Technische Universität Wien

Business Informatics Group

Institut für Softwaretechnik und Interaktive Systeme

Entwicklung eines webbasierten Institutsinformationssystems

Separation of Concerns und seine Umsetzung in LAMP

eingereicht von:

Andrea Schauerhuber

Diplomarbeit

zur Erlangung des akademischen Grades

Magistra rerum socialium oeconomicarumque

Magistra der Sozial- und Wirtschaftswissenschaften

(Mag. rer. soc. oec.)

Fakultät für Wirtschaftswissenschaften und Informatik
Universität Wien
Fakultät für Technische Naturwissenschaften und Informatik
Technische Universität Wien

Studienrichtung: Wirtschaftsinformatik

Begutachterin

O. Univ.-Prof. Mag. Dipl.-Ing. Dr. Gerti Kappel

Wien, 29. Oktober 2004

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 29. Oktober 2004

Danksagung

Für ihre finanzielle, moralische und kulinarische Unterstützung während meines Studiums möchte ich mich u. a. bei meinen Eltern Julius und Brigitte Schauerhuber, meinen Geschwistern Julia, Olivia, Lukas, Nora und Laura, meinen Großeltern Wilhelm und Brigitta Althaler, Alosia Schauerhuber, und Maria Kopatz bedanken.

Ein besonderes Dankeschön gilt meinen beiden Kollegen Alexander Heumayer und Nenad Jovanovic für die gute Zusammenarbeit in den letzten drei Jahren, speziell im Rahmen dieser Diplomarbeit.

Kurzfassung

Im Rahmen einer Kooperation dreier laufender Diplomarbeiten (Alexander Heumayer, Nenad Jovanovic, Andrea Schauerhuber) wurde mit Hilfe der Open-Source Web-Development-Plattform LAMP (der Kombination aus dem Betriebssystem Linux, dem Webserver Apache, dem Datenbank-Managementsystem MySQL und der Programmiersprache PHP) ein für die Business Informatics Group (Institut für Softwaretechnik und Interaktive Systeme, Technische Universität Wien) maßgeschneidertes Web-Informationssystem entwickelt. Die Web-Anwendung bietet unterschiedlichen Benutzergruppen, darunter Studierenden und Mitarbeitern des Instituts, verschiedenste Dienste an.

Der Schwerpunkt der vorliegenden Diplomarbeit ist die Entwicklung eines Architektur-Frameworks für LAMP, das den Anforderungen des Separation of Concerns von Web-Informationssystemen gerecht wird. Die Schwäche der traditionellen Entwicklung unter LAMP liegt in der Vermischung von Inhalten, Anwendungslogik und Präsentation. Zum einen erlaubt HTML keinen Rückschluss auf die Semantik der Inhalte von HTML-Tags, wodurch Inhalte und Präsentation nicht voneinander getrennt werden können. Zum anderen sorgt hinzugefügter PHP-Code, der die Anwendungslogik einbringen soll, für zusätzliche Komplexität. Die Hauptanforderung an die zu entwickelnde Architektur war es daher, diese Schwächen zu beseitigen und im Sinne des Diplomarbeitstitels dem Separation of Concerns-Paradigma zu folgen. Die in der Diplomarbeit entwickelte Architektur basiert auf dem XAO Framework (http://xao-php.sourceforge.net), erweitert um die Realisierung des Model View Controller Patterns. Diese Architektur stellt auch die Grundlage für das im praktischen Teil entwickelte Institutsinformationssystem dar.

Abstract

Within the scope of three ongoing diploma theses (Alexander Heumayer, Nenad Jovanovic, Andrea Schauerhuber) a custom-made web information system has been developed for the Business Informatics Group (Institute of Software Technology and Interactive Systems, Vienna University of Technology) using the open source web development platform LAMP (a combination of the operating system Linux, the web server Apache, the database management system MySQL and the programming language PHP). The web application offers various services to different user groups, amongst them students and employees of the institute.

The focus of this thesis is the development of an architectural framework for LAMP, which satisfies the demands of the separation of concerns within web information systems. The obvious weakness of the traditional development with LAMP is that content, application logic and presentation are mixed up and not properly separated. In fact, HTML does not allow to draw conclusions about the semantics of content within HTML tags, which makes it impossible to separate the content from the presentation. Moreover, the added PHP code, which is supposed to implement the application logic, further increases complexity. The contribution of the thesis is the development of an architecture that overcomes these weaknesses and that follows the separation of concerns paradigm. The herein developed architecture is based on the XAO framework (http://xao-php.sourceforge.net) and has been extended by the realization of the mode view controller pattern. This architecture also forms the basis for the information system of the Institute, which has been developed in the practical part of the thesis.

Inhaltsverzeichnis

Kapitel 1

1 Einleitung

Geht man den vielen Erzählungen über die Geschichte des Internets nach, dann ist die eigentliche Geburtsstunde einer der bedeutsamsten Entwicklungen der Informatik mit 1957 zu datieren, dem Start des ersten Satelliten Sputnik (UdSSR) ins All. Dieses Ereignis war Anlass genug für die Gründung der amerikanischen Advanced Research Projects Agency (ARPA), welche 1969 mit dem ARPANET[1] den Grundstein für das weltweite Netzwerk gelegt hat. 20 Jahre lang war das Internet nur für Wissenschafter und Computerexperten, die mit Kollegen an anderen Universitäten oder Forschungseinrichtungen kommunizierten, zugänglich. Der Durchbruch in der Öffentlichkeit gelang erst 1991, als Tim Berners-Lee das WWW, basierend auf dem HTTP-Protokoll, der Auszeichnungssprache HTML und dem „Adressschema“ URL, am europäischen Kernforschungszentrum CERN entwickelte. Mit diesen Technologien lassen sich Informationen mit multimedialen Inhalten ansprechend aufbereitet darstellen. Sie ermöglichen es weiters einem Computer-Laien durch die verteilten, miteinander verlinkten Dokumente in einfacher Weise zu surfen [Stan99, Zako04].

1 Ein Netzwerk bestehend aus vier Knoten: University of California Los Angeles (UCLA), Stanford Research Institute (SRI), University of California Santa Barbara (UCSB), and the University of Utah in Salt Lake City.

1.1 Motivation

Das WWW hat sich jedoch über diesen auf einzelne, statische Dokumente bezogenen Ansatz hinaus weiterentwickelt. Geschäftswelt, Behörden und Privatpersonen haben das Internet für sich entdeckt und nutzen gelernt. Menschen gehen bei Versandhäusern auf virtuellen Einkaufsbummel und suchen Informationen aller Art [Stan99]. Neben den ursprünglichen, statischen Websites haben sich im Laufe der Zeit komplexere, in die Richtung interaktiver und datenintensiver Software-Anwendungen gehende, Web-Architekturtypen herausgebildet. Heute funktionieren manche Geschäfte wie Ebay und Amazon sogar ausschließlich über den Vertriebskanal Internet. Die Architekturen dieser Web-Anwendungen müssen hohe Anforderungen erfüllen. Sie übernehmen Authentifizierungs-, Autorisierungs- und andere Sicherheitsmaßnahmen [Jova04], kümmern sich um Aspekte wie Lastausgleich und Caching, bieten Zugang zu Alt-Systemen über ein Web-Interface u. v. m. Dabei fällt die Präsentation von Inhalten auf verschiedensten Endgeräten (PCs, Handys, etc.) in die Kategorie jüngerer Entwicklungen [Heum04].

Web-Anwendungen und besonders deren Benutzerschnittstellen sind im Gegensatz zu gewöhnlichen Software-Anwendungen häufigen Änderungen unterworfen. Die Architektur einer Web-Anwendung muss daher besonders im Hinblick auf Erweiterbarkeit und Wartbarkeit entworfen werden. Im Zuge dieser Anforderung wird in Fachkreisen gerne von „Separation of Concerns“ gesprochen. Hinter diesem Begriff steckt der Gedanke, dass Inhalte bzw. Daten, Präsentation der Inhalte und die Anwendungslogik einer Anwendung meist von verschiedenen Personen oder Teams produziert und gewartet werden. Indem man versucht, diese drei Verantwortungsbereiche von einander zu trennen, soll es möglich sein, dass die erwähnten Personen bzw. Teams unabhängig voneinander an einem gemeinsamen Projekt arbeiten können [Mazz04]. Dies fließt nicht nur in die hier angesprochene Wartbarkeit, sondern auch speziell in die Erweiterbarkeit des Systems ein. Sofern die Inhalte unabhängig von einem Darstellungsformat vorhanden sind, ist das Einfügen zusätzlicher Präsentationskanäle (neue Endgeräte) oder die Abbildung von Informationen auf verschiedene Benutzersichten relativ problemlos durchzuführen.

1.2 Problemstellung und Ziel

Das Problematik in der herkömmlichen Web-Entwicklung liegt bei HTML selbst. HTML vermischt Information mit deren Präsentation. Hinzu kommen Server- oder Client-Skriptsprachen, um die Anwendungslogik der Anwendung einzubringen. Das Ergebnis ist ein kaum lesbares, geschweige denn wartbares und erweiterbares Code-Gemisch aus HTML- und Logik-Fragmenten. Schon im Diplomarbeitstitel enthalten ist die Vorgabe für die Web-Entwicklung im Rahmen dieser Arbeit, nämlich als Ausgangsbasis die Open-Source Web-Developement-Platform LAMP (Linux, Apache, MySQL, PHP) zu benutzen. Existierende Entwicklungsumgebungen bzw. –frameworks auf Basis von LAMP unterstützen allerdings Separation of Concerns nicht oder nur unzureichend.

Ziel war es daher ein Framework oder einen Framework-basierten Ansatz auf Basis von LAMP zu entwickeln, der das Prinzip von Separation of Concerns unterstützt. Das Resultat der praktischen Umsetzung in Form eines sog. „Proof of Concept"-Prototypen ist ein webbasiertes Institutsinformationssystem für die Arbeitsgruppe BIG (Institut für Softwaretechnik und Interaktive Systeme, Technische Universität Wien). Die Implementierung dieses Institutsinformationssystems geschah in Zusammenarbeit mit Alexander Heumayer [Heum04] und Nenad Jovanovic [Jova04].

Kapitel 2 und Kapitel 3 machen den theoretischen Teil der Arbeit aus. In Kapitel 2 wird das Thema Web-Architekturen behandelt und besonders auf die Schichten-Architekturen im Web-Kontext eingegangen. Die Begriffe Pattern und Framework sind Gegenstand des Kapitels 3, welches außerdem die in dieser Arbeit verwendeten Patterns bzw. Alternativen vorstellt. Kapitel 4 enthält eine Einführung in LAMP und geht speziell auf die Fragestellung ein, wie mit PHP in Hinblick auf die Trennung der Verantwortlichkeiten entwickelt werden kann. Die gewählte Architektur und die implementierten Klassen des SAN Frameworks werden in Kapitel 5 vorgestellt. Kapitel 5 beinhaltet außerdem das neu entwickelte Input Validator Pattern, ein Pattern für zentrale und konsistente Überprüfungen von Benutzereingaben innerhalb einer Web-Anwendung. Das gemeinsam mit Alexander Heumayer und Nenad Jovanovic verfasste Kapitel 6 stellt die Funktionalitäten des implementierten Institutsinformati-

onssystems vor. Kapitel 7 fasst die Erkenntnisse und Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf die zukünftige Web-Entwicklung mit LAMP.

Kapitel 2

2 Architekturen

Web-Anwendungen bzw. Web-Informationssysteme haben sich aus dokumentenzentrierten Websites entwickelt [Eich04]. Im wesentlichen erweitern sie ihre eigentlichen Vorläufer um die Fähigkeit auf Benutzereingaben dynamisch zu reagieren und den Zustand der Anwendung am Server zu verändern [Cona00]. Diese Weiterentwicklung konfrontiert uns mit vollwertigen Softwareanwendungen, die interaktive, datenintensive und individualisierbare Dienste über das Internet auf verschiedenen Endgeräten zur Verfügung stellen [KRP+03].

Die Architektur solcher komplexer Web-Anwendungen ist dabei ein maßgeblicher Faktor für Qualität und Erfolg des Projekts. Neben funktionalen und nicht funktionalen Anforderungen gilt es auf die technischen Randbedingungen, wie Einbindung von Legacy-Systemen, Rücksicht zu nehmen [Eich04].

In diesem Kapitel werden vor allem die Schichten-Architekturen allgemein und besonders im Hinblick auf heutige Web-Anwendungen näher betrachtet. In [DuGH03] sind darüber hinaus Ausführungen zu anderen verteilten Architekturen, darunter Web-Service-Architekturen und Peer-to-Peer-Architekturen zu finden.

2.1 Was ist eine Architektur?

Der Begriff Architektur wird in der Fachliteratur gerne und häufig definiert [Eich04, DuGH03]. Eine interessante und auch amüsante Herangehensweise an den Terminus technicus findet sich in [Fowl03a] und [Fowl03b]. So kehren in den meisten Begriffsdefinitionen zwei Elemente wieder:

Es handle sich bei einer Architektur einerseits um die Darstellung der Systemteile auf höchster Abstraktionsebene und andererseits um wichtige Entscheidungen, die zu einem späteren Zeitpunkt schwer änderbar sind und daher sehr sorgfältig getroffen werden müssen.

Welche Komponenten und welche Entscheidungen für die Architektur signifikant sind, ist allerdings subjektiv und hängt vom jeweiligen Entwicklerteam ab. Der Begriff Architektur ist demnach auch Mittel, um der hohen Relevanz einer Sache Ausdruck zu verleihen [Fowl03a, Fowl03b].

2.2 Schichten-Architekturen

Web-Anwendungen sind im wesentlichen verteilte Systeme. Unter einem verteilten System ist, eine Menge autonomer Computer zu verstehen, welche sich den Benutzern als ein einziges, kohärentes System präsentieren [TaSt02]. Um mit der Komplexität eines verteilten Systems besser umgehen zu können, wird gerne in dem Begriffspaar Client-Server gedacht.

Im klassischen Client-Server Modell werden zwei Prozessgruppen unterschieden [TaSt02]:

- Ein Server ist ein Prozess, der ein bestimmtes Service zur Verfügung stellt (Dateisystem-Service, Datenbank-Service, etc.).
- Ein Client ist ein Prozess, der ein Service eines Servers in Anspruch nehmen möchte und eine Anfrage an diesen schickt (vgl. Abbildung 2-1).

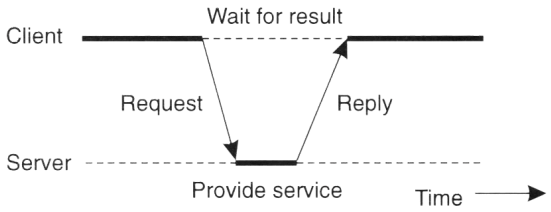

Abbildung 2-1: Request-Response-Verhalten zwischen Client und Server [TaSt02]

Der Ausdruck Client-Server, für über ein Netzwerk verbundene PCs, wurde schon verwendet, bevor die Client-Server-Architektur in den späten 80ern populär wurde. Diese ersten Netzwerke basierten auf der File-Sharing-Architektur, in denen der Server die Aufgabe hatte, Dateien aus dem gemeinsamen Dateisystem bzw. Speicherbereich auf einen bestimmten Desktop zu übertragen. Aufgrund der geringen Kapazitäten, es können gerade ein Dutzend Personen gleichzeitig auf eine Datei zugreifen, und dem Aufkommen der Graphischen Benutzerschnittstellen (GUI) wurden File-Sharing-Systeme von der klassischen 2-Schichten-Architektur abgelöst [Schu95].

2.2.1 2-Schichten-Architektur

Der Begriff 2-Schichten-Architektur stammt aus einer Zeit, als Clients direkt mit Datenbankserver kommuniziert haben [AnRo01]. Anstatt ganze Dateien zu übertragen, wurde auf die, über RPC oder SQL gestellte, Anfrage direkt geantwortet. Die 2-Schichten-Architektur erweist sich dadurch als besser skalierbar. Sie unterstützt bis zu 100 Benutzer [Schu95].

Neben der Betrachtung von physischen Schichten unterscheidet man in Client-Server-Systeme auch logische Schichten der Anwendung. Diese sind:

1. die Benutzerschnittstelle bzw. die Präsentation der Informationen,
2. die Anwendungslogik und
3. die Daten der Anwendung.

Die Benutzerschnittstelle enthält üblicherweise alles, um direkt mit den Benutzern interagieren zu können (Eingabefelder, etc.). Die Anwendungslogik stellt die tatsächliche Anwendung dar und die Datenebene beinhaltet im wesentlichen nur die zu bearbeitenden Daten [TaSt02]. Physisch betrachtet, liegt in der 2-Schichten-Architektur

die gesamte Benutzerschnittstelle auf Clientseite während die Daten sich auf Serverseite befinden. Die Anwendungslogik hingegen wird zwischen Client und Server aufgeteilt [Sado97]. Möglichen Arten dieser Aufteilungen werden in Abbildung 2-2 dargestellt.

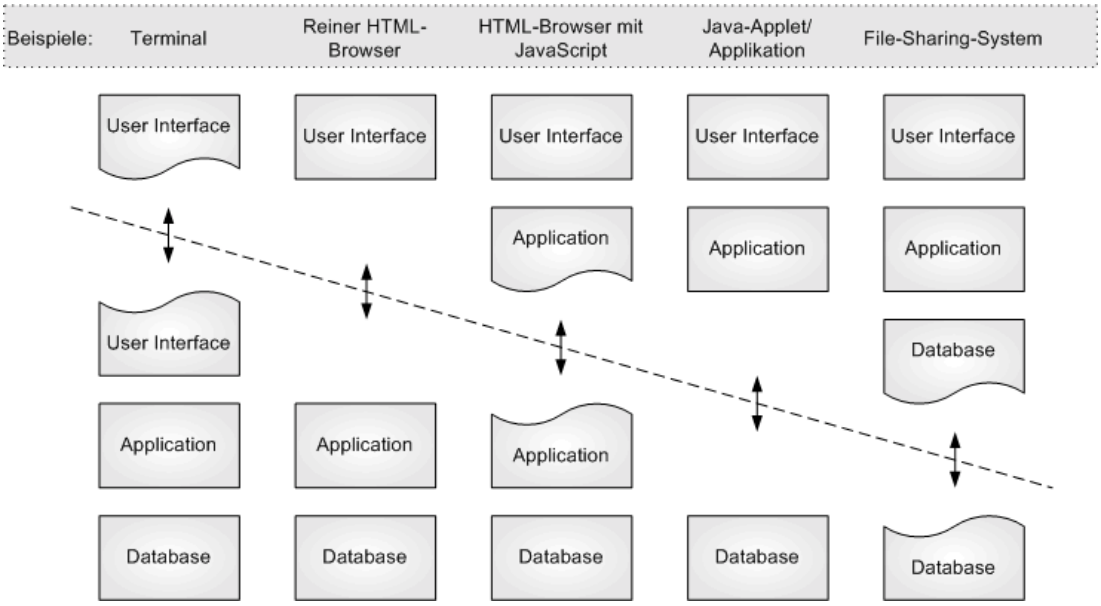

Abbildung 2-2: Alternative Client-Server-Organisationen (in Anlehnung an [TaSt02])

Im Web-Kontext nehmen 2-Schichten-Architekturen eine andere Form an. In [KRP+03] wird eine 5-teilige Klassifikation von Web-Informationssystemen getroffen. Die Varianten eins und zwei gehören dabei nach [AnRo01] der 2-Schichten-Architektur an.

Variante 1: Statische Web-Informationssysteme

Abbildung 2-3: Architektur eines statischen Web-IS [KRP+03]

Statische Websites stellen die eigentlichen Vorläufer von Web-Anwendungen dar. Die dem Benutzer präsentierten Webseiten liegen in statischer Form als HTML-Dateien im Dateisystem des Servers vor. Die Kommunikation zwischen Webclient, einem Browser, und Webserver erfolgt über das HTTP-Protokoll. Über die URL

wird der Pfad für die gewünschte Datei am Webserver ermittelt und deren Inhalt über HTTP als Antwort an den Webclient zurückgeschickt (vgl. Abbildung 2-3).

Obwohl statische Websites die älteste Generation von Web-Anwendungen darstellen, werden sie nach wie vor eingesetzt. Der hauptsächliche Vorteil dieser Architektur liegt in der Einfachheit und Stabilität so wie den niedrigen Antwortzeiten. Nachteilig ist die Vermischung von Inhalten bzw. Daten einer Web-Seite mit deren Präsentation zu sehen. Aktualisierungsarbeiten stellen sich als sehr aufwendig heraus, wenn die Website sehr änderungsintensiv ist. Werden Informationen im Sinne der Usability an mehreren Stellen der Website zur Verfügung gestellt, können durch Aktualisierungen leicht Inkonsistenzen entstehen [KRP+03].

Variante 2: Web-Informationssysteme mit DBS-Unterstützung

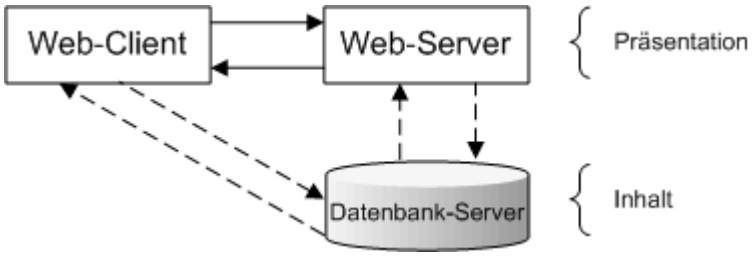

Abbildung 2-4: Architektur eines Web-IS mit DBS-Unterstützung [KRP+03]

Je größer die Anzahl der Web-Seiten und je umfangreicher der Datenbestand einer Web-Anwendung wird, desto eher muss man die Verwendung eines Datenbanksystems zur effizienten und konsistenten Verwaltung dieser Daten in Betracht ziehen (vgl. Abbildung 2-4). Der Einsatz eines Datenbanksystems trägt zur Trennung der in HTML vermischten Inhalts- und Präsentationsaspekte bei und vereinfacht Aktualisierungsaufgaben. Änderungen am Datenbestand können auch dezentral über die Benutzerschnittstelle (HTML-Formulare und CGI, Java Applets,...) vorgenommen werden.

Als Nachteil dieser Architektur gegenüber dem statischen Web-Informationssystem ist die hinzukommende Anwendungslogik zu sehen. Sollen in der Web-Anwendung die Datenbestände lediglich dargestellt werden, so muss zumindest die Logik für die Datenbankabfragen in die Anwendung eingebaut werden. Die Aktualisierung der

Daten könnte zentral über ein existierendes Front-end für ein Datenbanksystem durchgeführt werden. Handelt sich aber um eine interaktive Anwendung, bei welcher die Benutzer den Datenbestand dezentral verändern und erweitern können, dann muss zusätzliche Logik für u. a. die Eingabeüberprüfung, die Mehrbenutzerkontroller und die Datenbankabfragen zum Einfügen eines Eintrags implementiert werden.

2.2.2 N-Schichten-Architektur

3-Schichten-Architekturen tauchen zum ersten Mal in 90igern auf. Die klassischen 2-Schichten-Architekturen können die steigende Benutzerzahl nicht mehr handhaben. Der Hauptgrund dafür ist der Verbindungsprozess zwischen Client und Datenbankserver, denn für jeden Client wird eine offene Verbindung gehalten. Auch wenn der Client im Moment gerade inaktiv ist, werden Keep-Alive-Nachrichten ausgetauscht. Die Antwort auf dieses Problem ist eine weitere Schicht, welche zwischen Client und Datenbankserver positioniert wird. Die Anwendungslogik die vorher über die Benutzerschnittstelle und die Datenbank hinweg verteilt wurde, liegt nun in einer eigenen, nicht notwendigerweise physisch gekapselten, Komponente [Schu95].

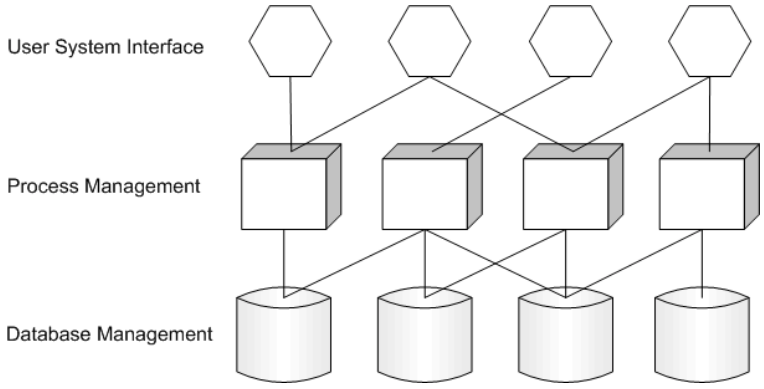

Abbildung 2-5: 3-Schichten-Architektur [SaCo00]

Der wesentliche Unterschied zwischen N-Schichten-Architekturen und 2-Schichten-Architekturen ist die Komponentenorientierung, welche die Modularität, die Wiederverwendbarkeit, die Flexibilität und Skalierbarkeit anspruchsvoller Web-Anwendungen fördert [AnRo01].

