Mein Script brauchte hier nur das Verzeichnis zu durchsuchen, aus dem Verzeichnisnamen die Altus-ID herrauslesen und die Dateien nach einer Bereinigung von Sonderzeichen an die gewünschten Stellen kopieren.
Mit Hilfe von Templates wurde dann eine Übersichtsseite mit den Thumbnails, sowie eine HTML-Seite pro Bild erstellt.
Ein *Cronjob* erledigte dann das automatische Starten alle 15 Minuten.
Um zu verhindern, dass Galerien immer wieder generiert werden, wurde in jedem Verzeichnis, das bereits erfolgreich bearbeitet wurde, eine *LOCK*-Datei abgelegt.

2.3 koeln.de Suche

2.3.1 Einführung

Auf *koeln.de* existierte bereits eine mit *ht://dig* realisierte Volltextsuche. Neben der Volltextsuche im "normalen" Inhalt gab es jedoch weitere Quellen zu durchsuchen: den "*Marktplatz*" - ein Branchenverzeichnis für den Großraum Köln/Bonn und den "*NetCologne Navigator*[7]" - ein Verzeichnis regionaler Webseiten, welches extern von *allesklar.com*[8] betrieben wurde.
Zusätzlich wurde Werbung zu bestimmten Suchwörtern verkauft bzw. eigene Angebote von *koeln.de* darüber geteasert.
Um ein einheitliches Suchinterface zu schaffen, existierte ein PHP-Script, welches die unterschiedlichen Quellen abrief und deren HTML-Output parste, um dann ein zusammengesetztes Suchergebnis präsentieren zu können. Da dieses Script relativ unflexibel was das Ändern und Hinzufügen von weiteren Quellen war, sollte die komplette Suche für das koeln.de Netzwerk überarbeitet werden.

2.3.2 Backend

Da die komplette Suche neu konzipiert werden sollte, stellte sich die Frage, welche Suchmaschine für die Volltextsuche benutzt werden sollte. Meine erste Aufgabe war es also, mehrere Kandidaten auf ihre Tauglichkeit hin zu testen.
Die Anforderungen waren unter anderem:

- Unix/Linux-Platform
- Anpassbare Ausgabe $Templates$
- Schneller und flexibler HTTP-Spider
  - da möglicherweise auch externe Seiten durchsucht werden sollten (z.B. *Stadt-Koeln.de*[9]) mit einstellbarer Suchgeschwindigkeit ("throttling")
  - mit selbst definierbarem "user-agent" Header
- Anlegen mehrerer *Collections* - also getrennter Datenbestände innerhalb des Index
- Möglichkeit, Teile einer Seite aus dem Index auszuschließen

[7] http://navigator.netcologne.de
[8] allesklar.com AG Wilhelm-Ostwald-Str. 0 55721 Siegburg
[9] http://www.stadt-koeln.de

- Unterstützung des *Robot-Exclusion-Standards* (robots.txt) sowie des robot Metatags

- Beeinflussung der Relevanz einzelner Seiten durch bestimmte Tags bzw. Möglichkeit, die Relevanzkriterien selbst vergeben zu können

Letztenendes standen uns drei Maschinen zur Auswahl: Die OpenSource-Lösungen *ht://dig*[10] und *mnoGo-Search*[11] sowie die kommerzielle Engine von *Inktomi*[12].
Alle drei Maschinen boten in etwa die selben Features, wobei natürlich jede ihre speziellen Fähigkeiten hatte.
Am Ende entschieden wir uns nach ausführlichen Tests für *mnoGo-Search*, welche vor allem durch die Geschwindigkeit des Spiders und die hohe Transparenz durch den Einsatz einer MySQL-Datenbank hervorstach.