Im Kontext von Web-Anwendungen führte die Problematik der steigenden Benutzerzahlen und mangelnde Performanz zur Auslagerung der Anwendungslogik auf einen oder mehrere Applikationsserver.

Die drei letzten Architekturtypen, Varianten 3 bis 5, für Web-Informationssysteme aus der 5-teiligen Klassifikation von [KRP+03] lassen sich in den Bereich der mehrschichtigen Architekturen einordnen.

Variante 3: Applikationsorientierte Web-Informationssysteme

Folgende Abbildung 2-6 zeigt den Einsatz eines Applikationsservers.

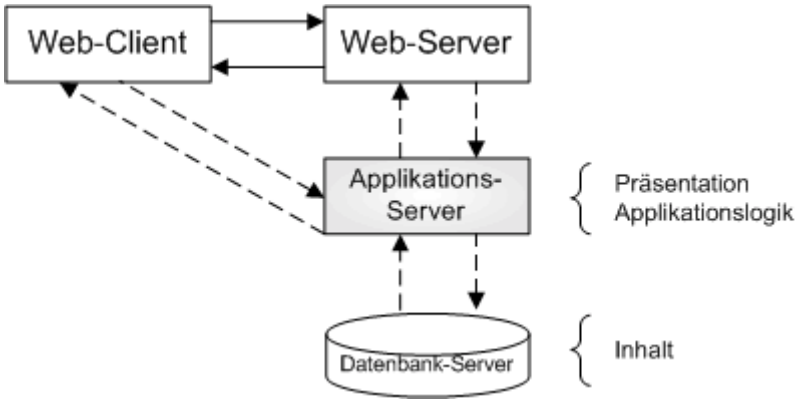

Abbildung 2-6: Architektur eines applikationsorientierten Web-IS [KRP+03]

Die Aufgabe des Webservers reduziert sich auf die Weiterleitung von Anfragen des Clients an den Applikationsserver und umgekehrt auf das Zurückschicken der Ergebnisse an den Client. Der Applikationsserver ist nicht nur für den Aufbau der HTML-Seite verantwortlich, sondern kümmert sich darüber hinaus um Anbindung an Datenbank- und Legacy-Systeme, Transaktionsmanagement, Sicherheitsvorkehrungen, Lastausgleich und Caching. Der Einsatz dieser Architektur macht vor allem dann Sinn, wenn mit einer großen Anzahl gleichzeitig zugreifender Clients zu rechnen und die Anwendungslogik sehr komplex ist [KRP+03].

Variante 4: Ubiquitäre Web-Informationssysteme

„Ein ubiquitäres Web-Informationssystem stellt personalisierte Dienste zu jeder Zeit an jedem Ort und für jedes sinnvoll nutzbare Medium zur Verfügung, womit ein allgegenwärtiger Zugriff möglich wird. Dabei ist das oberste Ziel, den richtigen Dienst zum richtigen Zeitpunkt am richtigen Ort in der richtigen Form anzubieten.“ [KRP+03]

Eine Beispielanwendung für einen lokationsbasierten Dienst könnte einer Person die sich in der nahen Umgebung befindlichen Restaurants anzeigen. Diese Beispielanwendung stellt darüber hinaus einen personalisierten Dienst zur Verfügung, wenn nur ein bestimmter Typ von Restaurant angezeigt wird, weil die Anwendung die Vorlieben des Benutzers kennt.

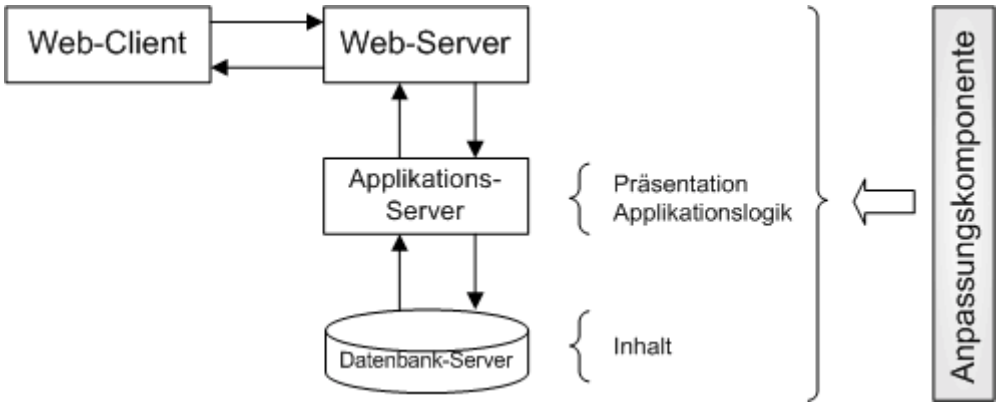

Abbildung 2-7: Architektur eines ubiquitären Web-IS [KRP+03]

Die Überwachung des Kontexts, in dem sich der Benutzer befindet, und die Anpassung der Inhalte und Präsentation werden von einer sog. Anpassungskomponente übernommen (vgl. Abbildung 2-7). Diese arbeitet mit Hilfe eines Regelwerks (Ereignis – Bedingung - Aktion). Die Notwendigkeit der Separation of Concerns wird hier besonders deutlich, denn sowohl die Inhalte als auch die Präsentation müssen unabhängig voneinander an die Gegebenheiten (Dienst, Ort, Zeit und Form) angepasst werden können.

Variante 5: Portal-orientierte Web-Informationssysteme

Mit Hilfe von Web-Portalen wird versucht verschiedene Dienste dem Benutzer in aggregierter Form zur Verfügung zu stellen (vgl. Abbildung 2-8). Der Endbenutzer kann aus den angebotenen Portletimplementierungen eines Portal-Servers auswählen und diese zu einem Portal zusammenfügen.

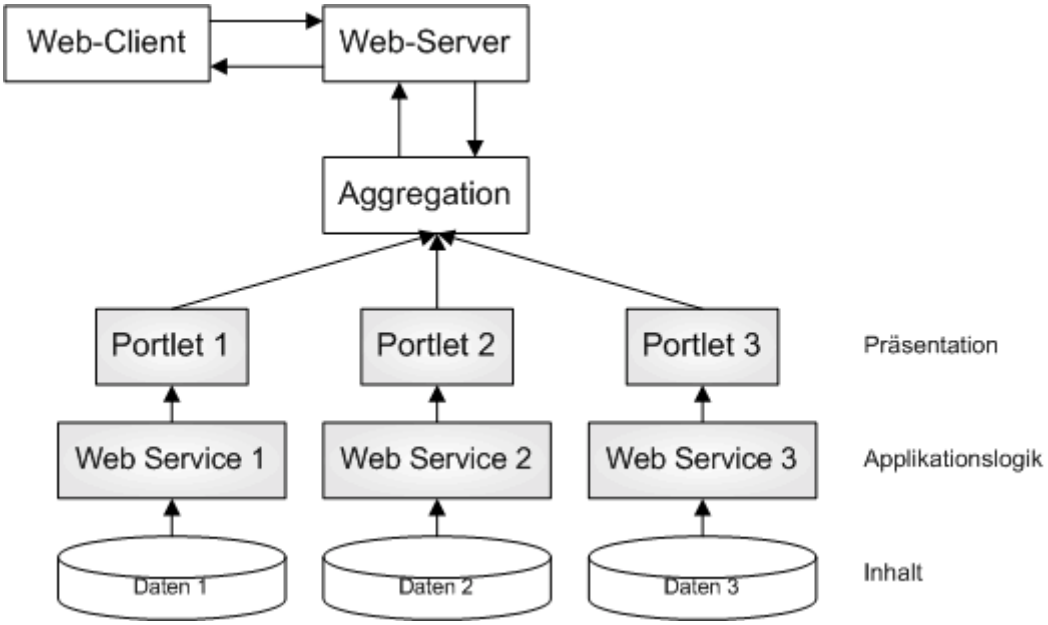

Abbildung 2-8: Architektur eines Portal-basierten Web-IS [KRP+03]

Um auch so genannte „remote“ Portlets integrieren zu können, wird auf die Web-Service-Technologie zurückgegriffen. Für eine detaillierte Betrachtung wird auf [KRP+03] verwiesen.

2.3 Architekturen für Web-Anwendungen

Eine weitere Klassifikation für Web-Informationssysteme wurde [Cona00] entnommen. Diese Einteilung findet auf weit höherer Abstraktionsebene als die vorgestellten Varianten in Unterkapitel 2.2 statt. In [Cona00] werden drei Architekturtypen vorgeschlagen, die im Folgenden diskutiert werden, nämlich

1. Thin Web Client
2. Thick Web Client
3. Web Delivery

2.3.1 Thin Web Client

Diese Architektur findet hauptsächlich dann Anwendung, wenn über die Konfiguration des Clients keine Annahmen getroffen werden können. Die meisten E-Commerce-Anwendungen basieren auf dieser Architektur. Es wird versucht einen möglichst großen Kundenkreis zu erreichen. Einen Teil der Kunden zu vergrämen, weil deren Browserkonfiguration den technischen Anforderungen der Web-Anwendung nicht entspricht, wirkt sich negativ auf den finanziellen Erfolg aus.

Der Thin-Web-Client-Architekturtyp könnte als die minimale Architektur für Web-Anwendungen verstanden werden (die Clientseite betreffend). Je nach Komplexität der Anwendung steigt die Anzahl der Komponenten auf der Serverseite. Die wesentlichen Komponenten werden im weiteren kurz erläutert. Eine grafische Darstellung des Architekturtyps ist in Abbildung 2-9 zu finden.

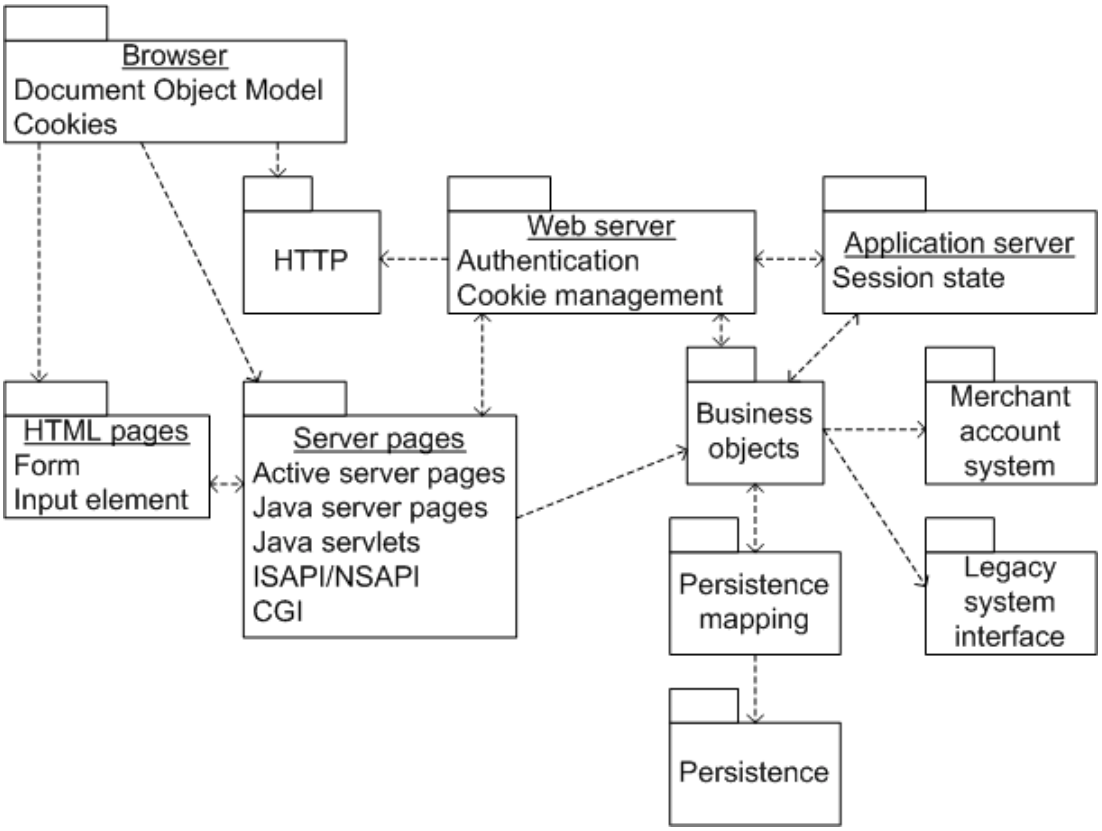

Abbildung 2-9: Thin Web Client Architektur [Cona00]

Der Client (Browser) stellt die Benutzerschnittstelle dar, der die HTML-Seiten interpretiert und abgesehen vom Empfangen und Senden von Cookies keine weiteren Aufgaben mehr übernimmt.

Einziger Anlaufpunkt aller Anfragen durch die Browser ist der Webserver. Dieser kann möglicherweise andere Prozesse anstoßen (CGI,...) und leitet eine Anfrage dementsprechend an einen Interpreter, ein ausführbares Modul oder an einen Applikationsserver weiter. Das Ergebnis der Abarbeitung auf Serverseite ist in jedem Fall reines HTML, egal ob statisch oder dynamisch erzeugt.

Die Kommunikation zwischen Client und Server erfolgt über die zustandslosen HTTP oder HTTPS[2] Protokolle.

[2] Die übertragenen Daten werden mit SSL verschlüsselt.

2.3.2 Thick Web Client

Das wesentliche Merkmal dieser Architektur ist die Ausführung von Logik auf Clientseite. Die Thick-Web-Client-Architektur ist dann sinnvoll, wenn bestimmte Konfigurationen der Browser bekannt sind, wie dies oft für ein Intranet zutrifft, oder eine aufwendigere Benutzerschnittstelle benötigt wird.

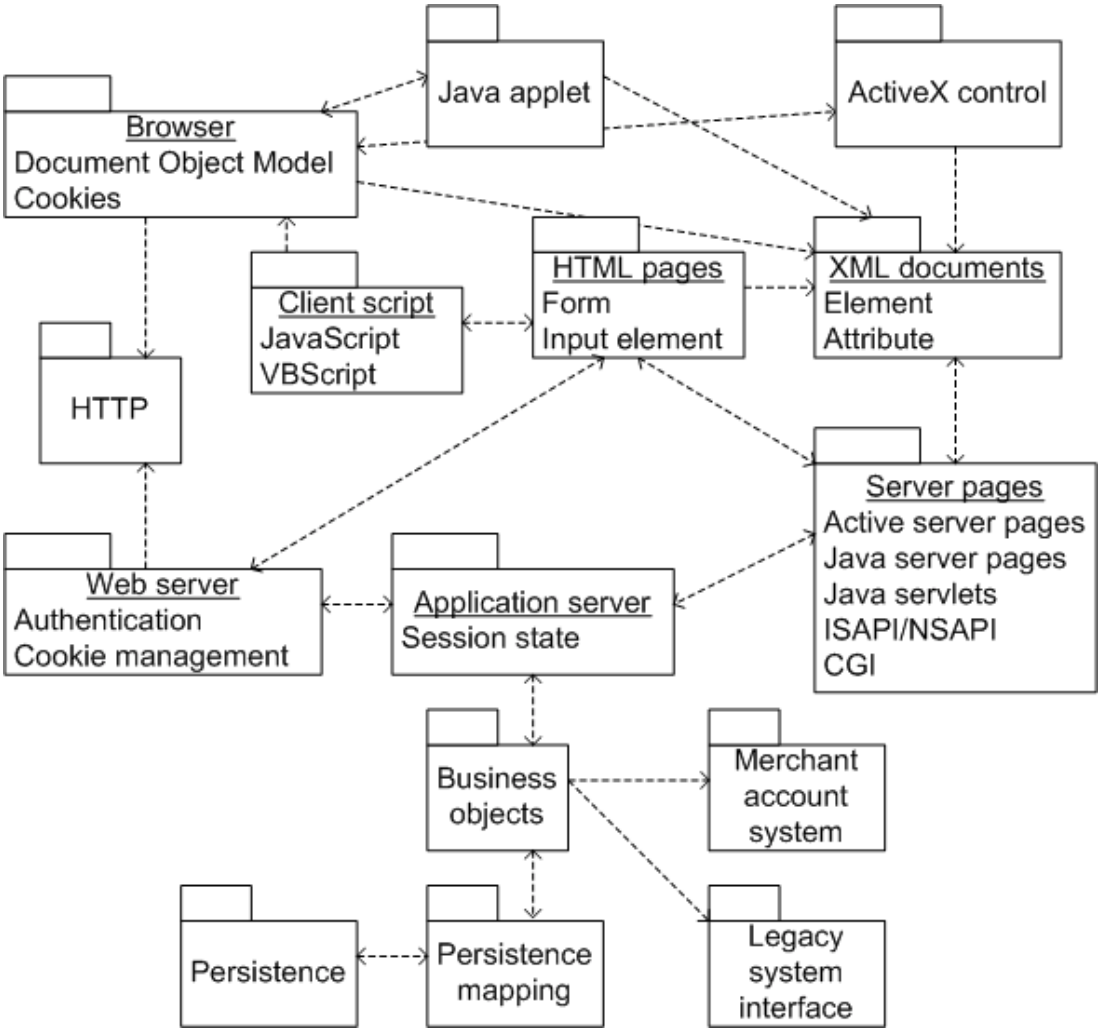

Abbildung 2-10: Thick-Web-Client-Architektur [Cona00]

Sie kann als Erweiterung der Thin-Web-Client-Architektur gesehen werden. Im Folgenden werden einige der zusätzlichen Komponenten kurz vorgestellt:

In die Kategorie Client Script fallen JavaScript und VBScript. Diese Skripts können in die HTML Seite eingebettet sein oder innerhalb separater Dateien übertragen werden. In beiden Fällen werden sie vom Browser ausgeführt.

Ein Java-Applet ist eine in sich geschlossene, kompilierte Komponente, welche zum Client übertragen und ebenfalls vom Browser ausgeführt wird.

ActiveX Kontrollobjekte sind mit Java-Applets vergleichbar. Im Gegensatz zu Java-Applets können sie in verschiedenen Programmiersprachen geschrieben sein. Sie sind nicht plattformunabhängig und können nur auf Windows-Plattformen ausgeführt werden [KRP+03].

Wichtig für diesen Architekturtyp ist, dass auch hier nur über HTTP kommuniziert wird. Für die oben erwähnten Technologien bedeutet das, dass sie nur mit Objekten der Clientseite interagieren können. Direkte Kommunikation mit serverseitigen Objekten bleibt der letzten in [Cona00] vorgestellten Architektur vorbehalten (vgl. nächster Abschnitt).

2.3.3 Web Delivery

Im Unterschied zu den beiden zuvor vorgestellten Architekturen werden in der Web-Delivery-Architektur zusätzliche Protokolle zu HTTP, wie z. B. IIOP, RMI oder DCOM, für die Client-Server-Kommunikation eingesetzt (vgl. Abbildung 2-11). Der Architekturname rührt von der Tatsache, dass das Web als Zustellmedium für ein traditionelles, verteiltes, objektorientiertes Client-Server System dient. Dabei stellt die Integration von existierenden Business-Objekten in den Kontext einer Web-Anwendung die wesentliche Stärke der Web-Delivery-Architektur dar.

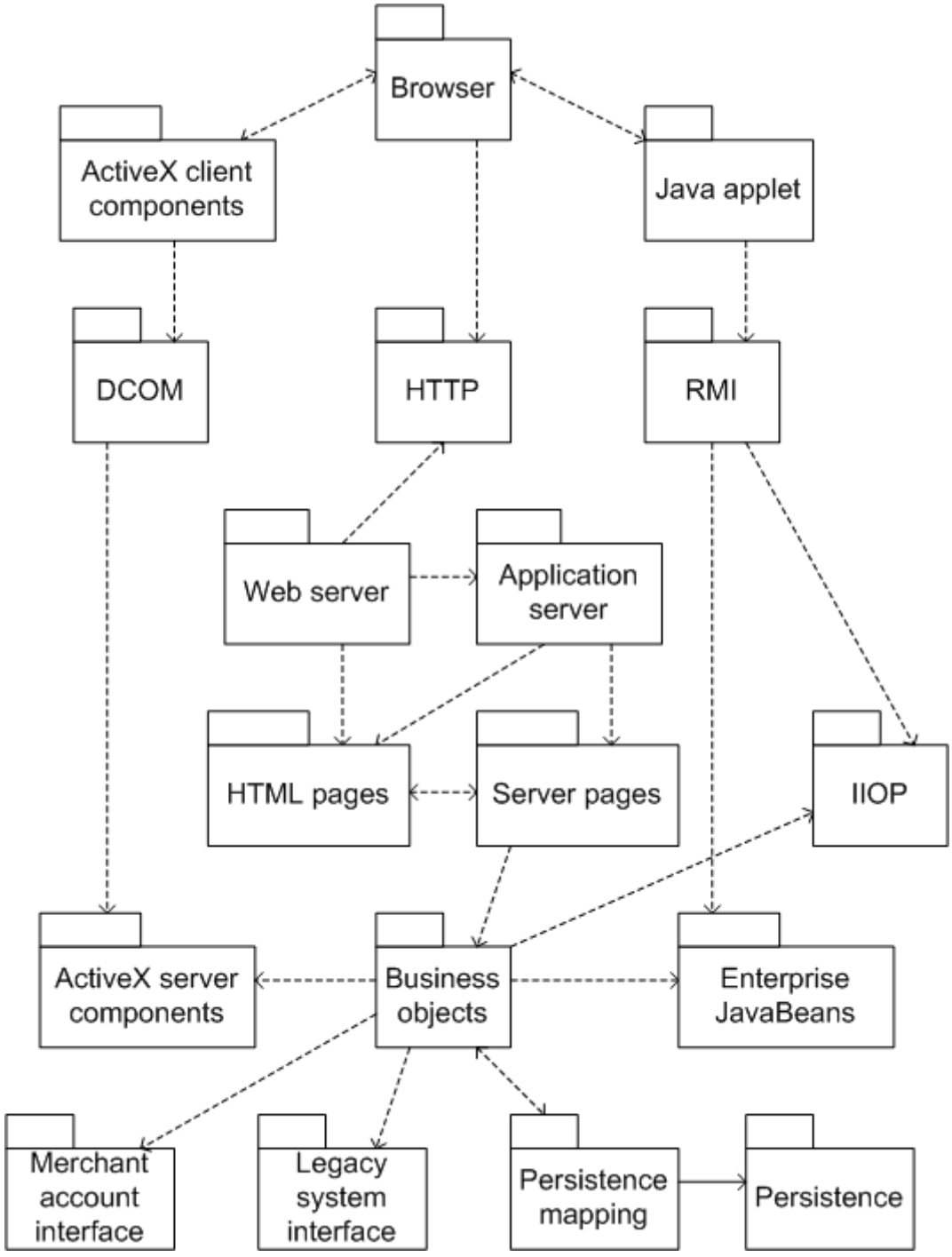

Abbildung 2-11: Web-Delivery-Architektur [Cona00]

Für den Einsatz dieser Architektur wird eine signifikante Kontrolle über Client- und Netzwerkkonfigurationen vorausgesetzt. Durch die direkte und persistente Kommunikation über eines der oben genannten Protokolle kann der Client einen beträchtlichen Teil der Anwendungslogik übernehmen. Für eine genauere Darstellung dieser Architektur wird auf [Cona00] verwiesen.

Kapitel 3

3 Patterns

Ein Muster (im Folgenden Pattern) für eine Software-Architektur beschreibt ein spezielles, wiederkehrendes Problem, dass in spezifischen Kontexten auftritt, und präsentiert ein bewährtes, allgemeines Schema für den Lösungsweg. Dieses Lösungsschema enthält die beteiligten Komponenten, beschreibt deren Verantwortlichkeiten und Beziehungen, und erklärt in welcher Weise sie miteinander interagieren. [BMR+01].

Das Konzept der Patterns wurde aus dem Bereich der Architektur und Planung übernommen. Christopher Alexander, Architekt und Professor für Architektur (Direktor des Center for Environmental Structure in Berkeley, California), definierte die bereits angesprochene Kontext-Problem-Lösung-Struktur, auch als die 'Alexander Form' bekannt. Zu den Pionieren betreffend Patterns in der Softwareentwicklung zählen Ward Cunningham und Kent Beck. Sie übernahmen die Pattern-Idee von Alexander. Die erste publizierte Arbeit über die Verwendung von Patterns in der Softwareentwicklung, ist die 1991 fertig gestellte, in deutscher Sprache verfasste Dissertation von Erich Gamma. Er ist einer der vier Autoren (auch bekannt als 'Gang-of-Four'), die 1994 den mittlerweile Klassiker „Design Patterns – Elements of Reusable Object-Oriented Software“ verfasst haben. Das Buch enthielt den ersten Design-Pattern-Katalog für objektorientierte Software. Davon wurde die Hälfte der enthaltenen Patterns bereits in der Dissertation von Gamma beschrieben [BMR+01].

Unterkapitel 3.1 und 3.2 gehen auf die Begriffe Pattern und Framework ein. Wie Patterns kategorisiert werden können, ist Thema von. Unterkapitel 3.3. Abschließend werden in den Unterkapiteln 3.4 und 3.5 die für den praktischen Teil dieser Arbeit benötigten und alternativen Patterns vorgestellt.

3.1 Was ist ein Pattern?

Patterns werden nicht künstlich erfunden, sondern ergeben sich aus der Arbeit, den Fehlern und Erfolgen von EntwicklerInnen in tatsächlichen Projekten. Hat sich einmal eine gute Lösung gefunden, werden die ExpertInnen auf dieses Erfahrungswissen in einer anderen, ähnlichen Problemsituation zurückgreifen. Das Festhalten bzw. Dokumentieren dieses Wissens in Form von Patterns, ermöglicht es aber auch unerfahrenen EntwicklerInnen existierende und vor allem bewährte Lösungen effektiv nutzen zu können.

In ihrer Definition eines Patterns lehnen sich [GHJV04] an die Definition Alexanders an:

„Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass sich diese Lösung beliebig oft anwenden lässt, ohne sie jemals ein zweites Mal gleich auszuführen.“ [AIS+77]

Patterns bieten demnach eine Lösung für ein wiederkehrendes Problem in einem bestimmten Kontext. Nach [GHJV04] umfasst ein Pattern vier wesentliche Teile:

Patternname: Der Patternname ist ein Stichwort bzw. eine Kurzbeschreibung des Problems, seiner Lösung und seinen Auswirkungen in wenigen Worten. Er soll helfen, ein gemeinsames Verständnis über ein Design zwischen KollegInnen zu finden und die Kommunikation untereinander zu erleichtern. Patternnamen stellen ein allgemeines Vokabular und Verständnis von Design-Prinzipien dar und können im bestmöglichen Fall Teil einer verbreiteten Design-Sprache werden.

Problem: Diese Komponente beschreibt die Problemsituation und in welchem Kontext das Pattern eingesetzt wird.

Lösung: Die Lösung enthält keinen konkreten Entwurf, sondern eine Vorgabe für die Struktur. Der Lösungsweg skizziert die beteiligten Objekte, deren Beziehungen, Verantwortlichkeiten und Kollaborationen.

Konsequenzen: Die Auswirkungen eines Patterns werden durch Auflistung der Vor- und Nachteile des resultierenden Ergebnisses dargestellt. Es ist dadurch einerseits möglich ein Verständnis für Konsequenzen (z. B. Trade-Offs, Kosten, etc.) zu entwickeln und andererseits Patternalternativen zu vergleichen.

3.2 Was ist ein Framework?

Eine andere Art der Wiederverwendung von bestehendem Design- bzw. Architekturwissen geschieht in Form von Frameworks. Ein Framework gibt die grundlegende Architektur und Basisfunktionalität für eine bestimmte Klasse von Softwaresystemen vor. Typischerweise implementiert ein Framework mehrere Patterns. Im Gegensatz zur Verwendung von Klassenbibliotheken wird beim Einsatz von Frameworks im wesentlichen nur Code geschrieben, der vom Framework aufgerufen wird.

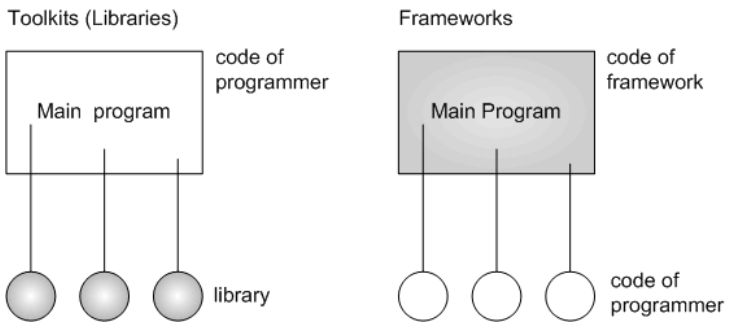

Abbildung 3-1: Framework-Toolekit-Unterschied [Benk03]

Ein Framework wird also für eine bestimmte Anwendung spezialisiert, indem anwendungsspezifische Unterklassen aus den abstrakten Framework-Klassen gebildet werden. Eine Klassenbibliothek hingegen bietet nützliche Funktionalität, legt aber die Entwurfsstruktur einer Anwendung nicht von vornherein fest (vgl. Abbildung 3-1) [Eich04, GHJV04].

3.3 Klassifikation von Patterns

Für die Fülle an bereits existierenden Patterns zeigt sich, dass es Unterschiede in Größe und Abstraktionsebene gibt. Einige Patterns werden zur Strukturierung eines Systems in Subsysteme eingesetzt während andere wiederum eine Implementierung eines Patterns auf Ebene einer spezifischen Programmiersprache darstellen. [BMR+01] treffen eine Einteilung in drei Kategorien:

1. Architektur Patterns,
2. Design Patterns und
3. Idiome

3.3.1 Architektur-Patterns

Ein Architektur-Pattern beschreibt eine fundamentale, strukturelle Organisation eines Softwaresystems. Die Komponenten in dieser Organisation bewegen sich dabei auf Subsytemebene. Daneben werden Verantwortlichkeiten dieser Subsysteme beschrieben und Regeln und Richtlinien für die Organisation der Beziehungen zwischen den Subsystemen definiert. Eines der bekanntesten Beispiele ist das Model View Controller Pattern, welches die Struktur für interaktive Systeme vorgibt (vgl. Unterkapitel 3.4).

3.3.2 Design-Patterns

Design-Patterns beschreiben ein Schema mit dem Subsysteme, Komponenten oder deren Beziehungen untereinander verfeinert werden können. Diese Beschreibung erfolgt auf einer niedrigeren Abstraktionsebene ist aber noch immer sprachunabhängig.

3.3.3 Idiome

Ideome sind Low-Level-Patterns, die sich auf eine spezifischen Programmiersprache beziehen. Sie beschreiben wie ein bestimmter Aspekt unter Ausnützung der speziellen Eigenschaften der gewählten Programmiersprache implementiert werden kann.

3.4 Model View Controller

Eines der ersten und auch eines der bekanntesten Patterns ist das Model View Controller-Pattern (im Folgenden MVC), das in Smalltalk-80 für den Entwurf von Software mit Benutzerschnittstellen eingesetzt wurde. Nach [BMR+01] ist das MVC Pattern in die Kategorie der Architektur-Patterns einzuordnen.

Die Herausforderung an die Architektur eines interaktiven Systems, ist den funktionalen Kern von der Benutzerschnittstelle zu trennen. Dieser funktionale Kern bleibt gewöhnlich über einen längeren Zeitraum stabil, während die Benutzerschnittstelle häufiger Änderungen oder kleineren Anpassungen unterworfen ist [BMR+01]. Speziell Web-Anwendungen müssen verschiedene Sichten auf Daten bzw. verschiedene Darstellungsformate zur Verfügung stellen. Dies ist einerseits erforderlich, um diverse Endgeräte (PCs, Handys, PDAs, Drucker,...) zu unterstützen und den Benutzer auf möglichst vielen Kanälen erreichen zu können, andererseits um bestimmten Benutzergruppen eine eingeschränkte Sicht, abhängig von ihren Benutzerrechten, auf gewisse Daten zu geben.

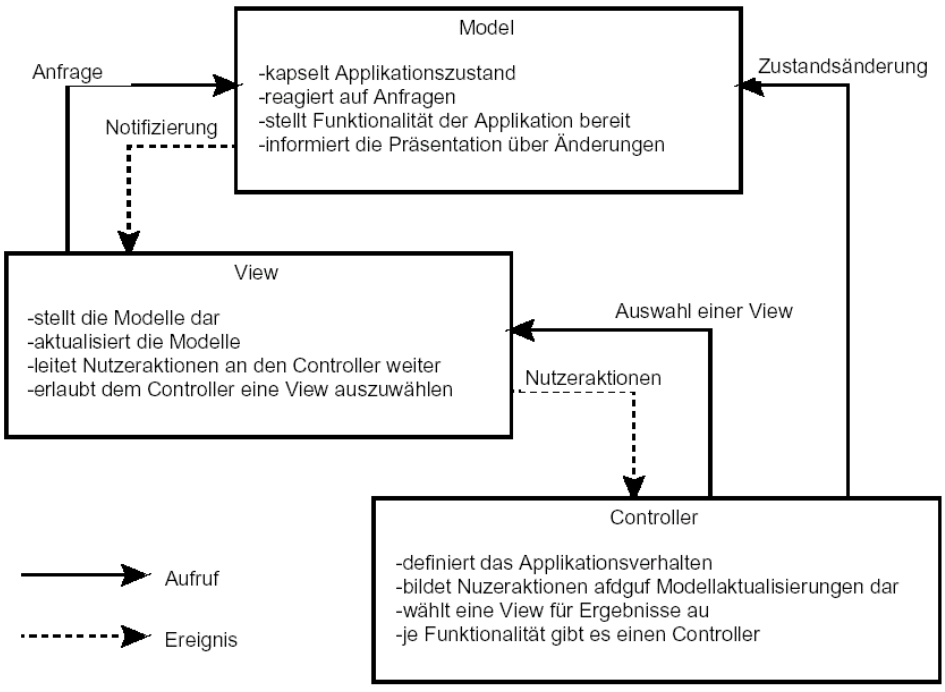

Abbildung 3-2: Model View Controller Pattern [AnRo01]

Wie in Abbildung 3-2 zu sehen ist, teilt das MVC Pattern eine interaktive Anwendung in drei Bereiche ein [BMR+01]:

Die *Model*-Komponente stellt die Anwendungslogik dar und beinhaltet Daten und die darauf definierten Operationen. Das Model ist völlig unabhängig von Präsentationsaspekten oder Eingabeverhalten. Im Falle einer Datenänderung, sorgt das Model dafür, dass alle interessierten Views und Controller informiert werden. An dieser Stelle kommt das Publisher-Subscriber Pattern [GHJV04] zum Einsatz, da das Model von den anderen interessierten Objekten entkoppelt ist.

Die *View*-Komponente stellt Informationen lediglich dar, indem sie die aktuellen Daten des Models abruft. Wird der View über eine Änderung des Models informiert, muss er die Informationen neu abrufen und die Anzeige aktualisieren. Für ein Model kann es mehrere Views geben, welche die selbe Information auf unterschiedliche Weise darstellen.

Jedem View ist ein *Controller* zugeordnet. Der Controller ist die Komponente, die Benutzerinteraktionen entgegennimmt und als Aufrufe bzw. Anfragen an das Model weiterleitet. Zusammen stellen Controller und View die Benutzerschnittstelle dar.

Im Kontext von Web-Anwendungen erfährt das MVC Pattern eine andere Interpretation. Durch die Verwendung des zustandslosen HTTP-Protokolls, können View- und Controller-Objekte im Falle einer Änderung des Models nicht benachrichtigt werden. Abgesehen davon wäre dies für unter Umständen Hunderte oder auch Tausende Clients nicht durchführbar. Stattdessen geht die Kommunikation immer vom Client aus und bewirkt einen vollständigen Neuaufbau der Ansicht [AnRo01]. Eine Trennung von Präsentations- und Anwendungslogik bleibt jedoch auch im Web-Kontext notwendig.

3.5 Verwendete Patterns und Alternativen

Dieses Unterkapitel stellt die in der Architektur des entwickelten Institutsinformationssystems verwendeten Patterns und eventuelle Alternativen vor. Die Patterns wurden [Fowl03a] und [AlCM01b] entnommen und werden an dieser Stelle skizziert.

3.5.1 Page Controller

„Ein Objekt, das eine Anfrage für eine bestimmte Seite oder Aktion einer Website bearbeitet.“[Fowl03a]

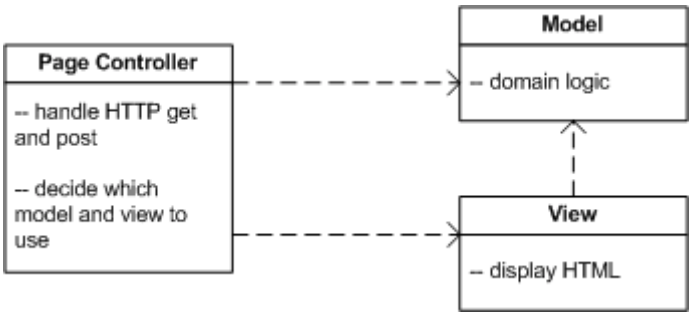

Abbildung 3-3: Page Controller Pattern [Fowl03a]

Wie der Name des Patterns bereits vermuten lässt, gibt es für jede Seite bzw. Aktion einer Web-Anwendung einen Controller. Ob der Page Controller als Skript oder als Server Page realisiert wird, hängt im wesentlichen von der Komplexität des Anwendungsfalls ab. Sobald die enthaltene Logik allerdings über eine simple Anzeige mit kleinen dynamischen Elementen hinausgeht, sollte man versuchen, die Logik in ein Hilfsobjekt auszulagern bzw. ein Skript zu verwenden, welches nach dem Abarbeiten des Codes die entsprechende, nächste Anzeige (möglicherweise eine Server Page) aufruft [Fowl03a].

Die grundlegenden Aufgaben des Page Controllers sind:

- Herausfiltern aller Formulardaten aus der HTTP-Anfrage oder der URL,
- Erzeugen und Aufrufen von Model-Objekten, die obige Daten verarbeiten,
- Ermitteln des nächsten Views und damit Anzeigen der Model-Informationen.

Das Page Controller Pattern bietet sich für jene Fälle an, in denen die Logik im Controller eher einfach ist. Die Alternative Front Controller sollte man dann in Betracht

ziehen, wenn bestimmte Codefragmente in jedem Page Controller vorkommen. Üblicherweise kann man dem Problem auch durch Vererbung, oder Einbindung von Bibliotheken im Falle nicht objektorientierter Programmierung, entgegenkommen. Beispielsweise können bestimmte, über alle Seiten hinweg gleich bleibende, Teile (z. B. Navigationsmenü) vom Elternobjekt übernommen werden. Für komplexere Aufgaben, wie Authentifizierung, bietet sich aber das Front Controller Pattern an, das im nächsten Abschnitt vorgestellt wird.

3.5.2 Front Controller

„Ein Controller, der alle Anfragen an eine Website empfängt und bearbeitet." [Fowl03a]

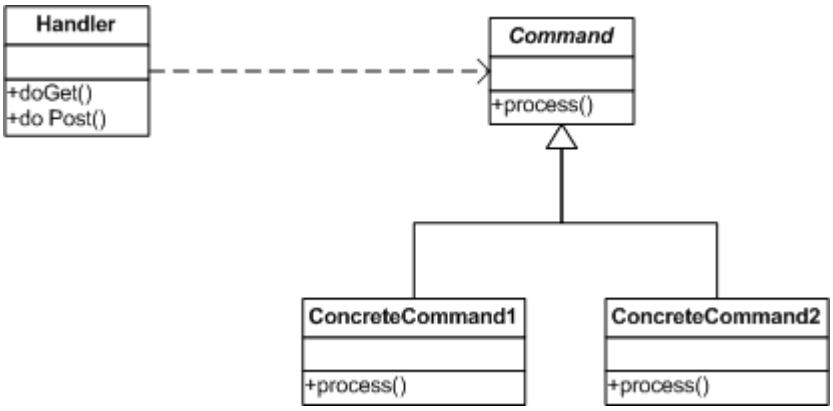

Abbildung 3-4: Front Controller Pattern [Fowl03a]

Der Front Controller ist der zentrale Anlaufpunkt aller Anfragen einer Web-Anwendung. Normalerweise ist der Front Controller zweiteilig:

1. Der *Web Handler* erhält alle Anfragen und entscheidet anhand der URL und dem HTTP-Request welches Command die eigentliche Bearbeitung übernehmen soll. Zusätzlich werden bestimmte Aufgaben erledigt, die für jede Seite benötigt werden. Dies betrifft Sicherheitsaspekte, Darstellungsformat, Zeichensatz, etc. (vgl. Abbildung 3-5).
2. Command-Hierarchie: Ein *Command* ist mit einem Page Controller in dem Sinn vergleichbar, als es ein Command für jede Seite bzw. Aktion einer Web-Anwendung gibt. Im Command befindet sich die spezielle Logik, die für eine Aktion ausgeführt werden muss.

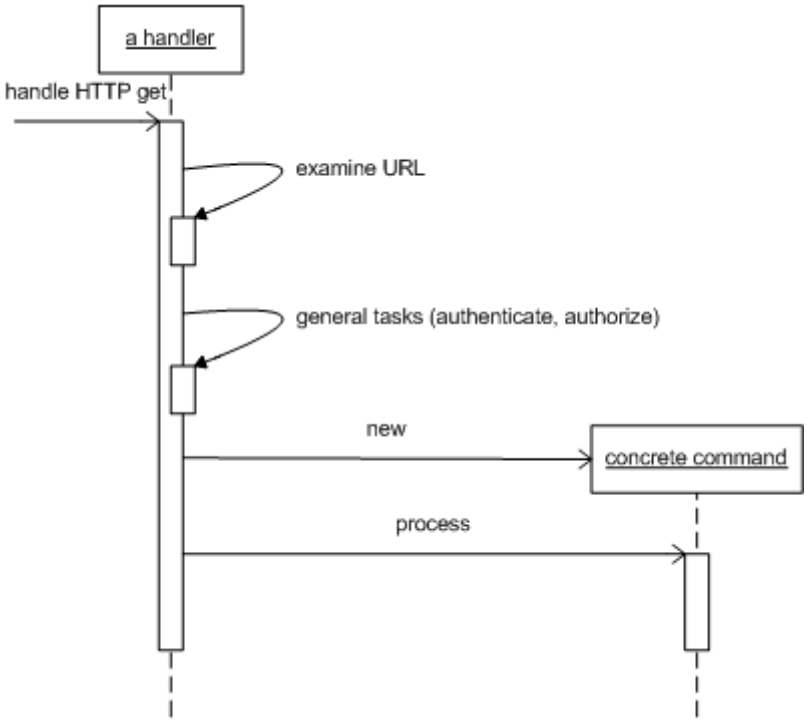

Abbildung 3-5: Front Controller Pattern Sequenzdiagramm (in Anlehnung an [Fowl03a])

Der Web Handler, der die Anfragen nur weiterleitet, wird als Skript oder Klasse implementiert. Ein Command kann, je nach Komplexität der Aktion, als Server Page, als Skript oder Klasse realisiert werden.

Die wesentlichen Vorteile des Front Controller Patterns ergeben sich durch die zentrale Kontrolle. So gibt es ausschließlich einen Einstiegspunkt in die Anwendung an dem

- Anfragen eines Benutzers sehr leicht nachvollzogen und geloggt werden können,
- zentral Sicherheitsmaßnahmen durchgeführt werden können
- die Wiederverwendbarkeit von Code gefördert wird, indem allgemeine Codefragmente nun nicht mehr in jede Seite bzw. Aktion eingebaut sind [AlCM01a].

Die Auswahl eines Commands kann auf zwei verschiedene Arten ausgeführt werden. Folgend werden die physikalische und logische Zuordnung von Anfragen zu einem Command vorgestellt.

Physikalische Zuordnung

Ein Webserver ist ein Programm, das auf einem leistungsstarken Rechner läuft. Ein bestimmtes Verzeichnis im Dateisystem des Servers dient als öffentliches Web-Verzeichnis. Bekommt der Webserver eine Anfrage, so verwendet er den Teil der URL nach dem Domain-Namen als Pfad zu einer Datei im Web-Verzeichnis. Demnach befindet sich die angeforderte Datei „index.php“ aus folgender Beispiel-URL direkt unterhalb der Wurzel des Web-Verzeichnisses.

```
http://www.example.com/index.php
```

Diese Art der Zuordnung wird auch physikalische Zuordnung genannt [AlCM01a].

Verwendet man das Front Controller Pattern, dann werden alle Anfragen der Clients über *eine* Ressource im Web-Verzeichnis laufen. Alle anderen notwendigen Dateien können, müssen aber nicht, in ein anderes Verzeichnis am Server abgelegt werden. Liegt eine Datei aber im Web-Verzeichnis, ist sie öffentlich abrufbar, sofern die URL bekannt ist. Sobald man einen Front Controller einsetzt, um Benutzer authentifizieren und autorisieren zu können, ist Vorsicht geboten. Jene Dateien, die eine bestimmte Berechtigung verlangen, sollten dann nicht mehr im Web-Verzeichnis liegen, denn mit Kenntnis der URL sind sie abrufbar!

Dem Front Controller muss über die URL die eigentlich gewünschte Aktion bekannt gegeben werden. Dies kann nun wieder über eine physikalische Zuordnung geschehen.

```
http://www.examle.com/index.php?action=people/Schauerhuber.html
```

Obiges Beispiel teilt dem Front Controller mit, dass die Datei „Schauerhuber.html“ im Verzeichnis „people“ gewünscht wird.

Logische Zuordnung

Die zweite Möglichkeit, ein Command auszuwählen, ist die logische Zuordnung. Die Auswahl einer Ressource über einen logischen Namen stattet die EntwicklerInnen mit größerer Flexibilität aus.

```
http://www.example.com/index.php?action=main
```

Bei der logischen Zuordnung bieten sich zwei Vorgehensweisen an. Entweder man arbeitet mit Namenskonventionen und die gewünscht Ressource wird dynamisch

über den Namen ermittelt oder die Zuordnung einer Anfrage zu einer Ressource wird statisch über eine Konfigurationsdatei gelöst.

- Im Falle der Namenskonvention bildet der Front Controller aus den URL-Informationen dynamisch einen Pfad zur gewünschten Datei. Um die richtige Ressource zu finden, könnte der Front Controller beispielsweise den logischen Namen um die Zeichenkette „_action.php“ erweitern und somit den Dateinamen der Ressource ermitteln.
- Mit Hilfe einer Konfigurationsdatei kann eine Zuordnung von logischen Namen zu physikalischen Ressourcen im Dateisystem in einfacher Weise verändert werden. Der wesentliche Vorteil dieser Methode ist die zusätzliche Abstraktionsschicht zum Dateisystem, die es den EntwicklerInnen freistellt die URLs völlig unabhängig vom Dateisystem zu gestalten bzw. die Ressourcen im Dateisystem zu einem späteren Zeitpunkt ohne großen Aufwand anders anzuordnen.

Eine fortgeschrittenere Zuordnung kann mehrere logische Namen auf eine physikalische Ressource abbilden. Dies geschieht über reguläre Ausdrücke. Möchte man beispielsweise alle Bilddateien in gleicher Weise behandeln, so könnte man sich darauf einigen, dass deren logische Namen die Endung „.pic“ haben müssen.

```
http://www.example.com/index.php?action=logo.pic
```

In einer Konfigurationsdatei muss es dann einen entsprechenden Eintrag geben, der dieses Muster einer bestimmten physikalischen Ressource zuordnet.

```
*.pic -> servePicture.php
```

Cocoon (http://cocoon.apache.org/) und Struts (http://struts.apache.org/), beides Java-basierte Open-Source Frameworks der Apache Software Foundation, setzen diese Zuordnung in ihren Konfigurationsdateien „sitemap.xmap“ bzw. „struts-config.xml“ um.

3.5.3 Intercepting Filter

Ein interessantes Pattern, das in Verbindung mit dem Front Controller verwendet werden kann, ist das Intercepting Filter Pattern [AlCM01b].

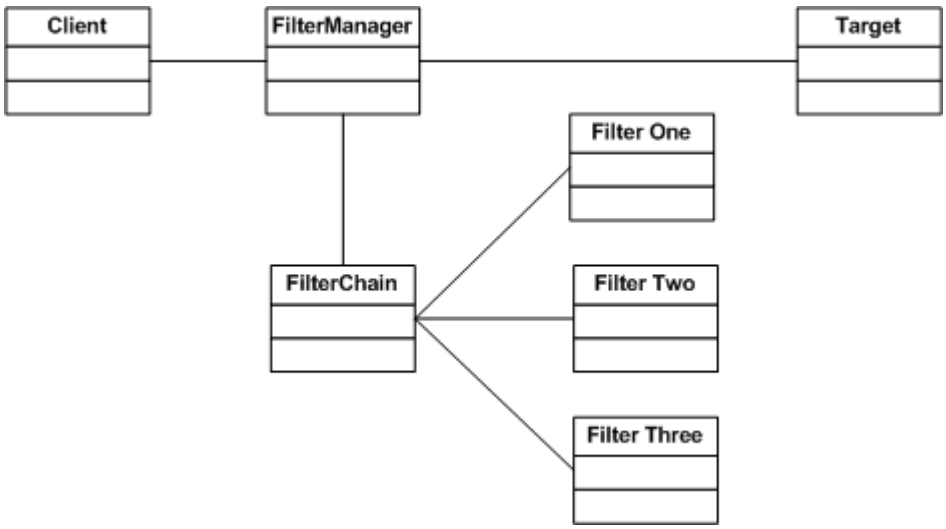

Abbildung 3-6: Intercepting Filter Pattern [AlCM01b]

Abbildung 3-6 zeigt die am Intercepting Filter Pattern beteiligten Objektklassen. Der FilterManager erzeugt die FilterChain und initiiert die Verarbeitung. Die FilterChain ist eine geordnete Liste der anzuwendenden Filter. Jeder Filter übernimmt eine spezielle Aufgabe (Loggen, Zeichensatz, Überprüfung des Browser-Typs, etc.), die für jede Anfrage durchgeführt werden muss, bevor die eigentliche Aktion (Target) ausgeführt wird. Die einzelnen Filter können abhängig von den Anforderungen aneinandergereiht werden, wodurch die Anwendung besser konfigurierbar wird.