2.3.3 Frontend

Der größere Teil des Projektes bestand im Entwurf und der Programmierung des Frontends. Auch hier gab es wieder zahlreiche Anforderungen, die es zu erfüllen galt:

- möglichst schnell

- konsistentes Interface

- Durchsuchen mehrerer *Collections*

- Verwendbarkeit auf mehreren Sites innerhalb des koeln.de Netzwerks

- flexibel anpassbar über Templates

Wir entschieden uns dafür, das Frontend in PHP zu schreiben, um den Overhead eines Perl CGIs (laden des Perl Interpreters) zu sparen und zusätzliche Apache-Module (wie *modperl*) zu vermeiden.

[10] http://www.htdig.org
[11] http://www.mnogosearch.org
[12] http://www.inktomi.com

2.3.4 XML

Um flexibel neue *sources* und *collections* [13] hinzufügen zu können, musste ein einheitliches Format für Suchergebnisse definiert werden. Dafür bot sich natürlich XML an. Ich legte also in einem XML Schema fest, wie Suchergebnisse an mein Frontend geliefert werden sollten. Das Stellen der entsprechenden Anfrage an eine Source und das Zurückliefern des XML-Codes überließ ich einem Plugin. Diese konnte dann beispielsweise eine Datenbankabfrage machen und das XML selbst erstellen oder aber einfach die Anfrage über CGI an eine Suchmaschine weitergeben, welche selbst für die Generierung des XML sorgt (z.B. über Templates).

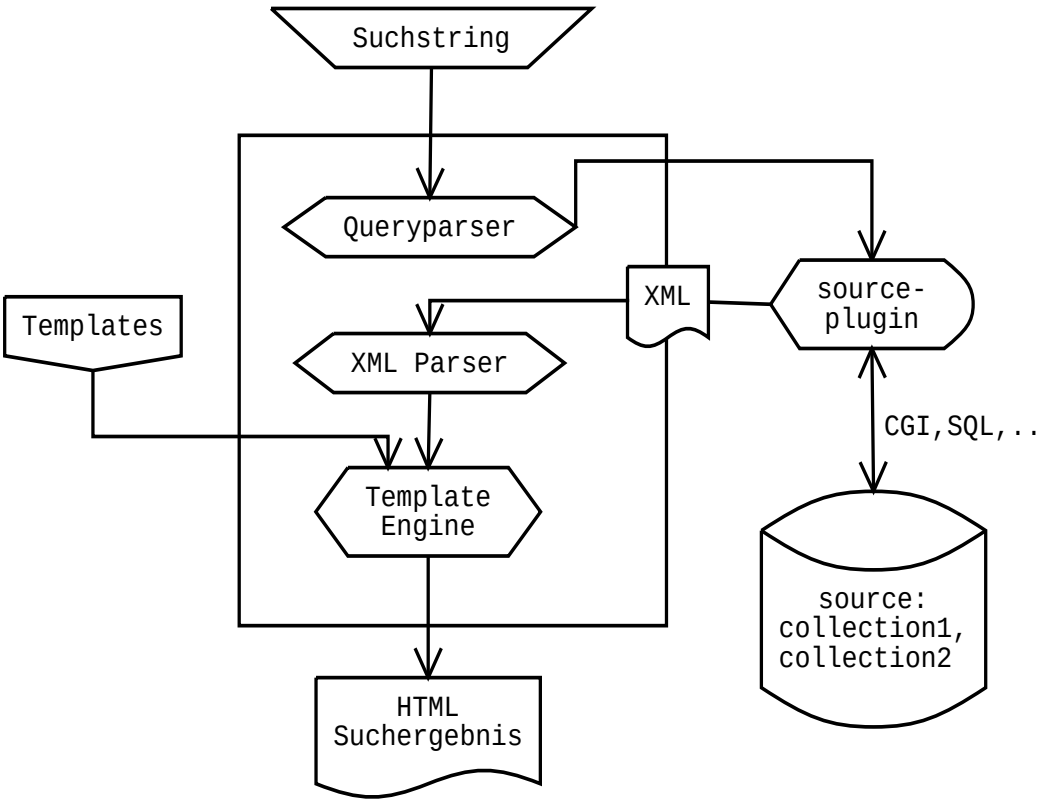

Abbildung 2.2: Frontend Schema

[13] Ich benutze das Wort source für eine Datenquelle (zB. eine Suchmaschine) und collection für einen Bereich innerhalb dieser Quelle (zB. alle koeln.de Seiten)