3.5.4 Template View

Eine HTML-Seite dient als Schablone für eine spezielle Anfrage durch einen Client. Markierungen innerhalb der HTML-Seite kennzeichnen dynamische Inhalte. Sie werden durch den für die Anfrage spezifischen HTML-Code ersetzt.

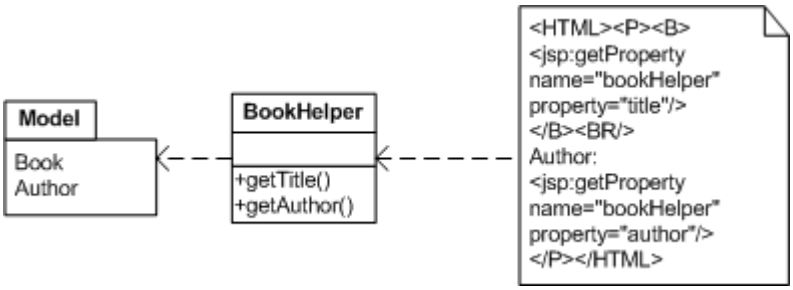

Abbildung 3-7: Template View Pattern [Fowl03a]

Für die Umsetzung der View-Komponente im MVC Pattern bieten sich das Template View Pattern oder das Transformer View Pattern an, welches im nächsten Abschnitt beschrieben wird.

In dynamisch generierten Webseiten werden Informationen beispielsweise aus Datenbanken beschafft und in den HTML-Code eingebunden. Üblicherweise werden dynamische Teile einer Webseite mit besonderen Markierungen gekennzeichnet. Diese werden beim Aufruf der Webseite erkannt und durch den dynamisch ermittelten HTML-Code ersetzt. Der statische Teil der Webseite dient im Prinzip als Schablone für eine spezielle Anfrage, daher der Name Template View.

Eine bekannte Form des Template View Patterns sind Server Pages, wie ASP, JSP oder PHP. Diese Technologien gehen allerdings einen Schritt weiter, indem sie es erlauben beliebige Anwendungslogik als so genannte Scriptlets in die Seite einzubauen. Die große Gefahr hierbei ist die Vermischung der verschiedenen logischen Ebenen einer Web-Anwendung, nämlich Inhalte, Anwendungslogik und Präsentation. Eine Möglichkeit Scriptlets zu vermeiden sind Hilfsobjekte, welche die Anwendungslogik vollständig kapseln. Die HTML-Seite selbst enthält lediglich die Objektaufrufe. Durch die Verwendung dieser Schnittstelle können sich DesignerInnen auf den Seitenaufbau und das Layout, und EntwicklerInnen auf das Hilfsobjekt konzentrieren. Problematisch sind allerdings bedingte Anzeigen und Iterationen innerhalb

einer Seite. In diesem Fall kann es zu keiner vollständigen Trennung der Verantwortlichkeiten kommen. Entweder man nimmt die Logik als spezielle Markierungen in die Schablone auf oder man lässt HTML-Code im Hilfsobjekt zu. Welche der beiden Lösungen die besser ist, bleibt subjektiv. [Fowl03a] bietet als Entscheidungshilfe einige Code-Beispiele an.

3.5.5 Transform View

„Ein View, der die Daten des Anwendungsbereichs in HTML[3] transformiert." [Fowl03a]

Abbildung 3-8: Transform View Pattern [Fowl03a]

Hinter der Idee des Transform View Patterns steckt ein Programm, das die Daten des Anwendungsbereichs als Input entgegennimmt und als Output reines HTML liefert. Das Programm inspiziert jedes Element der Daten und transformiert es in ein spezifisches HTML-Fragment.

Das Transform View Pattern kann in jeder beliebigen Sprache umgesetzt werden, jedoch ist die momentan vorherrschende Wahl XSLT. Für die Transformation ist zunächst ein Vorliegen der Daten in XML notwendig. Die Transformationslogik liegt separat in einem XSLT-Stylesheet. Ein XSLT-Prozessor wendet dann das Stylesheet auf die XML-Daten an und produziert strukturierten Output im gewünschten Format, z. B. HTML. Das Ergebnis der Transformation kann schließlich direkt in die HTTP-Response geschrieben werden.

---

[3] [Fowl03a] bezieht sich in seinen Ausführungen immer auf HTML als Ausgabeformat. Das erwähnte Programm könnte aber beliebige Formate unterstützen.

3.5.6 Table Module

„Ein Objekt, das die Anwendungslogik für alle Einträge in einer Datenbanktabelle oder einem Datenbank-View übernimmt.“ [Fowl03a]

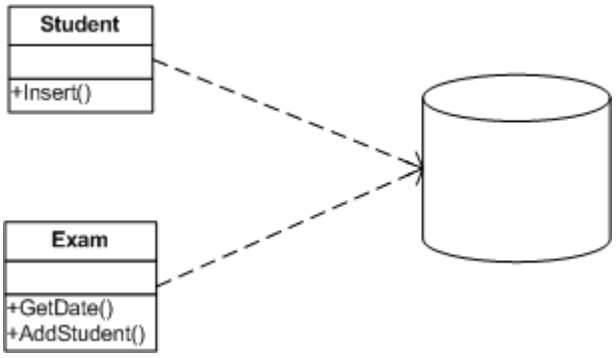

Abbildung 3-9: Table Module Pattern [Fowl03a]

Mit dem Table Module Pattern orientieren sich die Klassen der Anwendung an den Datenbanktabellen. Eine Klasse kapselt die gesamte Anwendungslogik für die Daten einer Datenbanktabelle. Ein wichtiges Merkmal des Patterns ist, dass eine Instanz der Klasse für alle Einträge der Tabelle verantwortlich ist. Soll ein spezieller Eintrag bearbeitet werden, muss dieser über eine ID angesprochen werden.

Ein Table Module kann direkt Datenbankabfragen enthalten (vgl. Abbildung 3-9: Insert-Methode). Als Alternative kann das Table Data Gateway Pattern (vgl. nächster Abschnitt) die Verbindung zur Datenbank herstellen. Dies hat den Vorteil, dass ein einziges Table Module Daten über verschiedene Table Data Gateways von verschiedenen Datenquellen beziehen kann. Nachteilig sind die dafür notwendigen zusätzlichen Klassen. Das Pattern wird eingesetzt, wenn die Anwendungslogik relativ einfach bleibt. Bei komplexeren Anwendungen ist eher das Domain Model Pattern[4] vorzuziehen [Fowl03a].

[4] Im Domain Model Pattern [Fowl03a] werden Klassen rund um die Nomen der Anwendung gebildet, z. B. Bestellung. Im Gegensatz zum Table Module Pattern entspricht eine Instanz einer Bestellung. Die Schwierigkeit dieses Patterns liegt in der Abbildung des Objekts auf die Struktur einer relationalen Datenbank. Das Data Mapper Pattern verschafft hier Abhilfe [Fowl03a].

3.5.7 Table Data Gateway

„Ein Objekt, das als Gateway für eine Datenbanktabelle fungiert. Eine Instanz ist für alle Einträge in der Tabelle verantwortlich.“ [Fowl03a]

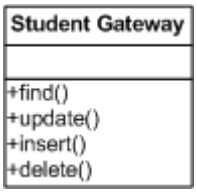

Abbildung 3-10: Table Data Gateway Pattern [Fowl03a]

Das Table Data Gateway Pattern kapselt den SQL-Code, der für die Interaktion mit einer Datenbanktabelle notwendig ist, innerhalb einer Klasse. Das einfache Interface zur Datenbanktabelle besteht aus einigen Methoden, um Daten zu lesen, und aus Methoden, um Daten zu schreiben (vgl. Abbildung 3-10). Im allgemeinen gibt es für jede Datenbanktabelle einen Table Data Gateway. Abhängig vom Anwendungsfall kann ein Table Data Gateway aber auch mehr als eine Tabelle ansprechen.

Kapitel 4

4 Web-Entwicklung mit LAMP

LAMP ist eine Open-Source Web-Development-Platform basierend auf dem Betriebsystem Linux, dem Webserver Apache, MySQL als relationales Datenbankmanagementsystem und der Skriptsprache PHP. Das Akronym wurde erstmalig in [Kunz98] erwähnt. In der Zwischenzeit wird das „P“ auch als Synonym für Perl oder Python (ebenfalls Skriptsprachen) verwendet. Darüber hinaus haben sich zusehends Begriffsderivate gebildet, wie z. B. WAMP, um den Einsatz von Windows anstelle von Linux anzudeuten. Im englischsprachigen Raum wurde der Begriff LAMP durch den O’Reilly Verlag [Doug01] und durch MySQL AB (www.mysql.org), der schwedischen Entwicklungsfirma von MySQL, bekannt [Wiki04]. Seitdem konnte sich die LAMP-Architektur in unzähligen Beispielen [Doug01] als eine billige, zuverlässige, skalierbare und sichere Lösung für Entwicklung und Einsatz von Web-Anwendungen etablieren.

In diesem Kapitel werden die verwendeten Technologien vorgestellt (siehe Unterkapitel 4.1), um dann genauer auf die Entwicklung von Web-Anwendungen in PHP einzugehen. Spezielles Augenmerk wird auf die Vorgehensweise der PHP-Entwicklergemeinde zur Trennung von Inhalten, Logik und Präsentation gelegt. U. a. wird das letztendlich eingesetzte XAO Framework vorgestellt (vgl. Unterkapitel 4.2).

4.1 XAMPP – das vorkonfigurierte LAMP-System

Die Installation von Apache, MySQL und PHP stellt sich für einen Neuling als ein schwieriges Unterfangen dar, vor allem, wenn man dazu auch noch völlig unerfahren in der Arbeit mit dem Betriebsystem Linux ist. Selbst die zahlreichen Installationsanleitungen, die im Web zu finden sind, können einem sog. „Newbie“ den bevorstehenden Installationsmarathon nicht wesentlich erleichtern. Abhilfe schafft das vorkonfigurierte LAMP-System XAMPP, der Apache Friends (www.apachefriends.org). Hinter dem Akronym XAMPP verstecken sich drei Distributionen für Windows, Linux und Solaris. Das zusätzliche „P“ repräsentiert die Skriptsprache Perl. Außer dem Webserver, der Datenbank und den beiden Skriptsprachen umfasst XAMPP noch eine Reihe weiterer nützlicher Komponenten, wie z. B. OpenSSL, phpmyadmin u. v. m[5]. Ist erst die dem verwendeten Betriebssystem entsprechende XAMPP-Distribution heruntergeladen, muss das Paket lediglich entpackt und gestartet werden. Die Installation ist für Test- und Entwicklungszwecke völlig ausreichend und erleichtert den Einstieg in die Web-Programmierung beträchtlich. Im Produktivbetrieb müssen im Sinne der Sicherheit für Anbieter und Benutzer der Web-Anwendung allerdings einige Vorkehrungen getroffen werden (vgl. dazu [Jova04]). Im Rahmen dieser Arbeit wurde die Linux-Distribution von XAMPP in der Version 1.4.6 verwendet. Folgend werden die einzelnen Komponenten des erfolgreichen Quartetts vorgestellt.

[5] Eine genaue Auflistung findet sich in der Beschreibung zur aktuellen Version (http://www.apachefriends.org/de/xampp-linux.html).

4.1.1 Linux

Linux entwickelte sich aus der Idee des finnischen Studenten Linus Torvalds, ein UNIX- bzw. MINIX[6]-ähnliches Betriebssystem zu programmieren. 1991 stellte Torvalds sein kleines Projekt, welches sich im Laufe der Zeit zu einem umfangreichen Betriebssystem entwickelt hat, in einer USENET-Newsgroup vor. Linux wird unter der *GNU Public License* (GPL) entwickelt und ist daher für jedermann frei erhältlich. Außer der freien Verfügbarkeit zeichnet sich Linux auch durch folgende Eigenschaften aus [Goll02]:

- Portierbarkeit auf verschiedenste Plattformen
- Multitasking – Programme werden (scheinbar) simultan ausgeführt.
- Multiuser – Linux kann mehrere Personen gleichzeitig (meist über ein Netzwerk) bedienen.
- Multiprozessor – Linux kann mehrere Prozessoren im Rechner ansprechen und eine Lastverteilung bewirken.
- u. v. m.

4.1.2 Apache

Apache (http://httpd.apache.org/) ist der meistverbreitete Webserver im Internet: In der Web Server Studie von Netcraft, Stand 01.10.2004, liegt Apache an erster Stelle mit 67.85 Prozent gefolgt von Microsoft's IIS mit 21.14 Prozent [Netc04]. Apache wird für eine Vielzahl verschiedener Betriebssysteme angeboten, darunter auch UNIX- und Windows-Betriebssysteme. Ebenso wie Linux ist auch Apache ein Open-Source-Projekt.

4.1.3 MySQL

MySQL ist ein relationales Datenbankmanagementsystem. Über fünf Millionen aktive Installationen und Kunden, wie Yahoo!, Google, oder NASA, sind Grund genug,

---

[6] MINIX ist ein UNIX-Derivat und dient hauptsächlich Lernzwecken. Es wurde von Andrew S. Tanenbaum entwickelt.

um MySQL als „The World's Most Popular Open Source Database“ zu bezeichnen[7]. Zu den Vorteilen von MySQL zählen u. a. [Goll02]:

- Multiuser-Architektur
- Multithreaded-Architektur
- Schnittstellen zu den gängigsten Programmiersprachen (C, C++, Perl, PHP, Python, Java etc.)
- Verfügbarkeit auf vielen Plattformen (Windows, Linux, etc.)
- Hohe Geschwindigkeit und Stabilität
- Entwicklung unter der GPL
- Kostenlos für den privaten und nichtkommerziellen Einsatz

4.1.4 PHP

Das rekursive Akronym PHP steht für PHP Hypertext Preprocessor. Darunter versteht man einerseits einen Interpreter und anderseits eine Open-Source-Skriptsprache, die vor allem in der Web-Entwicklung weit verbreitet ist. PHP-Befehle lassen sich als spezielle Kommentare direkt in den HTML-Code einbetten. Ein wichtiges Merkmal von PHP, im Gegensatz zu anderen Skriptsprachen wie beispielsweise JavaScript, ist die serverseitige Ausführung des Programmcodes. Webseiten, die PHP-Code enthalten, werden mit einer speziellen Endung, im allgemeinen „.php“, für den Webserver gekennzeichnet. Die zuvor erwähnten speziellen Kommentare, mit denen PHP-Code in den HTML-Code eingebettet werden kann, beginnen mit „<?php“ und enden mit „?>“. Variationen dieser Schreibweise sind möglich, sollten aber aus Portabilitätsgründen nicht verwendet werden [ABD+04]. Mit der erst kürzlich veröffentlichten Version 5 verlässt PHP den Status einer einfachen Skriptsprache und will als objektorientierte Programmiersprache ernst genommen werden.

[7] Siehe www.mysql.com

4.1.5 Das Zusammenspiel

Wie in Abbildung 4-1 zu sehen, ist die LAMP-Architektur eine einfache 2-Schichten Architektur. Nach [KRP+03] sind LAMP-Systeme in die Kategorie der datenbankgestützten Web-Informationssysteme einzuordnen.

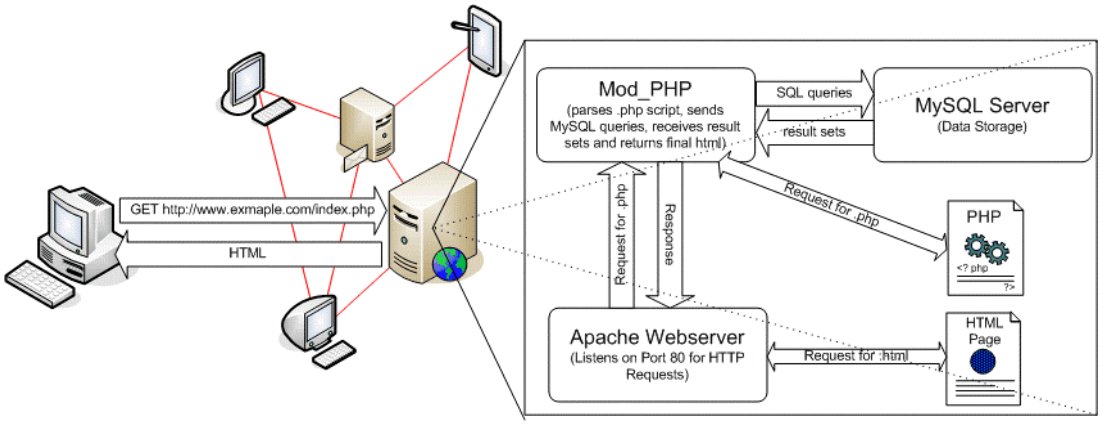

Abbildung 4-1: LAMP-Architektur (in Anlehnung an [Rols03])

Die nächsten beiden Code-Beispiele erzeugen im Browser des Clients die selbe Ausgabe. Im Gegensatz zur statischen HTML-Datei (vgl. Code-Beispiel 4-1) wird der Text für Titel und Überschrift in der PHP-Datei erst zum Zeitpunkt der Anfrage eingefügt (siehe Code-Beispiel 4-2). Die in den HTML-Code eingebetteten Befehle werden vom PHP-Modul ausgeführt und ersetzt, bevor die Webseite zum Client geschickt wird. Man spricht auch von „server parsed HTML“.

```
<!-- Hello.html -->

<html>
      <head>
            <title>Hello World!</title>
      </head>
<body>
      <h1>Hello World!</h1>
</body>
</html>
```

Code-Beispiel 4-1: Eine statische Seite

Der Unterschied liegt also in der Verarbeitung am Server. Im Falle der HTML-Version muss der Webserver nur die statische Datei zurückschicken. Anhand der

Endung, im Falle der PHP-Version, erkennt der Webserver, dass er diese Datei parsen, d.h. den enthaltenen PHP-Code ausführen, durch HTML-Code ersetzen und das Ergebnis an den Client zurückschicken muss (vgl. Abbildung 4-1).

```
<!-- Hello.php -->

<html>
      <head>
            <title><?php echo(“Hello World!”)?></title>
      </head>
<body>
      <h1><?php echo(“Hello World!”)?></h1>
</body>
</html>
```

Code-Beispiel 4-2: Eine dynamische Seite mit Code-Elementen in PHP

Der Client wird den Unterschied nicht bemerken, für ihn entspricht die Ausgabe in beiden Fällen der oberen Datei „Hello.html“.

4.2 Web-Entwicklung mit PHP

Die wichtigste Anforderung an die Architektur des zu entwickelnden Systems ist die Separation of Concerns, die Trennung von Daten, Logik und Präsentation (vgl. Unterkapitel 5.1). Eine wesentliche Aufgabe dieser Arbeit, ist es aufzuklären, ob und wie die PHP-Entwicklergemeinde dieser Herausforderung gegenübertritt. Drei Möglichkeiten sollen im Folgenden vorgestellt werden: Template Engines, Frameworks (die das MVC Pattern unterstützen) und der Einsatz von XML.

4.2.1 Template Engines

PHP-Skripte fürs Web gelten gemeinhin als schwer wartbares Gewirr aus HTML-Tags und PHP-Code. Rettung versprechen so genannte Templates (Schablonen). Dabei wird über die in einem PHP-Skript erzeugten Daten eine HTML-Schablone gelegt, die das Layout der Webseite beschreibt. Im Web sind unzählige Projekte für Schablonenverarbeitungssysteme[8] zu finden [WeLa04]. Zwei Projekte wurden zur Demonstration ausgesucht: PHP-Templates (http://sourceforge.net/projects/php-templates/) und Smarty (http://smarty.php.net/). Während Smarty wohl zu den bekanntesten und größten Template Engines zählt, hebt sich das eher leichtgewichtigere PHP-Template Projekt durch seine Implementierung in der Programmiersprache C von den zahlreichen Alternativen ab.

Folgendes Beispiel soll die Vorgehensweise in PHP-Templates verdeutlichen. Für jede logische Webseite gibt es nun zwei Dateien:

- eine Schablone und
- ein PHP-Skript, welches die Daten für die Schablone bereitstellt.

Für viele Anwendungen genügt ein „flat“ Template, in welches die dynamischen Teile durch Platzhalter (z. B. *{title}*) eingefügt werden. Siehe dazu Code-Beispiel 4-3.

---

[8] Ein Versuch alle existierende Template Engines aufzulisten, ist unter http://www.sitepoint.com/forums/showthread.php?threadid=123769 zu finden.

```
<!– test.tmpl -->
<html>
  <head>
    <title>{title}</title>
  </head>
  <body>
    <h1>{title}</h1>
  </body>
</html>
```

Code-Beispiel 4-3: Ein PHP-Template

Wird die Webseite aufgerufen, dann werden zuerst die notwendigen Daten ermittelt (z. B. durch Datenbankabfragen), das richtige Template wird ausgewählt und die Daten werden in Form eines assoziativen Arrays an das Template übergeben. Die Platzhalter werden während der Verarbeitung durch die Daten des assoziativen Arrays ersetzt, wobei der Name eines Platzhalters als Arrayindex dient (vgl. Code-Beispiel 4-4).

```
/* test.php */
<?php
$data = array(
      ´title´ => ´Hello World!´,
      ...
);
$tpl = tmpl_open(´test.tmpl´);
tmpl_set($tpl,$data);
echo tmpl_parse($tpl);
tmpl_close($tpl);
?>
```

Code-Beispiel 4-4: Ein PHP-Skript, das dynamische Daten in ein Template einfügt.

Mit einem „echo“-Befehl wird das Ergebnis (reines HTML) des Ersetzungsvorgangs an den Browser zurückgeschickt (siehe Code-Beispiel 4-5).

```
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

Code-Beispiel 4-5: Das HTML-Ergebnis

Template Engines implementieren im wesentlichen das Template View Pattern, welches im Unterkapitel 3.5 beschrieben wurde. Wie bereits dort angesprochen, ist eine vollständige Trennung von HTML- und PHP-Code nicht mehr möglich, sobald bedingte Anzeigen oder Iterationen notwendig sind. In PHP-Templates gibt es eine besonders saubere Lösung für Iterationen. Dabei bleibt der HTML-Code in der Schablone und die Logik wird von der Template Engine übernommen [WeLa04]. Ein Vorteil dieser Lösung ist, dass auch leicht anderer „Markup“ als HTML, z. B. WML, eingesetzt werden kann. Eine Lösung für eine bedingte Anzeige wird nicht angeboten[9].

Smarty geht für obiges Beispiel die Notation betreffend einen ähnlichen Weg wie PHP-Templates. Der Umfang des Handbuchs zeugt aber von der Komplexität dieses Werkzeugs. Laut [OZHS04] ist es mit Smarty möglich die Anwendungslogik vom Design zu trennen. Dies sei vor allem dann wünschenswert, wenn Applikationsentwickler und Designer nicht ein und die selbe Person sind. Leider wurde dieser Anspruch in Smarty nicht erfüllt. Der Teil des Handbuchs, der nur für das Designteam gedacht ist, umfasst acht Kapitel mit namhaften Überschriften, darunter „Variablen“ und „Eingebaute Funktionen“. Im Prinzip wurde in Smarty nichts anderes als eine

[9] Vgl. Dokumentation unter http://sourceforge.net/projects/php-templates/

Metasprache eingeführt, deren Sinnhaftigkeit mit folgendem Beispiel[10] in Frage gestellt werden soll.

```
{* ein Beispiel mit 'eq' (gleich) *}
{if $name eq "Fred"}
      Willkommen der Herr.
{elseif $name eq "Wilma"}
      Willkommen die Dame.
{else}
      Willkommen, was auch immer Du sein magst.
{/if}
```

Code-Beispiel 4-6: Bedingte Ausgabe in der Metasprache von Smarty

In PHP könnte obiges Problem folgendermaßen gelöst werden. Als einziger Unterschied ergibt sich die Syntax, die Logik bleibt die selbe.

```
<?php
if ($name == "Fred")
      echo 'Willkommen der Herr.';
elseif ($name == "Wilma")
      echo 'Willkommen die Dame.';
else
      echo 'Willkommen, was auch immer Du sein magst.';
?>
```

Code-Beispiel 4-7: Bedingte Ausgabe mittels PHP-Code

Umgeben von den vielen Schablonenverarbeitungssystemen vergisst man gerne, dass PHP selbst nichts anderes als eine Template Engine ist. Die HTML-Schablonen enthalten durch „<?php“ und „?>“ gekennzeichnete Platzhalter, die vom Interpreter erkannt und durch HTML-Code ersetzt werden. Damit stellt sich die Frage, ob die Notation einer Template Engine z. B. für einen dynamischen Titel wirklich um so vieles einfacher ist als jene von PHP?

`<h1>{$title}</h1>` ⟷ `<h1><?php echo $title;?></h1>`

PHP ist leicht zu erlernen, auch für das Designteam. Zumindest ist der Aufwand, um Konstrukte wie Schleifen und Verzweigungen zu erarbeiten nicht höher, als er es für

[10] Dieser Code stammt aus dem Beispiel 7-11 des Kapitels 7 im Smarty-Handbuch

die Smarty-Notation ist. Für das Programmierteam bedeutet es ein Wegfallen der zusätzliche Einarbeitung und Wartung von Smarty. Der wesentliche Vorteil für den Benutzer der Anwendung ergibt sich aus der besseren Performanz, da die Arbeitsschritte der Template Engine nicht ausgeführt werden müssen.

Abgesehen davon, dass eine perfekte Trennung von Anwendungslogik und Präsentation mit Hilfe von Schablonenverarbeitungssystemen nicht möglich ist, fehlt ferner eine Trennung zu den Daten der Anwendung. Diese sind nach wie vor mit den Präsentationsaspekten vermischt, da HTML-Elemente keine Auskunft über deren Inhalte geben können. Weil sich aber Template Engines ausschließlich mit der Trennung von PHP- und HTML-Code beschäftigen, können sie keine Lösung für die gewünschte Architektur nach dem Separation of Concerns-Paradigma bieten. In [Fuec02b] ist eine ausführliche Argumentation zum Thema Template Engines und PHP zu finden.

(http://smarty.php.net/manual/de/language.function.if.php)

4.2.2 MVC Frameworks in PHP

PHP wurde ursprünglich als Skriptsprache entworfen und hat den Ruf, leicht erlernbar zu sein und einen großen Funktionsumfang zu bieten. Dynamische Inhalte können mittels PHP oft auf sehr einfache Weise angeboten werden. Zahlreiche Tutorien erklären die wichtigsten Funktionalitäten für Web-Anwendungen (darunter Datenbankzugriff, Mail-Versand, Dateiupload, Sitzungsmanagement, etc.). Diese Einfachheit führt aber oft zu schwer wartbarem, mehrfach dupliziertem PHP-Code, der in den HTML-Code eingestreut wird. Diesem Durcheinander kann allerdings mit ein wenig Disziplin leicht Abhilfe verschafft werden, indem die Entwickler den Code in Funktionen und eigenen Dateien zu wieder verwendbaren Bibliotheken zusammenfassen. Gilt es jedoch große, interaktive und datenintensive Web-Anwendungen, die womöglich auch Alt-Systeme einbinden sollen, zu entwickeln, dann ist ein gut durchdachtes Design notwendig. Spätestens dann müssen und werden die EntwicklerInnen sich Gedanken über objektorientierte Programmierung in PHP und den Aufbau und mögliche Komponenten der Web-Anwendung machen. Harry Fuecks startete Ende 2002 die Initiative „Patterns in PHP“ (www.phppatterns.com), um die Akzeptanz von PHP für Enterprise-Applikationen zu steigern, auf die objektorientierten Möglichkeiten von PHP hinzuweisen und den Einsatz von Patterns in der Entwicklung mit PHP zu fördern. Artikel und Diskussionen zu Patterns und deren Anwendbarkeit für PHP findet man aber nicht nur unter www.phppatterns.com, sondern auch in zahlreichen PHP-Foren. Als Denkanstoß für viele Diskussionen dienen die J2EE Patterns[11]. Besonders gerne wird das MVC Pattern diskutiert[12] und auch angewendet. Viele in PHP geschriebene Frameworks implementieren das MVC Pattern bzw. ziehen das bekannte Java-Framework Struts (http://struts.apache.org/) als Design-Vorbild heran.

Dieser Abschnitt befasst sich mit PHP Frameworks, die das MVC Pattern implementieren. Leider hat sich unter den Frameworks kein klarer Führer wie z. B. Smarty unter den Schablonenverarbeitungssystemen herausgestellt. Oft stehen hinter einem Projekt nur eine oder wenige Personen, was es schwierig macht ein Framework als

---

[11] Siehe http://java.sun.com/blueprints/patterns/index.html

zukunftsträchtig einzuschätzen. Finden sich nicht genug Interessenten, dann kann auch ein sehr gutes Projekt nach kurzer Zeit „einschlafen“. Für die EntwicklerInnen bedeutet das, dass Dokumentation, Forum und Wiki wichtige Entscheidungskriterien sind, falls man sich Support und Weiterentwicklung eines Frameworks erwartet.

Drei bekannte Frameworks wurden im Rahmen dieser Arbeit untersucht, wobei Phrame (http://phrame.sourceforge.net/) detailliert, PHP-MVC[13] und Ambivalence[14] eher oberflächlich betrachtet wurden. Eine genaue Vorstellung der einzelnen Frameworks ist im Rahmen dieser Arbeit nicht möglich und es wird auf die Dokumentation der einzelnen Projekte verwiesen. Abbildung 4-2 orientiert sich an Phrame und stellt schematisch die grundlegenden Komponenten und Abläufe eines MVC-Frameworks dar.

Im wesentlichen wird die Controller-Komponente durch das Framework vorgegeben, während die View- und die Model-Komponenten *relativ* frei gestaltet werden können.

12 Siehe http://www.sitepoint.com/forums/showthread.php?t=154866&page=1&pp=25

13 Siehe http://www.phpmvc.net/

14 Siehe http://amb.sourceforge.net/

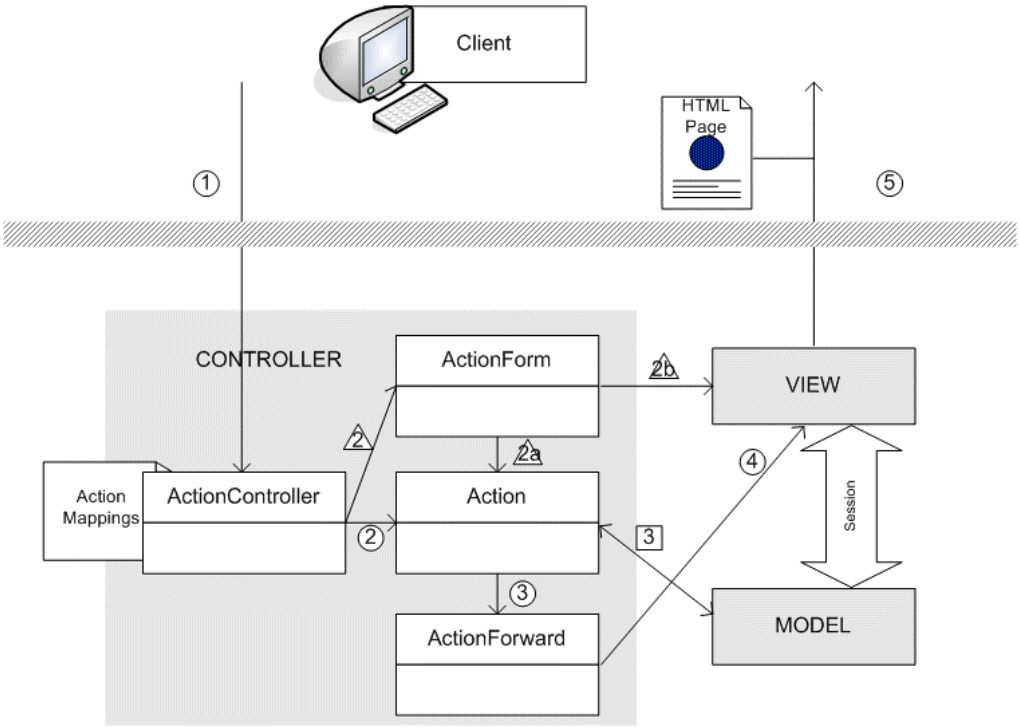

Abbildung 4-2: Architektur der MVC Frameworks in PHP

Der grundlegende Ablauf einer Anfrage kann in folgenderweise zusammengefasst werden: Ein *ActionController*-Objekt erhält eine Anfrage ① und ermittelt die durchzuführende *Action*. Falls Eingabedaten des Benutzers vorhanden sind ②, werden diese zuerst über das *ActionForm*-Objekt geprüft. Im Falle eines Fehlers wird die Verantwortung gleich an den entsprechenden View übergeben ②b. Sind die Eingabedaten gültig, kann die Action ausgeführt werden ②/②a und über das *ActionForward*-Objekt ③ der nächste View bestimmt werden. Das Action-Objekt führt die Anwendungslogik über Aufrufe des Models aus [3]. Die Verbindung zwischen View und Model wird über eine Session hergestellt.

Ein Beispiel

Jedes Framework besitzt eine Konfigurationsdatei (vgl. Abbildung 4-2: Action Mappings), in welcher für jede mögliche logische Anfrage eine Action definiert wird, die diese Anfrage bearbeitet. Der Ablauf einer Anfrage soll nun nochmals mit Hilfe einer kleinen Beispielkonfigurationsdatei dargestellt werden (siehe Code-Beispiel 4-8).

```
<?php
$mappings = array(
      _ACTION_FORMS => array(
            'form' => array(
                  _TYPE => 'HelloForm'
            )
      ),
      _ACTION_MAPPINGS => array(
            'sayHello' => array(
                  _TYPE => 'HelloAction',
                  _NAME => 'form',
                  _INPUT => 'index.php',
                  _VALIDATE => 1,
                  _ACTION_FORWARDS => array(
                        'hello' => array(
                              _PATH => 'hello.php',
                              _REDIRECT => 0
                        ),
                        'index' => array(
                              _PATH => 'index.php',
                              _REDIRECT => 0
                        )
                  )
            )
      )
);
?>
```

Code-Beispiel 4-8: Konfigurationsdatei in Phrame

Die dazugehörige Beispielanwendung besitzt ein Eingabefeld für den Namen des Benutzers (vgl. Abbildung 4-3), welcher dann nach dem Abschicken des Formulars angezeigt werden soll.

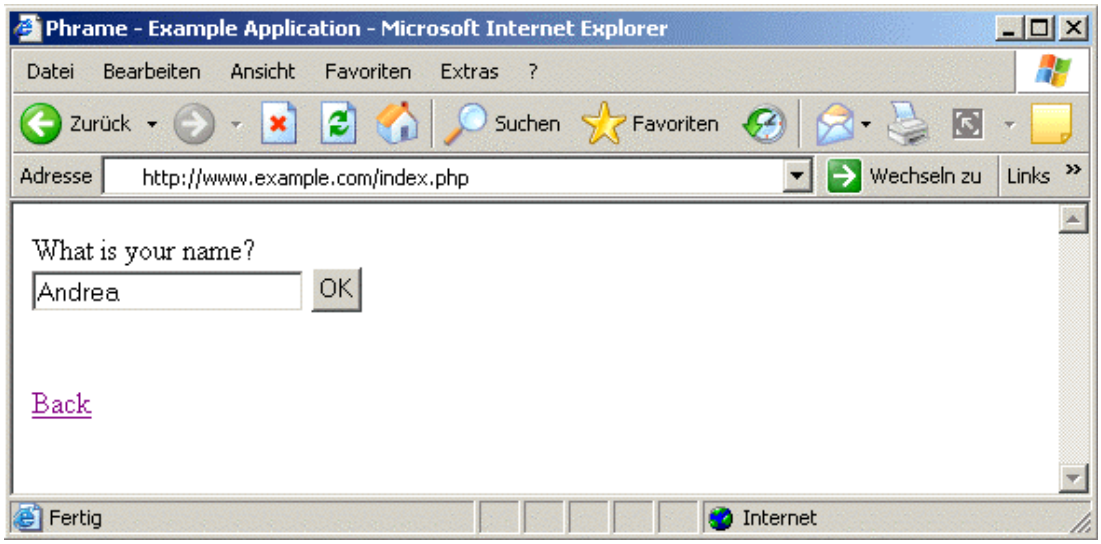

Abbildung 4-3: Phrame-Beispiel, Anzeige des Eingabeformulars

Das ausgefüllte Formular wird an ein PHP-Skript geschickt, welches ein ActionController-Objekt aufruft. Im Formular muss entweder über dir URL oder durch ein verstecktes Eingabefeld die gewünschte Action übermittelt werden, in diesem Fall „sayHello“ (vgl. Code-Beispiel 4-8).

Der ActionController erkennt anhand des „_NAME“-Eintrags, dass Eingabedaten des Benutzers übermittelt wurden. Er sucht im „_ACTION_FORMS“-Array nach dem Eintrag „form“ und übergibt die Verantwortung an das HelloForm-Objekt. Dieses Objekt ist für die Überprüfung der Eingaben zuständig und wird vom Entwickler bereitgestellt. War die Überprüfung erfolgreich, dann kann die tatsächliche Action (HelloAction) ausgeführt werden, sonst wird an den View übergeben der unter dem Eintrag „_INPUT“ zu finden ist. In Abbildung 4-4 ist erkennbar, dass wieder das Eingabeformular „index.php“ mit einer zusätzlichen Fehlermeldung angezeigt wird. Der Benutzer hatte vergessen einen Namen in das entsprechende Feld einzugeben.

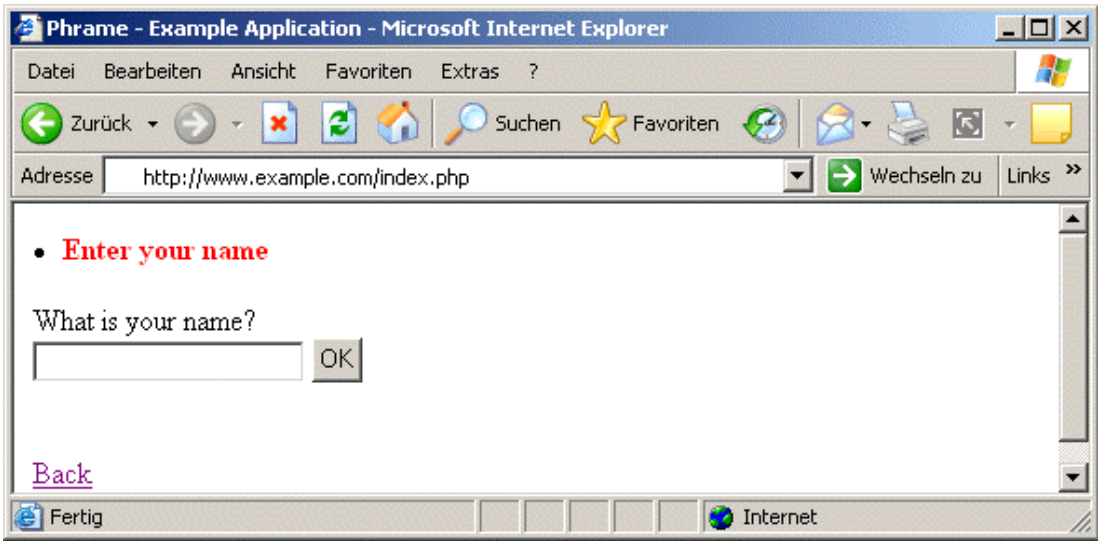

Abbildung 4-4: Phrame-Beispiel, Anzeige der Fehlermeldung

Innerhalb der HelloAction, die ebenfalls vom Entwickler bereitgestellt wird, muss der nächste View bestimmt werden. In der Konfigurationsdatei findet man im „_ACTION_FORWARDS“-Array die Einträge „hello“ und „index“. Falls während der Ausführung der Action noch ein Fehler aufgetreten sein sollte, dann wird an den „index“-View, sonst an den „hello“-View übergeben, der den Benutzernamen ausgibt (vgl Abbildung 4-5).

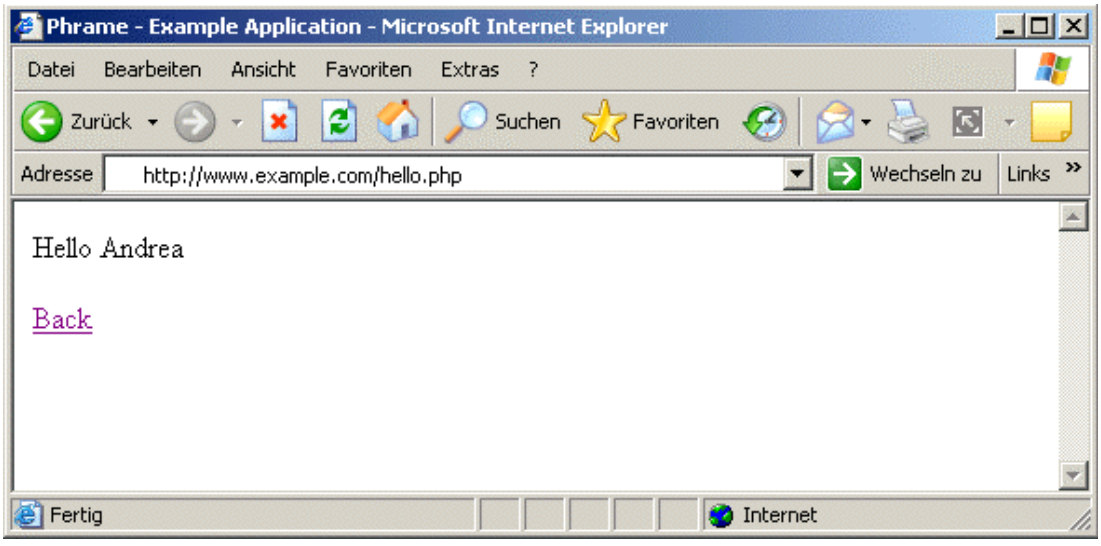

Abbildung 4-5: Phrame-Beispiel, „Hello“-Anzeige

Eigenschaften der untersuchten Frameworks

Die untersuchten Frameworks haben alle gemein, dass sie das Template View Pattern unterstützen. Entweder werden HTML-Templates mit kleinen PHP-Befehlen gespickt oder es wird sogar Unterstützung für eine ausgereifte Template Engine, meist Smarty, angeboten. Diese Vorgehensweise wird von den EntwicklernInnen scheinbar bevorzugt[15]. Warum viele PHP-Entwickler nicht bereit sind XML zu verwenden, liegt möglicherweise an den eher schlechten XML-Erweiterungen vor PHP 5 (vgl. Abschnitt 4.2.3) bzw. daran, dass XSLT (eine funktional Sprache) für jemanden der prozedurales Programmieren gewöhnt ist, schwieriger zu erlernen ist.

Als nachteilig sind außerdem folgende Eigenschaften der Frameworks zu bewerten:

- Ambivalence ist das einzige Framework das Authentifizierung und Autorisierung anbietet. Für PHP-MVC und Phrame sind diese Punkte erst in Planung.
- Die Überprüfung von Benutzereingaben wird in Ambivalence gar nicht in Phrame und PHP-MVC nur auf Formularebene (ActionForms) unterstützt. Eine feinere Granulierung wäre hier wünschenswert. Im Sinne der Wiederverwendbarkeit von Code ist eine Validierung von Benutzereingaben auf Ebene von einzelnen Eingabefeldern bzw. zusammenhängenden Eingabenfeldern zu bevorzugen (vgl. Abschnitt 5.2.3).
- Die detaillierte Untersuchung von Phrame hat hervorgebracht, dass die Weiterleitung einer Anfrage von einem Action-Objekt an den nächsten View über einen HTTP-Redirect[16] geschieht. Dies bedeutet eine doppelte Anzahl von Anfragen an den Webserver und ein dementsprechender Performanzverlust. Über die Weiterleitung in Ambivalence und PHP-MVC kann aufgrund der weniger ausführlichen Untersuchung keine Aussage getroffen werden.

---

[15] Siehe http://www.sitepoint.com/forums/showthread.php?t=123769, http://www.sitepoint.com/forums/showthread.php?t=95679 und http://www.peterrobins.co.uk/it/mojavi/tutorial.htm

[16] Der Client erhält als Antwort vom Webserver eine URL anstatt einer HTML-Seite. Diese URL verwendet er, um erneute eine Anfrage an den Webserver zu stellen.

4.2.3 Das XML Framework XAO

XML Support gab es in PHP bereits in frühen Tagen ab Version 3 durch die XML Parser Funktionen, die ein ereignisbasiertes Parsen von XML-Daten erlaubten. In PHP 4 kam es dann zu einer besseren Unterstützung durch die DOM XML- und die XSLT-Erweiterung. Leider hat die DOM XML-Erweiterung ihren experimentellen Status nie verlassen und wurde demnach in vielen Installationen erst gar nicht aktiviert. Die Entwickler waren mit einer dem W3C Standard (http://www.w3c.org/) nicht entsprechenden, sich mehrmals ändernden API und unvollständiger Funktionalität konfrontiert. In Version 4.3 wurde versucht diese Mängel zu beheben und die Methodennamen weitgehend an den Standard anzupassen. Einige grundlegendere Probleme konnten offenbar nicht behoben werden, denn für PHP 5 wurde eine völlige Überarbeitung des XML Supports nach dem W3C Standard DOM beschlossen. Die im PHP-Handbuch unter dem Kapitel „DOM Funktionen“ zu findende neue Erweiterung ersetzt somit die alten Funktionen des Kapitels „DOM XML Funktionen“ [ABD+04].

In PHP 5 hinzugekommen, ist die Möglichkeit XML-Dokumente gleich auf drei Arten zu validieren, nämlich mit DTDs, XML Schemas oder RelaxNG. Die jüngste Erweiterung in der Familie der XML-Funktionen in PHP ist die SimpleXML API. Diese Erweiterung ist einfacher in der Handhabung als DOM und um nicht viel weniger mächtig. SimpleXML basiert wie DOM auf der libxml2-Bibliothek, was eine Konvertierung von SimpleXML-Objekten in DomDocument-Objekte und umgekehrt erlaubt [Stock04].

XSLT ist eine Sprache zur Transformation von XML-Dokumenten in andere XML-Dokumente und ist ebenfalls ein W3C Standard. In PHP 4 stehen den EntwicklerInnen zwei XSLT-Prozessoren zur Verfügung. Dabei genießt der Sablotron-Prozessor einen höheren Bekanntheitsgrad und wird öfters eingesetzt als der in die DOM XML-Erweiterung integrierte libxslt-Prozessor. Letzterer verursacht schon bei einfachen Transformationen Probleme. In PHP 5 wurde vorerst nur der libxslt-Prozessor weiter bzw. neu entwickelt, weil dieser wie auch die DOM-Funktionen auf der libxml2-Bibliothek basiert. Als Detail am Rande sei erwähnt, dass der neue Prozessor beinahe

doppelt so schnell arbeitet, als es mit dem Sablotron-Prozessor in PHP 4 möglich ist [Stock04].

XAO Framework

Das XML Application Objects Framework (http://xao-php.sourcefourge.net) stellt eine einfache API für die Arbeit mit XML in PHP 4 dar. Diese API umfasst einige grundlegende Funktionen, um ein PHP-DomDocument-Objekt bearbeiten zu können, darunter auch eine Funktion zum Transformieren der XML-Daten mit XSLT. XAO implementiert somit das Transform View Pattern (vgl. Unterkapitel 3.5). Im Kontext des MVC Patterns implementiert XAO die View Komponente in Form der XSL-Stylesheets und die Controller Komponente in Form eines PHP-Skripts, das die Logik für den Aufbau der XML-Daten enthält.

Besonders hervorzuheben sind nachfolgende Eigenschaften des Frameworks

- Die Fehlerbehandlung von XAO ist nahezu vorbildlich gestaltet. Im Falle eines Fehlers erhält man eine farblich unterstütze HTML-Ausgabe. Viel wichtiger ist aber die Aussagekraft der Fehlermeldung. So erhält man nicht nur Klasse, Methode und Codezeile, sondern auch den gesamten „Stack Trace“ (die Folge der Methodenaufrufe). Erwähnenswert ist auch der Debug-Modus des Frameworks, durch den man die Transformation der XML-Daten mit einem Stylesheet vorübergehend ausschalten kann. Durch Eingabe eines bestimmten Parameters in der URL wird nur das XML-Dokument im Browser angezeigt (vgl. Abbildung 4-8).
- XAO ist vollkommen objektorientiert und schafft durch die einfache API eine Abstraktion von den DOM XML-Funktionen und den XSLT-Funktionen in PHP. Dadurch soll wiederum ein einfacher Umstieg auf PHP 5 gewährleistet werden, sofern man das DomDocument-Objekt nur über die API von XAO nicht aber über die DOM XML-Funktionen von PHP anspricht.

Die wichtigsten Klassen des Frameworks sind in Abbildung 4-6 zu sehen und werden in Tabelle 4-1 in kurzer Form beschrieben.

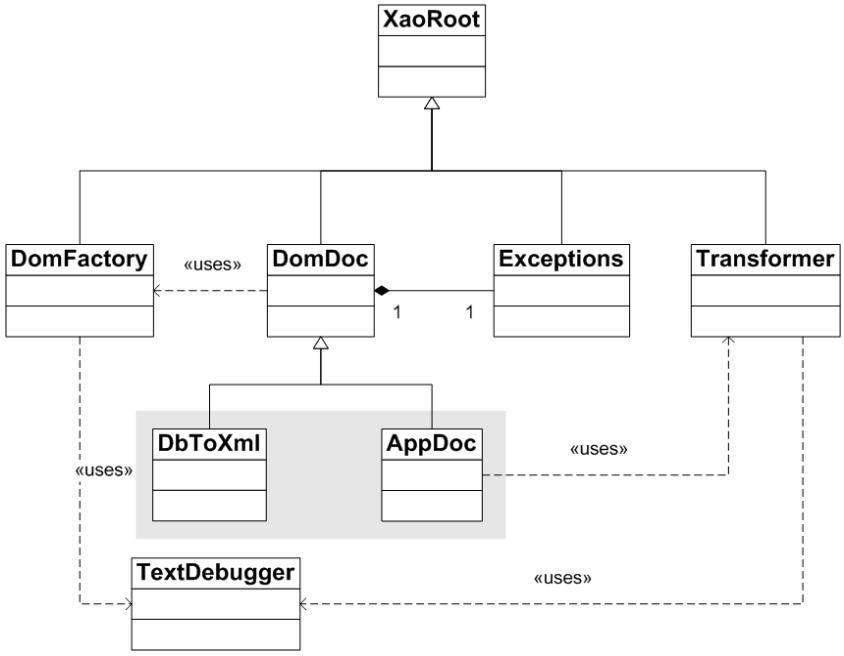

Abbildung 4-6: XAO Klassendiagramm

XaoRoot	Diese Klasse stellt die Basisklasse dar, von der vererbt wird. Sie dient der Zentralisierung, damit Eigenschaften und Methoden, die von allen Klassen des Frameworks benötigt werden, an einer zentralen Stelle hinzugefügt werden können. Die Fehlerbehandlung ist beispielsweise an dieser Stelle implementiert.
DomDoc	Diese Klasse enthält ein DomDocument-Objekt und bietet Methoden, um dieses bearbeiten zu können. Beispielsweise können XML-Daten über Dateien, Strings oder Referenzen auf andere DomDocument-Objekte dem XML-Baum hinzugefügt werden.
Exceptions	Ein Objekt dieser Klasse wird von DomDoc verwendet, falls Fehler während der Verarbeitung aufgetaucht sind. Das Framework bricht nicht ab, sondern fügt die Fehlermeldungen in den XML-Baum ein. Das Framework liefert ein Stylesheet mit, welches das XML-Fragment in eine HTML-Ausgabe konvertiert. Dieses Stylesheet kann zu Debugging-Zwecken in die Stylesheets der Anwendung eingebunden werden. Im Produktivbetrieb sollte es aber aus Sicherheitsgründen durch ein Stylesheet ersetzt werden, dass weniger Informationen über die aufgetretenen Fehler preisgibt.

Transformer	Die Transformation der XML-Daten übernimmt die Transformer-Klasse. In der Regel erhält diese Instanz Referenzen auf die XML-Daten und das zu verwendende Stylesheet. Es wird zwar die Möglichkeit geboten mit dem libxslt-Prozessor zu arbeiten, die Voreinstellung wählt aber den Sablotron-Prozessor für die Transformation aus.
DomFactory	DomDoc benutzt diese Klasse, um ein DomDocument-Objekt aus verschiedensten Quellen (z. B. aus einer Datei oder einer Variable) erzeugen zu können.
TextDebugger	Diese Klasse wird als einzige nicht von XaoRoot abgeleitet. Sie erzeugt aus Fehlermeldungen, wie das Exceptions-Stylesheet, eine HTML-Ausgabe. Falls beim Erzeugen eines DomDocument-Objekts (vgl. DomFactory) oder beim Transformieren im Stylesheet ein Fehler aufgetreten ist, dann kann die Fehlerausgabe nicht mehr über ein Exceptions-Objekt erfolgen. Es ist dann entweder kein DomDocument vorhanden, welches die XML-Fehlermeldungen aufnehmen kann, oder die Transformation der XML-Daten ist nicht möglich.
AppDoc	Die wichtigste Klasse des Frameworks ist die AppDoc Klasse. Sie erweitert DomDoc um die Möglichkeit XML-Daten zu transformieren und das Ergebnis der Transformation an den Browser zu schicken. Dazu verwendet sie ein Transformer-Objekt. Der Debug-Modus kann in dieser Klasse eingeschaltet werden.
DbToXml	Sollen in einer Anwendung die Ergebnisse von Datenbankabfragen ausgegeben werden, dann müssen diese erst in das XML-Format konvertiert werden. DbToXml übernimmt diese Aufgabe und packt die Daten in ein DomDocument-Objekt.

Tabelle 4-1: Klassen des XAO Frameworks

Die Funktionsweise des Frameworks soll mit Hilfe eines einfachen Beispiels demonstriert werden. Die Inhalte der Webseite sind statisch und stammen aus einer XML-Datei (HelloWorld.xml). Das Ausgabegerät ist ein gewöhnlicher Browser, weshalb die Inhalte mit Hilfe eines entsprechenden Stylesheets (vgl. Code-Beispiel 4-10) nach HTML transformiert werden müssen. Das PHP-Skript in Code-Beispiel 4-9 zeigt den Controller.