2.3.5 Templates

Der Einsatz einer Template-Engine in PHP war komplett neu für mich, bei bisherigen Projekten bei koeln.de kamen jedoch *PHPLib*[14] Templates zum Einsatz, weshalb ich sie ebenfalls einsetzte. Ich mußte jedoch feststellen, daß ich mit den von PHPLib gebotenen Funktionen schnell an die Grenzen des Machbaren stieß und sah mich deshalb nach einer Alternative um.
Meine Wahl viel schließlich auf *smarty*[15], welche neben der normalen Platzhalterfunktionalität einige Vorteile bot:

- Eingebaute Funktionen und Modifikatoren (zum Beispiel zur URL-Encodierung)
- Erweiterbarkeit durch Plugins
- Sehr schnell durch Vorkompilieren der Templates (Templates müssen nur einmal geparst werden)

2.3.6 Datenbanksuche

Neben der Suche in den HTML-Seiten, die durch mnoGo-Search abgedeckt wurde, sollten auch einige dynamisch erzeugte Teile der koeln.de Site durchsuchbar gemacht werden. Dazu gehörte der koeln.de-Marktplatz und der Veranstaltungskalender.
Um das aufwändige Spidern aller erzeugten Seiten zu vermeiden, sollte direkt auf Datenbankebene gesucht werden. Dazu wären jedoch sehr aufwendige `LIKE`-Abfragen mit `JOIN`s über mehrere Tabellen nötig gewesen, welche die Performance der ganzen Suche beeinträchtigt und die Belastung des Datenbankservers möglicherweise stark erhöht hätten. Stattdessen legte ich eine zusätzliche Datenbank an, in der die zu durchsuchenden Informationen in flachen Tabellenstrukturen abgelegt wurden. Zur Indizierung der Daten kam der seit der Version 3.23.23 in MySQL enthaltende `FULLTEXT`-Index zum Einsatz, der Volltextsuchen ermöglicht und die gefundenen Ergebnisse mit einem Score bewertet – ideal also für Suchmaschinen. Die Volltext-Datenbank liess sich mit einem einzigen `SELECT` pro Collection füllen. Hier am Beispiel des Marktplatz:

```
INSERT INTO marktplatz
SELECT DISTINCT
       A.nr,
       CONCAT('http://www.koeln.de/marktplatz/firma.php3?v=',A.nr),
```

[14] http://http://phplib.sourceforge.net/
[15] http://www.phpinsider.com/php/code/Smarty/

```
       A.firma,
       CONCAT_WS(' ',A.strasse,A.plz,A.ort,A.stadtteil,A.besch),
       A.changed,
       CONCAT_WS(' ',REPLACE(A.sw,'<A7>',' '),A.langtext),
       A.flag,
       NULL,
       B.nr,
       B.name,
       A.plz,
       NULL
  FROM marktplatz.kunde A,
       marktplatz.branche B,
       marktplatz.br_fa C
 WHERE A.flag > 0
   AND C.firma = A.nr
   AND C.branche = B.nr
;
```

Die Einbindung dieser Datenbanksuche in die globale koeln.de Suche geschah über ein Plugin, welches aus den Ergebnissen der Datenbankabfrage den benötigten XML-Code erzeugte.

2.4 Live-Ticker

Inspiriert durch den Erfolg des WM-Tickers entstand die Idee, zukünftig auch selbst Liveticker anbieten zu können und so aktuelle Geschehnisse besser in die Site einzubinden.
Meine Aufgabe bestand nun darin, eine Möglichkeit für die Redakteure zu schaffen, neue Liveticker zu erstellen und diesen Nachrichten hinzuzufügen.
Auch hier setzten wir wieder eine *MySQL*-Datenbank ein, in der die Tickerdaten gespeichert wurden. Um die Last auf dem Server möglichst gering zu halten, sollten aus den Daten in der Datenbank statische Seiten erzeugt werden. Dabei kam die bereits beim WM-Ticker (Siehe Kapitel 2.2) eingesetze Templatetechnik erneut zum Einsatz.