```
<?php
require_once 'xao/XAO_AppDoc.php';

//Variablendeklarationen
//Erzeugen des Application Objects
$strDocRoot          = 'doc'; //Wurzelelement
$objAppDoc           = new AppDoc($strDocRoot);
$objAppDoc->blnDebug = TRUE;

//Einlesen und bearbeiten der XML Daten
//durch Datenbankabfragen und Benutzereingaben
$objAppDoc->ndConsumeFile('HelloWorld.xml');

//Auswahl eines Stylesheets Transformation
//Senden der Ergebnisse an den Benutzer
$objAppDoc->ndSetStylePI('page2html.xsl');
$objAppDoc->Transform();
$objAppDoc->Send();
?>
```

Code-Beispiel 4-9

Das AppDoc-Objekt enthält das PHP-DomDocument-Object und bietet verschiedenste Methoden, um die XML Daten zu bearbeiten. In diesem Fall soll nur der Inhalt der XML-Datei eingelesen und unterhalb des Wurzelelements „doc“ eingefügt werden. Prinzipiell könnten aber zusätzliche Elemente dynamisch in den XML-Baum eingebaut werden. Zuletzt werden die XML-Daten mit Hilfe eines XSL-Stylesheets transformiert und das Ergebnis an den Browser geschickt. Die Ausgabe ist in Abbildung 4-7 zu sehen.

Abbildung 4-7: Aus XML generierte HTML-Ausgabe mit XAO

Für dieses Beispiel wurde außerdem der Debug-Modus eingeschaltet (siehe Code-Beispiel 4-9). Durch Eingabe des Parameters „xao:XML“ in die URL, wird die Transformation der Daten umgangen und die Ausgabe im Browser ist reines XML. Die nächste Abbildung zeigt somit den im Skript aufgebauten XML-Baum an, der in diesem Fall gleich dem Inhalt von „HelloWorld.xml“ ist.

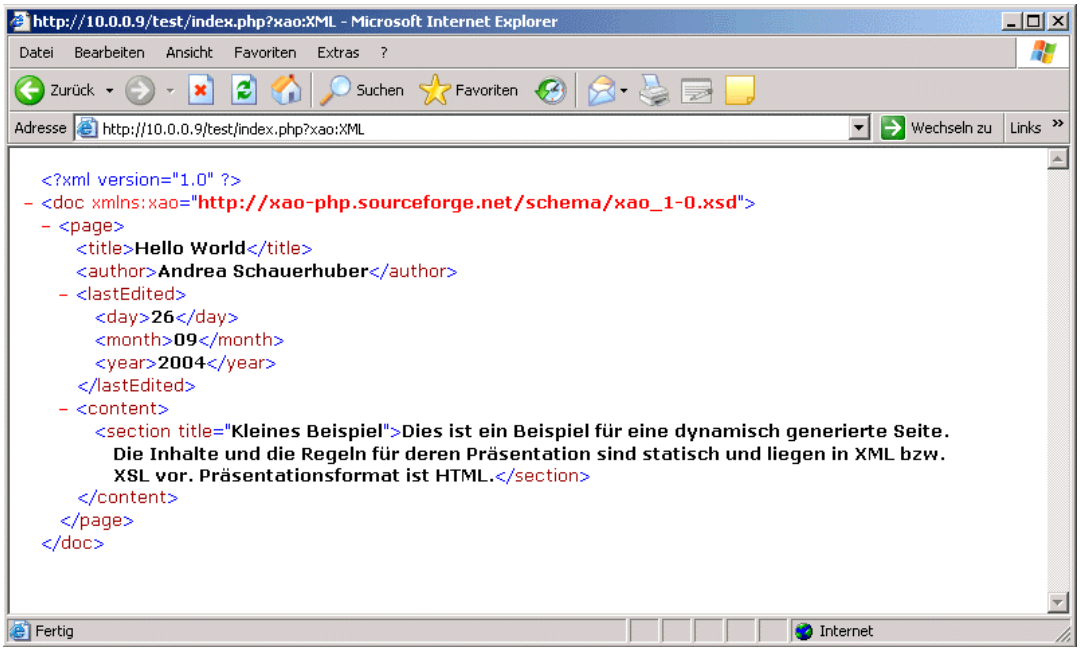

Abbildung 4-8: XML-Ausgabe im Debug-Modus von XAO

Um die Ausgabe in Abbildung 4-7 erzeugen zu können, wurde folgendes XSL-Stylesheet für die Transformation der XML-Daten verwendet.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl ="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>

   <xsl:template match="/doc">
      <html>
         <head><title><xsl:value-of select="page/title"/></title>
         </head>
         <body>
            <xsl:apply-templates/>
            <p><em>
                  <xsl:value-of select="page/author"/>
                  <br/>Letzte Änderung:
                  <xsl:value-of select="page/lastEdited/day"/>
                  <xsl:value-of select="page/lastEdited/month"/>.
                  <xsl:value-of select="page/lastEdited/year"/>
            </em></p>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="page">
      <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="content">
      <h1><xsl:value-of select="../title"/></h1>
      <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="section">
      <h2><xsl:value-of select="@title"/></h2>
      <p><xsl:apply-templates/></p>
   </xsl:template>
    <xsl:template match="*"/>
</xsl:stylesheet>
```

Code-Beispiel 4-10: XSL-Stylesheet

4.2.4 Zusammenfassung & Vergleich

Wie bereits in Unterkapitel 3.5 beschrieben, bieten sich für die Implementierung der View-Komponente des MVC Patterns zwei Möglichkeiten an. Die Frage, ob nun das Template View Pattern oder das Transform View Pattern die bessere Lösung im Sinne der Separation of Concerns ist, spaltet die PHP-Entwicklergemeinde in zwei Meinungsgruppen. Die Menge an Template Engines und Frameworks, die wiederum den Einsatz von Template Engines unterstützen, zeigt jedoch eher eine Vorliebe für das Template View Pattern. In dieser Arbeit wird dagegen der Standpunkt vertreten, dass nur durch das Transform View Pattern eine vollständige Trennung von Inhalten, Logik und Präsentation erreicht werden kann. Damit scheiden Template Engines und die vorgestellten MVC-Frameworks als Möglichkeit aus und die Entscheidung fällt zugunsten des Einsatzes der XML-Standards mit Hilfe des XAO Frameworks.

Kapitel 5

5 Architektur-Framework SAN

In diesem Kapitel werden Entscheidungen bezüglich der Architektur und des Designs dokumentiert. Die wichtigsten Anforderungen an die Architektur des implementierten Institutsinformationssystems und die eingesetzten Patterns werden besprochen und die Architektur anhand von Grafiken und Code-Beispielen erläutert. Unter anderem enthält das Unterkapitel 5.2 das Input Validator Pattern, einen Pattern-Vorschlag zur zentralen, konsistenten Eingabeüberprüfung in Web-Anwendungen. Den Abschluss bildet ein detaillierter Einblick in das entwickelte SAN Framework, insbesondere für die Anwendung der PhpServlet-Klasse, einer Implementierung des Front Controller Patterns.

5.1 Überlegungen zur Architektur

Zu Beginn dieser Arbeit wurden drei wesentliche Anforderungen an die Architektur des zu entwickelnden Institutsinformationssystems gestellt:

- Separation of Concerns: Die sich bereits aus dem Titel dieser Arbeit ergebende wichtigste Anforderung an die Architektur ist die Trennung von Inhalten, Anwendungslogik und Präsentation.
- Sicherheit: Das Thema der Arbeit in [Jova04] fließt in die Architekturüberlegungen derart ein, dass die Web-Anwendung private Bereiche mit Zugriffsbeschränkungen besitzt, also Authentifizierung und Autorisierung der Benutzer

erforderlich ist. Von besonderer Bedeutung ist außerdem die Überprüfung von Benutzereingaben. Eine durchgängige, zentrale Kontrolle der Eingaben ist unerlässlich.

- Konfigurier- & Wartbarkeit: Der wichtigste Punkt im Rahmen dieser Anforderung ist Zentralisierung. Darunter fallen einerseits die Vergabe von Zugriffsberechtigungen und andererseits eine Abstraktion vom Dateisystem durch die Verwendung von logischen Namen. Als mögliche Lösung bietet sich eine Konfigurationsdatei an, die zu jeder logischen Webseite den Dateisystempfad zur geforderten Ressource und die berechtigten Benutzergruppen speichert. Darüber hinaus soll eine Abstraktionsschicht zum verwendeten, persistenten Datenspeicher (in der Regel ein relationales Datenbankmanagementsystem) eingeführt werden, um von diesem unabhängig zu bleiben.

5.2 Eingesetzte Patterns

Die im Sinne der Separation of Concerns möglichen Architekturen, Patterns und Frameworks wurden bereits in früheren Kapiteln ausführlich beschrieben. Die LAMP-Architektur ist eine typische 2-Schichten-Architektur (vgl. Unterkapitel 2.2). Nach [Cona00] folgt die entwickelte Web-Anwendung der Thin-Web-Client-Architektur, wonach jede Logik serverseitig ausgeführt wird und der Client Informationen ausschließlich in einem bestimmten Format (z. B. reines HTML) erhält.

5.2.1 Model View Controller

Die entwickelte Web-Anwendung implementiert außerdem das Model View Controller Pattern, um die Benutzerschnittstelle von der Anwendungslogik trennen zu können.

- **View**: Durch den Einsatz des in Abschnitt 4.2.3 beschriebenen XAO Frameworks wird die View-Komponente nach dem Transform View Pattern umgesetzt. Damit können die einzelnen XSL-Stylesheets als die verschiedenen Sichten der Anwendung angesehen werden.
- **Model**: [Fowl03a] teilt seine Patterns unter anderem in die Kategorien Domain Logic und Data Source. Das Table Module Pattern gehört der ersten Kategorie

an und wurde, wie auch das Table Data Gateway Pattern, bereits in Unterkapitel 3.5 erläutert. Das Table Data Gateway ist in die Kategorie Data Source einzuordnen. Es lässt sich darüber streiten, welches Pattern für die Umsetzung der Model-Komponente verwendet wurde. Ein Table Data Gateway kapselt alle SQL-Anweisungen innerhalb einer Klasse. Ein Table Module wiederum kapselt darüber hinaus die Anwendungslogik für eine Datenbanktabelle [Fowl03a] und wäre damit die plausiblere Antwort. Tatsächlich hat die entwickelte Web-Anwendung kaum Anwendungslogik im eigentlichen Sinn. Es müssen keine komplizierte Berechnungen angestellt werden, wie es beispielsweise bei E-Commerce-Anwendungen der Fall ist. Im Gegenteil handelt es sich, vereinfacht gesprochen, um ein Informationssystem, das durch Benutzerinteraktionen Informationen aufnimmt und ablegt, und auf Anfrage wieder anzeigt. Die Model-Klassen der Anwendung sind dadurch großteils Implementierungen des Table Data Gateway Patterns[17]. Für (fast) jede Tabelle wurde eine Klasse entwickelt.

- **Controller**: Der Controller übernimmt eine Anfrage, überprüft die Eingaben des Benutzers mit dem Input Validator (vgl. Abschnitt 5.2.3) und ruft die Methoden der Model-Klassen auf. Im Gegensatz zu Phrame und den anderen Frameworks aus Abschnitt 4.2.2 wird die Verbindung zwischen View und Model nicht über eine Session hergestellt. Im Controller wird ein AppDoc-Objekt des XAO Frameworks erzeugt und die Ergebnisse der Model-Aufrufe werden in den XML-Baum eingebunden. Als Verbindungselement dienen also die XML-Daten.

5.2.2 Front Controller

Das Front Controller Pattern wurde im Sinne der Konfigurier- und Wartbarkeit implementiert. Der Front Controller erhält über die URL einen logischen Namen für die gewünschte Aktion und entscheidet anhand einer Konfigurationsdatei, welche Datei (bzw. Klasse) für die Ausführung geladen werden muss. Ebenfalls in der Konfigura-

[17] Beispiel für die wenigen Methoden, die Anwendungslogik implementieren, sind die Methoden die ein eingegebenes Passwort für ein Benutzerkonto validieren.

tionsdatei enthalten sind die berechtigten Benutzergruppen für eine Aktion. Falls notwendig, führt der Front Controller dann Authentifizierung und Autorisierung der Benutzer durch.

5.2.3 Input Validator

Wie bereits in Unterkapitel 4.2 angemerkt wurde, sind Überprüfungen von Benutzereingaben meist auf Formularebene angesiedelt. Viele Frameworks, so auch Phrame, stellen dafür ein ActionForm-Objekt zur Verfügung, welches vor der eigentlichen Verarbeitung der Action aufgerufen wird. Bei dieser Vorgehensweise wird vorgeschrieben, dass alle Eingaben des Benutzers überprüft werden, bevor die Action ausgeführt wird. In dieser Arbeit wird ein etwas flexiblerer Ansatz vorgeschlagen.

Formulare oder Parameter die in der URL übergeben werden, stellen beliebte Ausgangspunkte für Angriffe dar [Jova04]. Umso wichtiger ist eine Überprüfung der Benutzereingaben. Die sicherste Vorgehensweise ist die Inputs auf ein gültiges Schema hin zu kontrollieren. So gibt es unterschiedliche Typen von Eingaben:

- Text,
- Zahlen,
- Zeichenketten aus einem bestimmten Pool (diskrete Werte)
- oder Zeichenketten, die mit Hilfe von regulären Ausdrücken überprüft werden können (E-Mail, Matrikelnummer, etc.).

Inputs können über ein Formular oder über einen URL-Parameter an den Server geschickt werden.

Im Rahmen der praktischen Arbeit wurde die Überprüfung der Benutzereingaben auf Ebene von einzelnen und zusammenhängenden Eingaben angedacht (im Folgenden Inputs bzw. InputSets). Das daraus abgeleitete Input Validator Pattern enthält die folgenden vier Komponenten:

- Ein *Input* entspricht einem Eingabefeld im Formular (Textfeld, Auswahlboxen, etc.). Ein Input kann obligatorisch oder optional sein. Er muss aber überprüft werden, wenn er entweder obligatorisch oder eine Eingabe des Benutzers vorhanden ist.

- Ein *InputSet* besteht aus mehreren Inputs. In vielen Web-Anwendungen gibt es beispielsweise die Möglichkeit sich registrieren zu lassen. Um sicher zu gehen, dass sich der Benutzer bei der Eingabe seines Passworts nicht vertippt hat, muss dieser das Passwort in einem zweiten Eingabefeld bestätigen. Für diesen Anwendungsfall wird zuerst jedes einzelne Passwort überprüft (z. B. auf Mindestlänge). Die Passwörter müssen außerdem identisch sein, weshalb eine zusätzliche Überprüfung für die zusammenhängenden Inputs durchgeführt wird. InputSets und auch deren zugehörige Inputs können obligatorisch oder optional sein. Das InputSet wird validiert, wenn es obligatorisch ist oder mindestens ein Eingabefeld des Sets vom Benutzer ausgefüllt wurde. Dann werden zuerst die einzelnen Inputs überprüft und danach die zusätzlichen Kontrollen für die Abhängigkeiten zwischen den Inputs durchgeführt.
- Der *InputValidator* ist ein Objekt, bei dem die Inputs und InputSets für eine Aktion registriert werden. Je nach Typ (Zahl, regulärer Ausdruck, etc.) des Inputs bzw. InputSets wird ein bestimmtes Objekt beim InputValidator bekannt gegeben (vgl. Abbildung 5-1). Der InputValidator stößt die Überprüfung an und validiert alle registrierten Objekte. Die Validierungsfunktionen der Inputs und InputSets haben die Möglichkeit Fehlermeldungen für die jeweilige Eingabe im *InputError*-Objekt, das jeder InputValidator besitzt, zu speichern. Die Überprüfung sollte beim Auftauchen der ersten fehlerhaften Eingabe allerdings nicht abgebrochen werden. Das Skript bzw. Objekt, das den InputValidator erzeugt hat, kann im Falle eines Fehlers auf die Fehlermeldungen im InputError-Objekt zurückgreifen und diese ausgeben. Der Vorteil für den Benutzer ist, dass immer alle Eingabefehler in einem Formular angezeigt werden.
- Jeder InputValidator besitzt ein *InputError* Objekt, dass alle Fehlermeldungen der Inputs und InputSets enthält. Es wäre sinnvoll einen „Fehlertitel“, der das betroffene Eingabefeld angibt, mitzuspeichern.

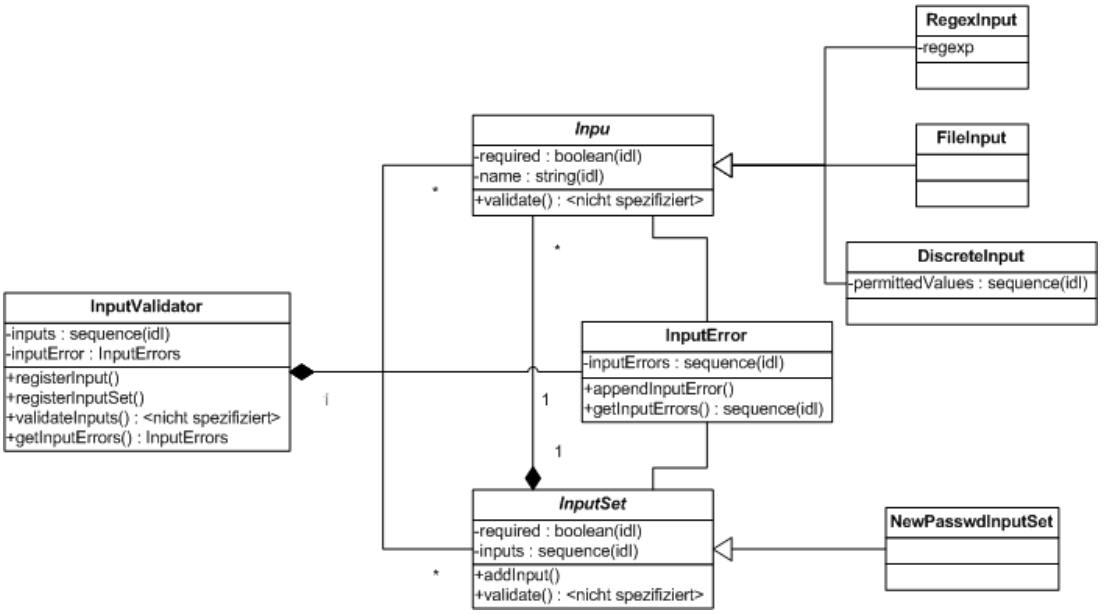

Abbildung 5-1: Input Validator Pattern

Es sei darauf hingewiesen, dass in der entwickelten Anwendung nur eine Input- und eine InputSet-Klasse existieren, welche die vom Entwickler bereitgestellten PHP-Funktionen für die verschiedenen Input-Typen aufrufen. Die Umsetzung des Patterns ist also nicht durchgehend objektorientiert, so ist anstatt einer RegexInput Klasse (vgl. Abbildung 5-1) in der Anwendung eine regex-Funktion vorzufinden.

Mit dem Input Validator Pattern soll eine Vorgehensweise zur konsistenten und zentralen Überprüfung von Benutzereingaben in Web-Anwendungen aufgezeigt werden. Abbildung 5-1 zeigt ein vollständig objektorientiertes Pattern und ist nicht auf den Einsatz in PHP beschränkt. Die Inputs und InputSets erhalten die Benutzereingaben über ein Request-Objekt, das die Informationen eines HTTP-Requests enthält. Es wurde aus Gründen der Übersicht nicht in obiges Klassendiagramm aufgenommen.

5.3 Die Architektur des Institutsinformationssystems

5.3.1 Übersicht über die Bearbeitung einer Anfrage

Die einzelnen Komponenten der Anwendung und deren Zusammenspiel sollen mit folgender Grafik dargestellt werden. Abbildung 5-2 beschreibt die Bearbeitung einer Anfrage durch den Client in sechs Schritten.

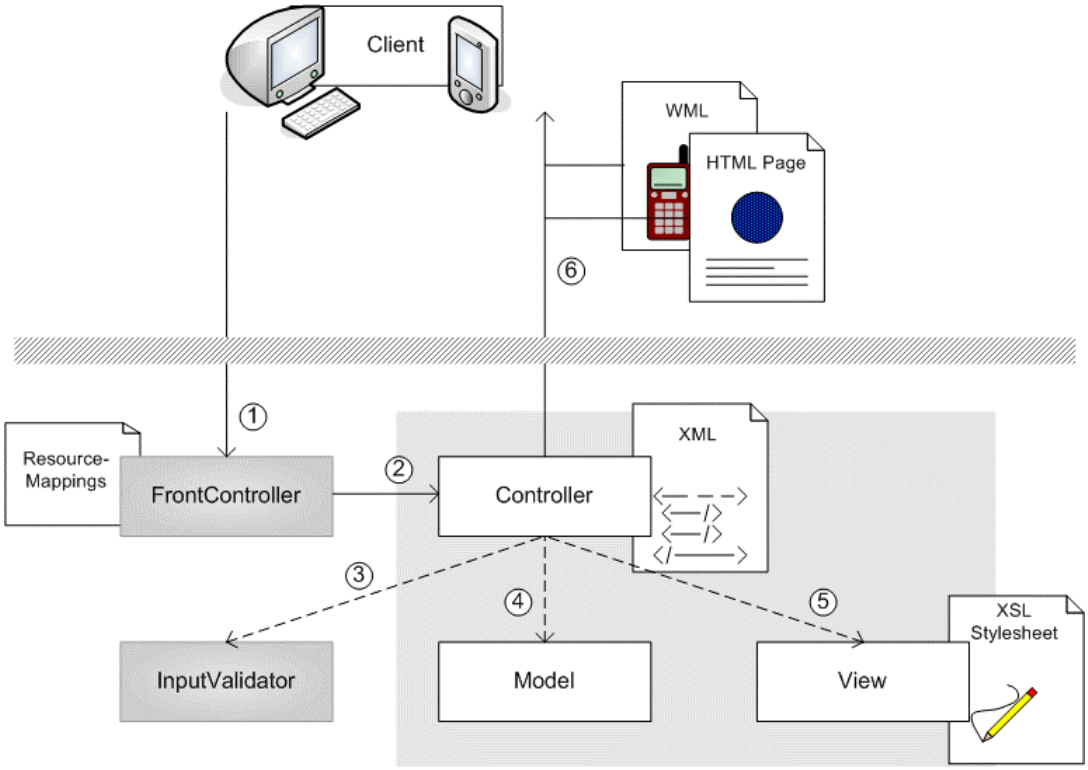

Abbildung 5-2: Architektur der entwickelten Anwendung

1. Anfrage an den Front Controller, eventuelle Authentifizierung und Autorisierung des Benutzers und Auswahl des richtigen Controllers für die Aktion
2. Weiterleitung der Anfrage an den richtigen Controller
3. Eventuelle Überprüfung von Benutzereingaben
4. Eventuelle Model-Aufrufe (Speichern oder Abrufen von Anwendungsdaten)
5. Auswahl des benötigten Views und Transformation der XML-Daten in das gewünschte Output-Format.
6. Senden des Transformationsergebnisses an den Client

Je nach Komplexität einer Anfrage werden drei Typen von Controllern unterschieden: der minimale Controller, der Controller mit Model-Aufrufen und der Controller mit Model-Aufrufen und Benutzereingaben.

5.3.2 Minimaler Controller

Der Client schickt eine Anfrage an den Webserver ①. Dabei laufen alle Anfragen über den Front Controller, dessen Aufgabe es ist, den richtigen Controller[18] für die weiteren Ausführungen auszuwählen. Der Front Controller ist das einzige Objekt, dass sich im öffentlichen Web-Verzeichnis des Webservers befindet. Alle anderen Ressourcen können an anderer Stelle des Dateisystems abgelegt sein, denn sie werden vom Front Controller zum richtigen Zeitpunkt eingebunden. Abbildung 5-2 zeigt, dass der Front Controller mit einer Konfigurationsdatei arbeitet, um den Dateisystempfad und die nötigen Zugriffsberechtigungen für die gewünschte Ressource zu ermitteln. Folgendes Code-Beispiel 5-1 zeigt einen Ausschnitt aus der Konfigurationsdatei, die als 2-dimensionales, assoziatives Array organisiert ist.

```
$resourceMap = array(
    'main'  => array(
        'path'      => CONTROLLER . 'main.php',
        'interpret' => true ,
        'access'    => '')
...
);
```

Code-Beispiel 5-1: Die Konfiguration für die Aktion „main“

Der Eintrag zeigt die Konfiguration für die Startseite der Anwendung. Der logische Name der Seite lautet „main“, unter dem Eintrag „path“ wird der Pfad im Dateisystem und unter dem Eintrag „access“ werden die Zugriffsberechtigungen angegeben (in diesem Fall ist jeder Benutzer berechtigt). Beispiele für die URL dieser Ressource könnten folgendermaßen aussehen:

```
http://www.example.com/index.php?id=main
http://www.exmaple.com/index.php?action=main
```

[18] In der Patternbeschreibung in Kapitel 3 wurde der Begriff ‘Command’ verwendet. Um den Einsatz des MVC Patterns zu verdeutlichen, wird ‚Controller’ als Synonym für ‚Command’ gebraucht.

Alle Anfragen gehen an die Datei „index.php“, die das Front Controller Objekt enthält. Der Front Controller filtert den logischen Namen „main“ der gewünschten Ressource aus der URL und überprüft anhand der Konfigurationsdatei die Zugriffsberechtigungen (vgl. Code-Beispiel 5-1). Ist eine Berechtigung erforderlich und der Benutzer ist nicht eingeloggt, wird auf eine Login-Seite umgeleitet. Gehört der Benutzer einer nicht-berechtigten Benutzergruppe an, wird der Zugriff verweigert und eine entsprechende Fehlermeldung ausgegeben.

Ist der Benutzer, falls notwendig authentifiziert und autorisiert, dann gibt der Front Controller die Verantwortung an den entsprechenden Controller ab ②. Im Falle der Startseite werden nur statische Informationen angezeigt, die in einem XML-Template gespeichert sind. Der Controller „main.php“ arbeitet nun mit den Klassen des XAO Frameworks, die bereits im Unterkapitel 4.2 vorgestellt wurden. Tatsächlich wird die Klasse BigDoc, eine Kind-Klasse von AppDoc, verwendet. Diese erweitert das XAO Framework zum Bearbeiten der XML-Daten und stellt außerdem einige Funktionen zur Verfügung, die von jeder Seite bzw. recht häufig verwendet werden. So liegen beispielsweise die Navigationsmenüs (allgemein und benutzergruppenspezifische) in eigenen XML-Templates vor. Das BigDoc Objekt bindet die allgemeine Navigation automatisch und, abhängig vom Login-Status, die für die Benutzergruppe spezifische Navigation ein.

Code-Beispiel 5-2 zeigt den minimalen Controller „main.php“, der nur statische Daten aus einem XML-Template einliest, einen View abhängig von den Benutzerrechten und dem verwendeten Endgerät auswählt und die transformierten Daten an den Client zurückschickt ⑤.

```
$strDocRoot = 'doc';
$objAppDoc = new BigDoc($strDocRoot);
$objAppDoc->ndConsumeFile(CONTENT.'main/main.xml');
$objAppDoc->ndSetStylePi(VIEW.'page2html.xsl');
$objAppDoc->Transform();
$objAppDoc->Send();
```

Code-Beispiel 5-2: Minimaler Controller „main.php“

Das Ergebnis der Transformation kann in Abbildung 5-3 betrachtet werden.

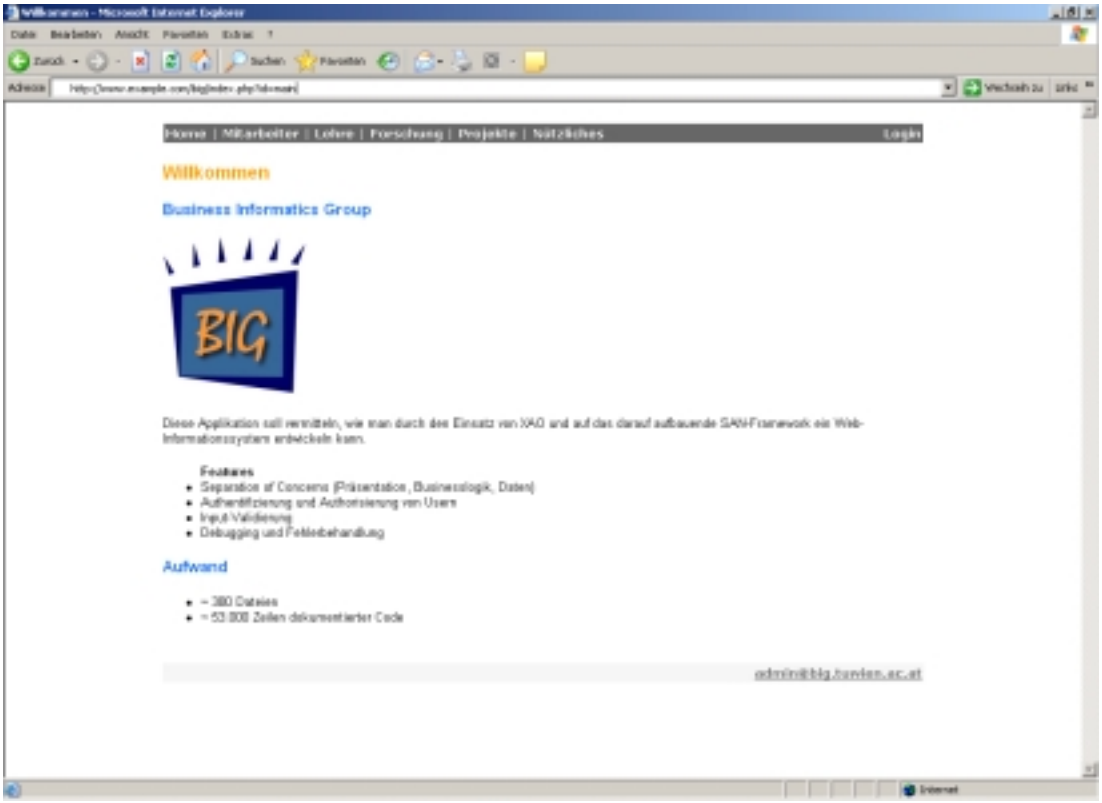

Abbildung 5-3: Startseite für einen nicht eingeloggten Benutzer

5.3.3 Controller mit Model-Aufrufen

Ein etwas fortgeschrittener Controller fügt dynamische XML-Fragmente in das Template ein, in dem er die Daten des Models abfragt ④. Das nächste Beispiel zeigt Ausschnitte aus einem Controller, der eine Mitarbeiterliste ausgeben soll. Die Informationen werden über die Model-Klasse Staff aus der Datenbank geholt und mit dem DbToXml-Objekt von XAO in XML-Daten umgewandelt (vgl. Code-Beispiel 5-3).

```
require_once MODEL.'Staff.php';
//statisches Dokument aufbauen
$strDocRoot     = 'doc';
$objAppDoc      = new BigDoc($strDocRoot);
$objAppDoc->ndConsumeFile(CONTENT.'main/people.xml');

//Mitarbeiterliste aus der Datenbank holen
$objStaff = new Staff();
$arrStaff = $objStaff->mxdGetPeopleList();

//Fehlerbehandlung bei DB-Abfragefehler oder wenn kein Eintrag
//vorhanden ist, wurden hier der Übersichtlichkeit halber entfernt

//Umwandlung der DB-Ergebnisse nach XML
$db2xml = new DbToXml($arrStaff);
//Namen der XML-Elemente auswählen
$db2xml->SetResultTagName('staff');
$db2xml->SetRowTagName('staff_entry');
$db2xml->Execute();
//Zeiger auf das “staff”-Element erhalten
$dbNode = $db2xml->arrNdXPath('/staff');

//Einfügen der XML-Daten in das XML-Template
$ndStaff = $objAppDoc->arrNdXPath('//staff');
$objAppDoc->ndReplaceNode($ndStaff[0], $dbNode[0]);

//Transformation der XML-Daten nach HTML
$objAppDoc->ndSetStylePi(VIEW.'people2html.xsl');
$objAppDoc->Transform();
$objAppDoc->Send();
```

Code-Beispiel 5-3: Controller mit Model-Aufrufen

5.3.4 Controller mit Benutzereingaben

In vielen Fällen werden vom Benutzer Daten an den Controller geschickt ③. Diese müssen überprüft werden, um den weiteren Verlauf der Bearbeitung bestimmen zu können. Für die entwickelte Anwendung wurde das zuvor vorgestellte Input Validator Pattern implementiert. Der Hintergrund des nächsten Beispiels ist eine Seite, die einem Benutzer ermöglicht seine Account-Informationen zu ändern. Folgende Codefragmente 5-4 zeigen, wie ein InputValidator konfiguriert werden kann.

```
$objValidator = new InputValidator();
$objValidator->setInput(
    new Input('firstname',TRUE,'Vorname',studenten_VORNAME,'text'));
$objValidator->setInput(
    new Input('email', TRUE, 'E-Mail',
    studenten_EMAIL, 'regex', '/^[^@]*@[^@]*$/'));
$objPasswdNewSet = new InputSet('passwdNewSet', FALSE);
    $objPasswdNewSet->addInput(
        new Input('passwd_old', TRUE, 'Altes Passwort',
        auth_PASSWORD, 'regex', '/^[A-Za-z\d.:_$&%-]{6,10}$/'));
    $objPasswdNewSet->addInput(
        new Input('passwd_new1', TRUE, 'Neues Passwort',
        auth_PASSWORD, 'regex', '/^[A-Za-z\d.:_$&%-]{6,10}$/'));
    $objPasswdNewSet->addInput(
        new Input('passwd_new2', TRUE, 'Neues Passwort
       (bestätigen)',
        auth_PASSWORD, 'regex', '/^[A-Za-z\d.:_$&%-]{6,10}$/'));
$objValidator->setInputSet($objPasswdNewSet);
...
$arrSecureValues = $objValidator->arrValidateInputs();
$objInputError   = $objValidator->getInputError();

$firstname = $arrSecureValues['firstname'];
$email     = $arrSecureValues['email'];
$passwd    = $arrSecureValues['passwdNewSet']['passwd_new1'];
```

Code-Beispiel 5-4: Controller mit Benutzereingaben

Als Inputs werden beispielhaft der Vorname und die E-Mail des Benutzers beim InputValidator registriert. Die Überprüfung des Vornamens übernimmt die Funktion „text“, während für die E-Mail die Funktion „regex“ aufgerufen wird. Diese Funkti-

onen müssen von den EntwicklerInnen bereitgestellt werden und benötigen eine bestimmte Signatur. Als InputSet werden drei Inputs für Passwörter registriert. Um sein Passwort ändern zu können, muss der Benutzer sein altes Passwort, sein neues Passwort und die Bestätigung des neuen Passworts eingeben. Die einzelnen Passwörter werden mit der Funktion „regex“ überprüft. Die Funktion „passwdNewSet“ überprüft die Richtigkeit des alten Passworts und die Übereinstimmung der neuen Passwörter. Ist während der Überprüfungen ein Fehler aufgetreten, dann können die Fehlermeldungen über das InputError-Objekt abgefragt und in den XML-Baum eingebaut werden. Sonst stehen alle Inputs in Form eines 2-dimensionalen, assoziativen Arrays ($arrSecureValues) für weitere Bearbeitungen zur Verfügung.

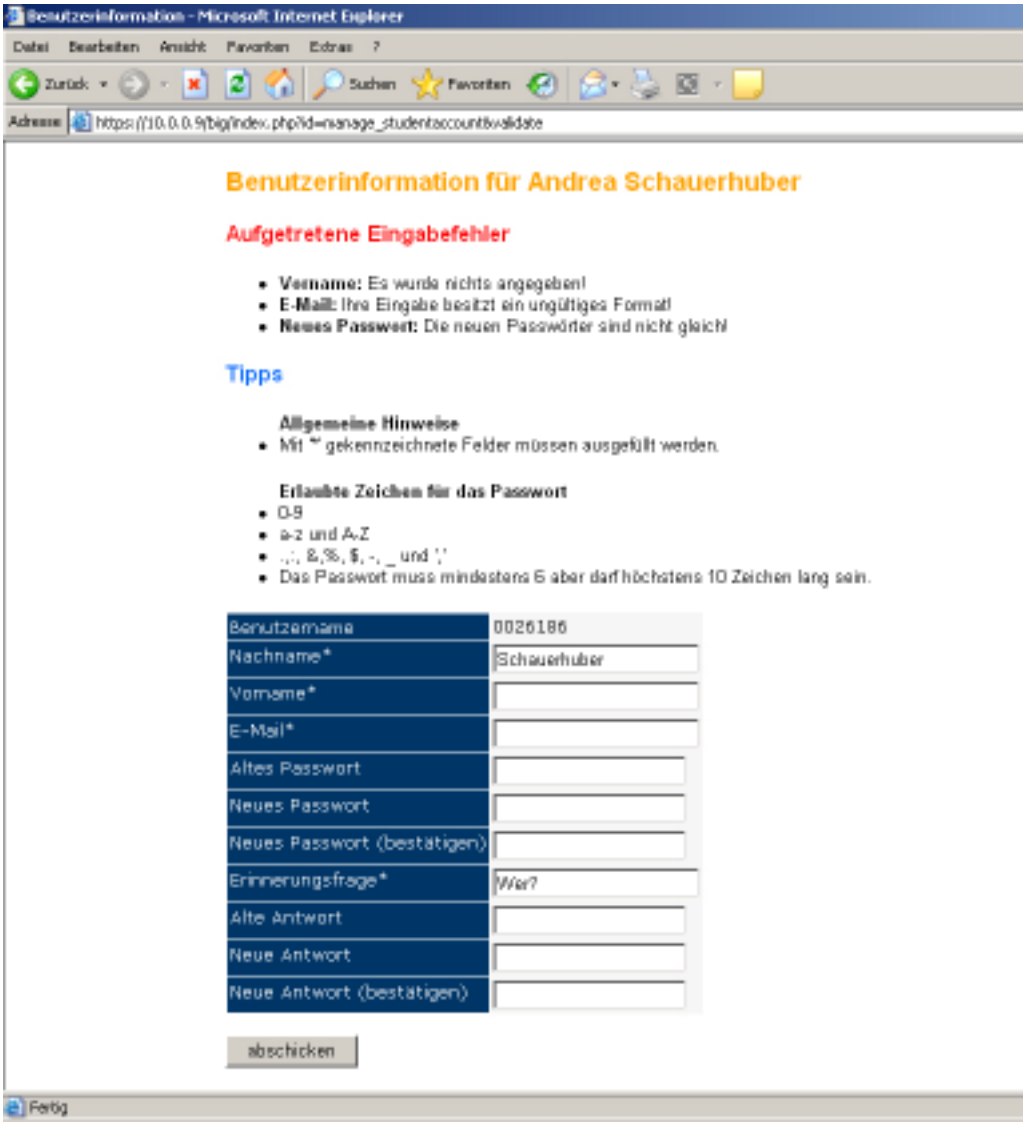

Abbildung 5-4: Fehlerhafte Eingaben beim Ändern eines Studenten-Accounts

In Abbildung 5-4 wird gezeigt, wie das implementierte Institutsinformationssystem auf fehlerhafte Eingaben reagiert. Beim Ändern der Benutzerinformationen hat die Account-Inhaberin Andrea Schauerhuber den Vornamen geändert und vergessen einen neuen Wert in das entsprechende Feld einzugeben. Die E-Mail-Adresse enthielt keinen Klammeraffen. Und beim Ändern des Passworts war zwar das alte Passwort korrekt, bei der Eingabe des neuen Passworts ist hingegen ein Tippfehler passiert.

Allgemein reagiert die implementierte Anwendung auf fehlerhafte Eingaben folgendermaßen:

1. Es werden alle Fehler auf einmal ausgegeben.
2. Jede Fehlermeldung bezieht sich auf ein bestimmtes Eingabefeld.
3. Die gültigen Werte werden wieder in das Formular eingetragen. Geheime Eingaben, wie Passwörter werden nicht nochmals eingetragen.

5.4 SAN Framework im Detail

Die für die Web-Anwendung eigens entwickelten Komponenten, wie der Front Controller oder der Input Validator, werden unter dem Begriff SAN[19]-Framework zusammengefasst. Die nächste Grafik zeigt eine Abbildung der Architektur auf die verwendete Verzeichnisstruktur. Diese Verzeichnisse liegen nicht im Web-Verzeichnis. In letzterem liegt nur das PHP-Skript „index.php“, welches den Front Controller aufruft.

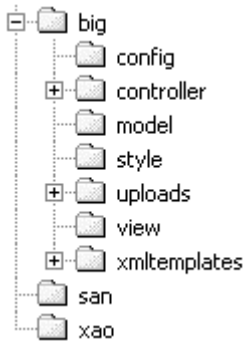

Abbildung 5-5: Projekt-Verzeichnisstruktur

Im Ordner „san“ befinden sich die Klassen des Frameworks: Die Klassen InputValidator, Input, InputSet und InputError stellen die Implementierung des Input Validator Patterns dar und wurden zuvor schon ausführlich beschrieben. Die Beziehungen der Klassen wurden ebenfalls bereits im Abschnitt 5.2.3 erläutert und werden in Abbildung 5-6 aus Gründen der Übersicht nicht eingezeichnet. Die PhpServlet-Klasse implementiert den Front Controller und benutzt ein Objekt der Klasse FileServer, um den Client mit so genannten nicht-interpretierbaren Ressourcen wie Bildern zu bedienen. Alle Klassen des Frameworks sind Kind-Klassen von SanRoot, welche zentral Funktionalitäten, wie eine allgemeine Fehlerbehandlung, implementiert[20]. In weiterer Folge soll nur auf die Implementierung des Front Controllers genauer eingegangen werden.

---

[19] Das Akronym ergibt sich aus den Vornamen der am Projekt beteiligten EntwicklerInnen: Sascha (für Alexander), Andrea und Nenad. SAN ist außerdem die japanische Bezeichnung für die Zahl 3.

[20] Die InputError Klasse ist eine Kind-Klasse von AppDoc. Damit werden die Fehlermeldungen werden in XML gespeichert und können so besser in den XML-Baum eingebaut werden.

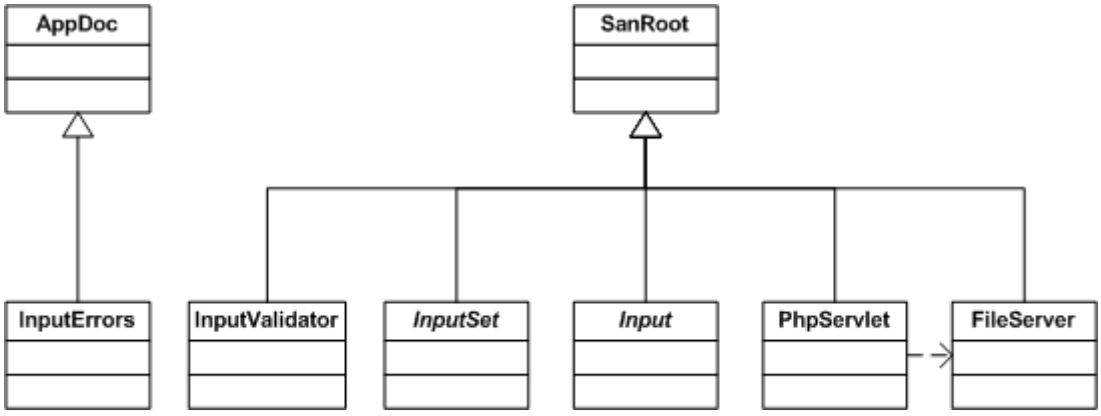

Abbildung 5-6: SAN Klassendiagramm

5.4.1 PhpServlet

Innerhalb des SAN Frameworks wurde das Front Controller Pattern mit der PhpServlet-Klasse implementiert. Die Hauptaufgaben des Front Controllers sind die Auswahl der gewünschten Ressource und eventuelle Authentifizierung und Autorisierung des Benutzers. Für die Verwendung der PhpServlet-Klasse ergeben sich aus den Sicherheitsansprüchen und der Forderung nach Konfigurier- und Wartbarkeit einige wenige Voraussetzungen.