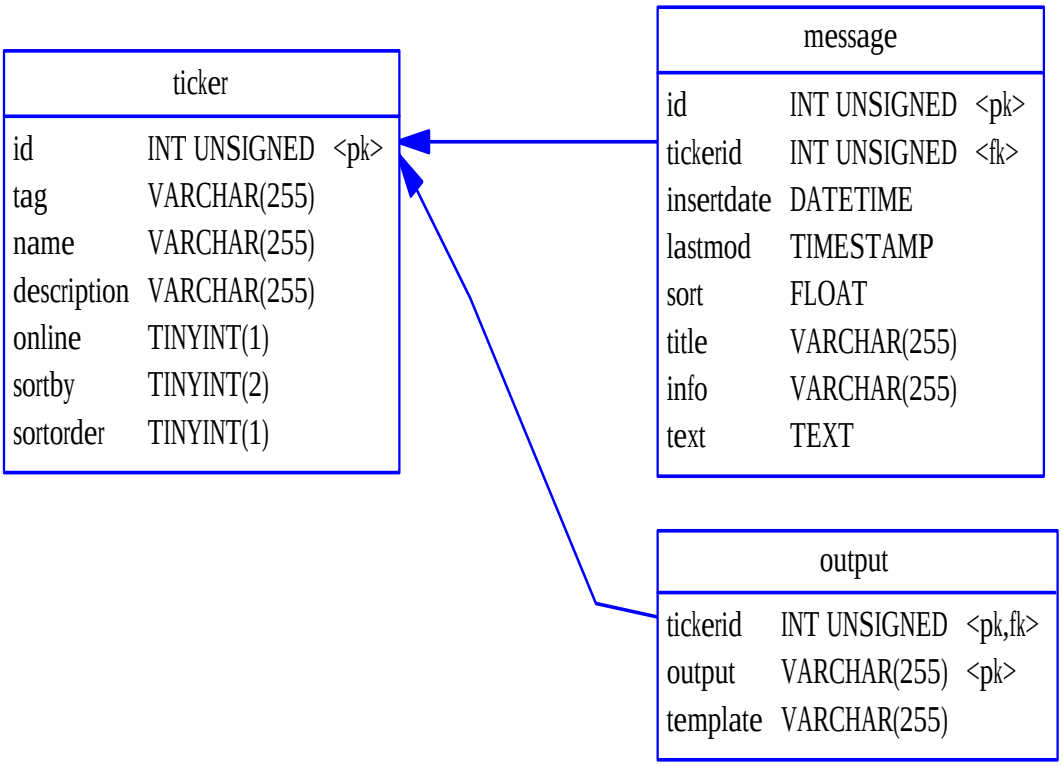

Abbildung 2.3: Datenbank Schema

Kapitel 3

Fazit

Nach diesen knapp dreieinhalb Monaten bei NetCologne kann mein Fazit nicht nur im Hinblick auf die Themengebiete meiner Arbeit nur positiv ausfallen. So konnte ich meine Kenntnisse in *Perl* und *PHP* stark erweitern und bin mit XML in ein mir völlig neues Themengebiet vorgedrungen.
Die Zusammenarbeit mit externen Partnern wie *Altus* oder *Allesklar* hat mir weitere Einblicke in andere Firmen im IT-Bereich eröffnet.
Auch die volle Einbindung in das Team, die prompte Unterstützung und Hilfe meiner Kollegen in schwierigen Situationen und die flexible Zeiteinteilung, welche mir eine eigenständige Projektbearbeitung möglich machten, kann ich nur positiv bewerten.