Vorraussetzungen

Im Sinne der Sicherheit wurde während der Entwicklung des Institutsinformationssystems auf die Klassenbibliotheken von PEAR[21] (pear.php.net) zurückgegriffen. Das Authentifizierungspaket von PEAR verlangt zum Speichern der Zugangsdaten (Benutzername und Passwort) eines Benutzers ein relationales Datenbankmanagementsystem. Zur Abstraktion von einer spezifischen Datenbank verwendet das Authentifizierungspaket das Datenbankabstraktionspaket von PEAR. Im wesentlichen wird eine Tabelle mit einer Spalte für den Benutzernamen und einer Spalte für das, über eine Hash-Funktion verschlüsselte, Passwort verlangt. Da das Authentifizierungspaket von PEAR keine Möglichkeit zur Autorisierung eines Benutzers bietet, verlangt die PhpServlet-Klasse zusätzlich eine Tabelle mit einer Spalte für den Benutzernamen und einer Spalte für die Benutzergruppe. Die Zugriffsberechtigungen

für einzelne Ressourcen werden in einer Konfigurationsdatei gespeichert. Das nächste Code-Beispiel 5-5 zeigt die verwendeten Befehle zum Erzeugen der gewünschten Datenbanktabellen in der entwickelten Web-Anwendung. Die Tabellen- und Spaltennamen werden entweder durch die jeweiligen PEAR-Pakete oder durch die PhpServlet-Klasse vorgegeben.

```
CREATE TABLE auth(
    username VARCHAR(50) PRIMARY KEY,
    password VARCHAR(32) NOT NULL
) ENGINE = InnoDB;

CREATE TABLE access(
    auth_username VARCHAR(50) PRIMARY KEY,
    usergroup VARCHAR(50) NOT NULL,
    INDEX (auth_username),
    FOREIGN KEY (auth_username)
    REFERENCES auth(username)
    ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE = InnoDB;
```

Code-Beispiel 5-5: SQL-Befehle zur Erstellung der „auth“ und „access“ Tabellen

Als weitere Sicherheitsmaßnahme werden Cookies zum Speichern von Sitzungsdaten nur über sichere Leitungen, also über HTTPS übertragen [Jova04]. Die entsprechende Einstellung in der PHP-Konfigurationsdatei php.ini ist session.cookie_secure. Das PhpServlet-Objekt erzeugt für jede Anfrage eine Session bzw. führt sie weiter, egal ob die gewünschte Ressource Zugriffsrechte verlangt oder nicht. Sobald ein Benutzer sich eingeloggt hat, wird die Session für jede Anfrage weitergeführt, auch wenn öffentlich zugängliche Seiten besucht werden. Durch diese Vorgangsweise und die erwähnte Einstellung in der PHP-Konfiguration entstehen am Server teilweise nicht benötigte Sessions, die vom PhpServlet-Objekt erkannt und sofort wieder gelöscht werden. Der entsprechende Codeteil kann auskommentiert werden, wenn auf diese Cookie-Einstellung verzichtet wird.

---

[21] PEAR ist ein anerkanntes Projekt in der PHP-Entwicklergemeinde. Ziel des Projekts sind stabile objektorientierte Bibliotheken, die von deren Entwicklern auch gewartet und weiterentwickelt werden.

Den Konfigurations- und Wartbarkeitsansprüchen betreffend, arbeitet die PhpServlet-Klasse mit einer Konfigurationsdatei. Im Abschnitt 5.3.2 wurde bereits eine Beispielkonfiguration vorgestellt. An dieser Stelle sollen mit Hilfe von Code-Beispiel 5-6 die verschiedenen Einstellungsmöglichkeiten genauer erläutert werden.

```
$resourceMap = array(
    'main'  => array(
        'path'      => CONTROLLER . 'main.php',
        'interpret' => TRUE ,
        'access'    => ''),
    'add_student'  => array(
        'path'      => CONTROLLER . 'manage_users/add_student.php',
        'interpret' => TRUE ,
        'access'    => array('admin','staff')),
...
    'style'  => array(
        'path'      => STYLE . 'big.css',
        'interpret' => FALSE ,
        'access'    => ''),
    'biglogo'  => array(
        'path'      => STYLE . 'biglogo.gif',
        'interpret' => FALSE ,
        'access'    => '')
);
```

Code-Beispiel 5-6: Eine Beispielkonfiguration

Die gewünschten Ressourcen werden über einen logischen Namen angesprochen, z. B. „main“ für die Startseite oder „biglogo“ für eine Bilddatei. Für jede Ressource müssen Pfad, Zugriffsberechtigungen und „Interpretierbarkeit“ angegeben werden. Mit letzterer Eigenschaft ist gemeint, ob eine Datei vom PHP-Interpreter geparst werden muss oder nicht. Ist der Wert dieser Einstellung „FALSE“, so muss das PhpServlet-Objekt nur die Inhalte der Datei an den Client zurückschicken. Dies ist beispielsweise bei Bilddateien notwendig. Das PhpServlet-Objekt greift bei dieser Aufgabe auf ein Objekt der Klasse FileServer zurück, welches

- anhand des Dateipfades die gewünschte Ressource ermittelt,
- den MIME-Type der Dateiinhalte bestimmt,
- den HTTP-Content-Type-Header dementsprechend setzt,

- die Dateiinhalte ausliest
- und an den Client schickt.

Der Dateisystempfad zu einer Ressource liegt als Zeichenkette unter dem Schlüssel „path“ vor, während die Zugriffsberechtigungen unter dem Schlüssel „access“ zu finden sind. Ist eine Ressource öffentlich zugänglich, so wird ein eine leere Zeichenkette als Wert eingetragen. Falls hingegen auf eine Ressource nur beschränkter Zugriff besteht, werden die autorisierten Benutzergruppen mit Hilfe eines Arrays angeführt (siehe „add_student“-Ressource in Code-Beispiel 5-6).

Da der Front Controller mit logischen Namen arbeitet, muss ihm dieser für jede Anfrage mitgeteilt werden. Üblicherweise geschieht dies über einen speziellen Parameter in der URL.

```
http://www.example.com/index.php?id=main
```

Falls nicht anders konfiguriert, erwartet das PhpServlet-Objekt den logischen Namen der angeforderten Ressource als Wert des Parameters „id“. Mit der Methode setIdName() kann der Parametername aber verändert werden, z. B. auf „action“ (vgl. Klassendokumentation auf der beiliegenden CD).

Als letzte Voraussetzung benötigt das PhpServlet-Objekt eine Konfigurationsdatei mit mindestens folgenden drei logischen Namen:

main Diese Ressource stellt die Startseite der Web-Anwendung dar. Auf diese Seite wird umgeleitet, falls der Benutzer denn Parameter „id“ in der URL weglässt bzw., wenn sich der Benutzer über eine Login-Seite authentifiziert hat.

login Hinter diesem logischen Namen versteckt sich die Login-Seite.

logout Auf diese Seite wird umgeleitet, wenn sich der Benutzer ausgeloggt hat.

Falls die Web-Anwendung keine autorisierten Benutzersichten besitzt, kann auf die letzten beiden Ressourcen verzichtet werden. Obige logische Namen stellen die Grundeinstellung des PhpServlet-Objekts dar und können mit der Funktion setRessource() geändert werden (vgl. Klassendokumentation auf der beiliegenden CD).

Konfiguration des Front Controllers

Die nächsten Codezeilen sollen andeuten, wie das Front Controller-Skript „index.php“ aufgebaut und das PhpServlet-Objekt konfiguriert werden kann.

```
<?php
//KONFIGURATIONEN von Konstanten
define('BIG_PROJECT','big/');
define('MODEL', BIG_PROJECT . 'model/');
define('VIEW', BIG_PROJECT . 'view/');
define('CONTROLLER', BIG_PROJECT . 'controller/');
define('CONTENT', BIG_PROJECT . 'xmltemplates/');
define('CONFIG', BIG_PROJECT . 'config/');
define('STYLE', BIG_PROJECT .'style/');
define('UPLOADS', BIG_PROJECT . 'uploads/');

//EINBINDEN der Klasse und der Konfigurationsdatei
require_once 'san/PhpServlet.php';
require_once CONFIG . 'resourcemap.php';
global $resourceMap; //Konfigurationsarray aus resourcemap.php

//Datasourcename für PEAR::Auth
$dsn_auth = 'mysql://auth_reader:auth_reader@localhost/bigdb;
//Datasourcename für Authorisierung in PhpServlet
$dsn_access = 'mysql://auth_reader:auth_reader@localhost/bigdb;

//DEFINITION der Callbackmethoden
//Callback-Methoden der Auth-Klasse
function loginFunction($username, $loginStatus){...}
function logoutFunction(){...}
//Callback-Methode der PhpServlet-Klasse
function errorFunction(){...}

//ERZEUGEN und KONFIGURIEREN des FrontControllers
$servlet = new PhpServlet($resourceMap, $dsn_auth, $dsn_access,
           'loginFunction', 'logoutFunction', 'errorFunction');

//STARTEN des FrontControllers
$servlet->start();
?>
```

Code-Beispiel 5-7: Konfiguration des Front Controllers

Zu Beginn müssen die benötigten Konstanten definiert werden. An dieser Stelle finden u. a. Konfigurationen für den E-Mail-Versand statt, welche der Einfachheit halber ausgelassen wurden. Wie in Code-Beispiel 5-7 zu sehen, werden die Konstanten für die benötigten Dateisystemverzeichnisse des Projekts definiert (vgl. Abbildung 5-5). Das PhpServlet-Objekt wird im wesentlichen über den Konstruktor konfiguriert. Dieser verlangt maximal sechs Parameter, wobei die letzten drei optional sind:

`$resourceMap`	Stellt das Konfigurationsarray dar.
`$dns_auth`	Enthält die Zugangsdaten für die Datenbanktabelle, die Benutzernamen und Passwörter speichert. `Datenbantyp://DB-benutzer:Passwort@Server/Datenbank`
`$dns_access`	Enthält die Zugangsdaten für die Datenbanktabelle, die Benutzernamen und Berechtigungen speichert.
`$loginFunc`	Das Authentifizierungspaket von PEAR stellt eine Login-Seite zur Verfügung. Diese ist nur im HTML-Format vorhanden. Der Entwickler kann eine eigene Login-Funktion angeben, die dann vom PhpServlet-Objekt aufgerufen wird. Durch diese Variable wird der Name dieser Funktion bekannt gegeben.
`$logoutFunc`	Wenn dieser Parameter nicht verwendet wird, dann loggt das PhpServlet-Objekt den Benutzer aus und leitet ihn auf die „logout“-Ressource um. In einer eigenen Logout-Funktion könnten die EntwicklerInnen bestimmte Abschlussarbeiten ausführen.
`$errorFunc`	Während der Verarbeitung einer Anfrage durch das PhpServlet-Objekt können vier verschiedene Fehler auftreten. Ist keine eigene Funktion vorhanden, erfolgt eine einfache HTML-Ausgabe der Fehlermeldung. Fehlercode und URL stehen im Falle eines Fehlers in der Session unter den Schlüsseln ‚PhpServletError_Code’ und ‚PhpServletError_Request’ für spezielle Ausgaben in einer eigenen Funktion zur Verfügung (vgl. Klassendokumentation auf der beiliegenden CD).

Tabelle 5-1: Parameter des PhpServlet Konstruktors

Bevor das PhpServlet-Objekt allerdings konfiguriert werden kann, müssen die Klasse und die Konfigurationsdatei eingebunden und die Login-, Logout- und Fehler-Funktionen definiert werden. Mit der „start“-Methode wird die Verarbeitung der Anfrage schließlich angestoßen.

5.4.2 Bearbeitung einer Anfrage durch den Front Controller

Stellt ein Client eine Anfrage an einen Webserver, so laufen im Hintergrund gewöhnlich mehrere HTTP-Requests ab. Das liegt daran, dass eine HTML-Seite nicht nur Text sondern auch andere multimediale Inhalte, wie Bilder und Musik, enthält. Zusätzlich können JavaScripts und Cascading Style Sheets mitgeladen werden, um die Anzeige im Browser zu modifizieren bzw. dynamisch zu machen. Für jede Ressource, die in die HTML-Seite eingebettet wird, muss der Browser aber eine neue Verbindung zum Webserver aufbauen. Diese zusätzlichen Anfragen müssen ebenfalls vom Front Controller verarbeitet werden. Dazu reicht es jedoch nicht die Datei lediglich einzubinden, so wie dies bei den in PHP geschriebenen Controllern geschieht. Damit der Browser eine Bilddatei auch als solche erkennt, muss in der HTTP-Response der so genannte HTTP-Content-Type-Header auf den richtigen MIME-Type gesetzt werden. Im Falle der Bilddatei „biglogo.gif“ wäre dieser „image/gif“. Sobald dies geschehen ist, können die Inhalte der Datei an den Client geschickt werden. Die PhpServlet-Klasse verwendet ein FileServer-Objekt, das genau diese Aufgabe übernimmt. Die FileServer-Klasse benötigt dazu nur den Dateisystempfad einer Ressource und kann auch unabhängig von PhpServlet eingesetzt werden. Die nächste Abbildung soll den gerade beschriebenen Vorgang illustrieren. Wie schon zuvor, dient die Startseite der entwickelten Web-Anwendung als Beispiel für die weiteren Erklärungen.

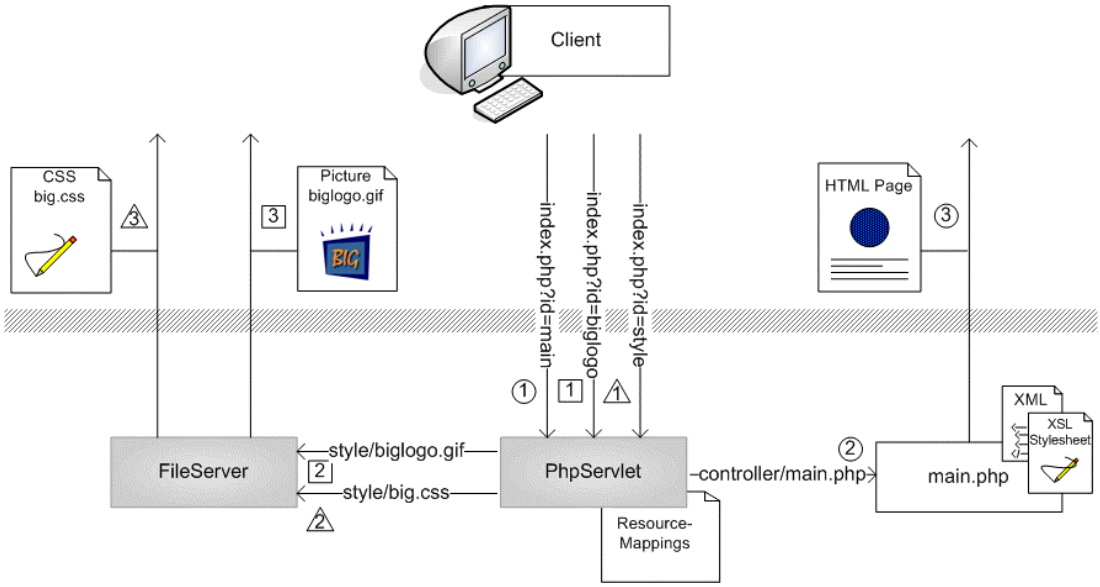

Abbildung 5-7: Bearbeitung einer Anfrage durch das PhpServlet

Die vom Benutzer mit Hilfe der URL http://www.example.com/index.php?id=main initiierte Anfrage an den Webserver (siehe Abbildung 5-7 ① -③) veranlasst den Front Controller den Controller main.php aufzurufen. Dieser hat, wie schon in früheren Beispielen beschrieben, die Aufgabe statische XML-Daten aus einem XML-Template einzulesen und diese mit einem entsprechenden XSL-Stylesheet in das HTML-Format zu konvertieren (vgl. Abschnitt 5.3.2). Das Ergebnis dieser Transformation ist eine HTML-Seite mit zwei Referenzen auf andere Ressourcen, nämlich einem Bild und einem CSS-Stylesheet.

Der folgende Codeausschnitt 5-8 aus dem HTML-Quelltext zeigt die beiden URLs, unter denen die Ressourcen zu finden sind.

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
    <link rel="stylesheet" type="text/css"
          href="http://www.example.com/index.php?id=style">
    <title>Willkommen</title>
  </head>
  <body>
    ...
    <h1>Willkommen</h1>
    <h2>Business Informatics Group</h2>
    <img src="http://www.example.com/index.php?id=biglogo">

    ...
  </body>
</html>
```

Code-Beispiel 5-8

Der Browser stellt nun automatisch zwei zusätzliche Anfragen an den Webserver, siehe Abbildung 5-7 ① - ③ für das CSS-Stylesheet und ⚠ - ⚠ für die Bilddatei. Für den Front Controller zeigt sich durch die Konfigurationsdatei aufgrund des Schlüssels „interpret“ (vgl. Code-Beispiel 5-6), dass sowohl die Ressource mit dem logischen Namen „style“ als auch die Ressource mit dem logischen Namen „biglogo“ mittels FileServer-Objekt an den Client zurückgeschickt werden muss. Der Browser fügt die gesammelten Daten schlussendlich zu der in Abbildung 5-3 dargestellten Anzeige zusammen.

Kapitel 6

6 Funktionalität des Institutsinformationssystems

Das Institutsinformationssystem wurde für die Bedürfnisse der Arbeitsgruppe BIG (Institut für Softwaretechnik und interaktive Systeme, Technische Universität Wien) maßgeschneidert und unterstützt primär deren Mitarbeiter in den Bereichen Lehre, Forschung und Verwaltung. Dieses Kapitel bietet eine knappe Übersicht über die Funktionalitäten, welche durch diese Anwendung zur Verfügung gestellt werden (siehe beiliegende CD für die gesamte Software). Das System unterscheidet neben einer öffentlichen Sicht auch weitere drei autorisierte Benutzersichten, mit denen jeweils verschiedene Funktionalitäten einhergehen. Diese Benutzersichten sind

- öffentliche Sicht
- Studentensicht
- Mitarbeitersicht
- Sicht des Systemadministrators

Benutzer besitzen gegenüber einer Web-Anwendung die Erwartungshaltung, einzelne Funktionen „intuitiv“ in Anspruch nehmen zu können, ohne dass eine vorherige Einarbeitung mittels eines Bedienungshandbuchs erforderlich ist. Um dieser Erwaltungshaltung zu entsprechen, wurde im Zuge der Entwicklung einerseits auf altbewährte Layoutkonzepte zurückgegriffen und andererseits dem Benutzer konsequent durch erläuternde Anmerkungen Hilfe geleistet. Daher wurde von der Erstellung ei-

nes detaillierten Benutzerhandbuchs abgesehen. Stattdessen wird in Abbildung 6-1 ein Überblick über die angeboten Anwendungsfälle gegeben.

Da die Implementierung des Institutsinformationssystems im Team stattfand, ist dieses Kapitel in identischer Form in jeder der drei Diplomarbeiten enthalten (siehe [Heum04, Jova04]).

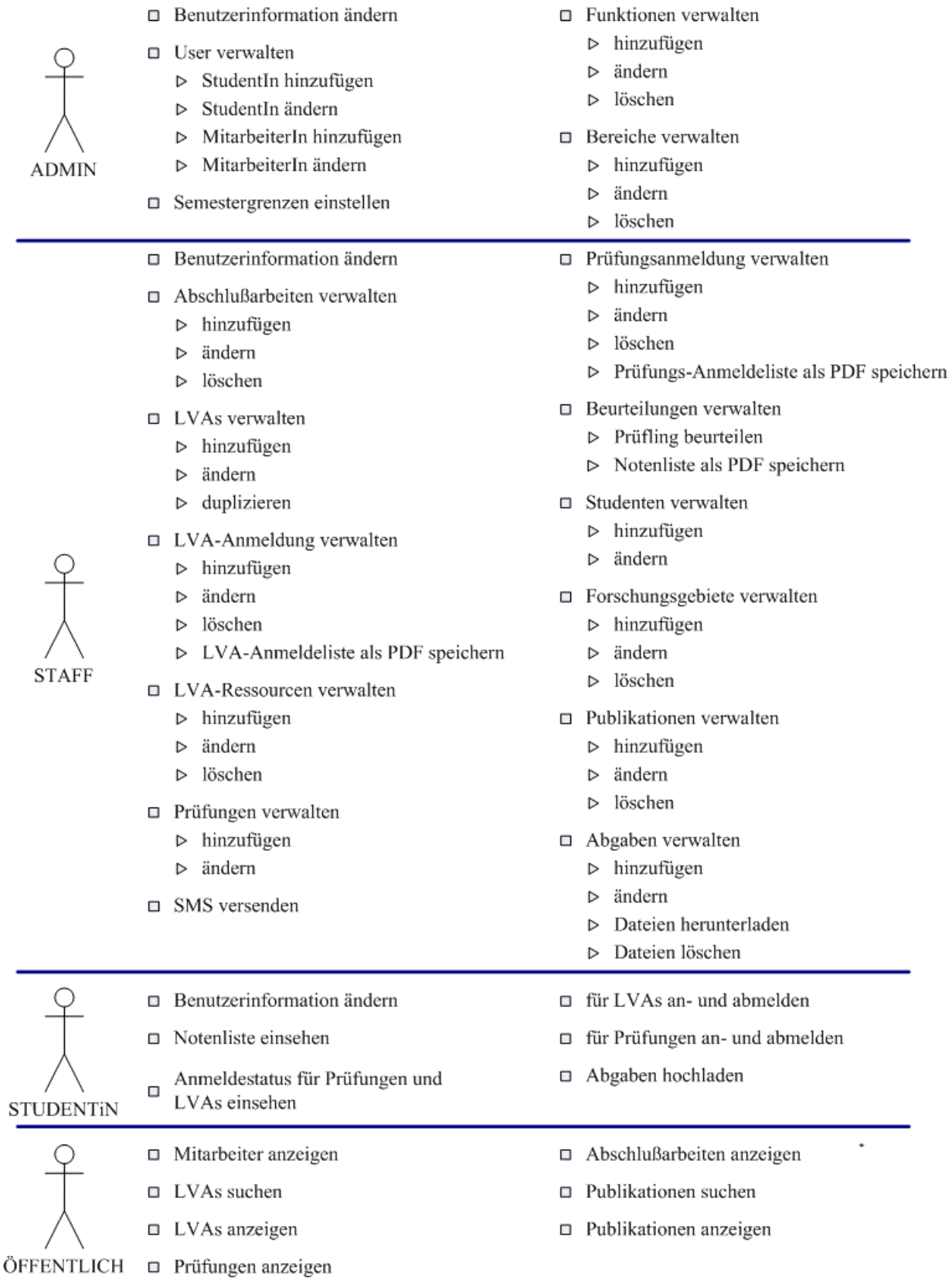

Abbildung 6-1: Funktionen des Institutsinformationssystems

Wie in Abbildung 6-1 erkennbar, bietet das entwickelte System ein reichhaltiges Spektrum an Funktionen, die die gängigen Aufgaben eines kleineren bis mittleren Instituts abzudecken vermögen. Neben einer Reihe "typischer" Anwendungsfälle zählen dazu auch weniger weit verbreitete Features. Als Beispiel hierfür ist u. a. die Wiki-Engine zu erwähnen, die der beliebten PHP Wiki (http://phpwiki.sourceforge.net/) nachempfunden wurde. Abgesehen von der strukturierten, formularbasierten Eingabe von Lehrveranstaltungsinformationen besteht damit für eine Lehrkraft auch die Möglichkeit, ergänzend frei strukturierte Angaben zu einer Lehrveranstaltung hinzuzufügen. Auf diese Weise kann auf die spezifischen Eigenheiten einer Lehrveranstaltung eingegangen werden. Darüber hinaus ist die Konvertierung diverser dynamisch erzeugter Listen (wie z. B. Notenlisten oder Anmeldelisten) in das PDF-Format zu erwähnen, die ein optisch ansprechendes Ausdrucken dieser Informationen gestattet.

Kapitel 7

7 Zusammenfassung und Ausblick

Eine fundierte Auseinandersetzung mit Architekturen und Patterns ist im Software Engineering schon lange ein absolutes Muss. Aber auch im Bereich Web Engineering mit den immer anspruchsvolleren Anforderungen an Web-Anwendungen beginnen EntwicklerInnen sich eingehendst mit einer systematischen Vorgehensweise zur Web-Entwicklung mit Hilfe von Patterns zu beschäftigen.

Das entwickelte Institutsinformationssystem implementiert in seiner Architektur eine Reihe der in Kapitel 3 vorgestellten Patterns, darunter das allseits bekannte Model View Controller Pattern und das Front Controller Pattern. Um eine vollkommene Trennung von Inhalten und Präsentation erreichen zu können, arbeitet das entwickelte Institutsinformationssystem mit XML als Datenformat und XSLT zur Transformation der Daten in ein beliebiges Präsentationsformat. Im Zuge dessen kam das XAO Framework (http://xao-php.sourceforge.net) zum Einsatz, das mit Hilfe der XML-Technologien das in dieser Arbeit eingesetzte Transform View Pattern implementiert. Auf diese Weise können zusätzlich Endgeräte und Ausgabeformate oder neue Benutzersichten auf die Daten der Anwendung durch ein einfaches Hinzufügen der dementsprechenden XSL-Stylesheets unterstützt werden. Damit ist die ursprüngliche Anforderung der Separation of Concerns für das entwickelte Institutsinformationssystem erfüllt.

Aufgrund der hohen Sicherheitsanforderungen an die Architektur, die sich aus dem Diplomarbeitsthema des Kollegen Nenad Jovanovic [Jova04] ergaben, wurde u. a. besonders großer Wert auf die Überprüfung der Benutzereingaben gelegt. Im Rahmen der vorliegenden Arbeit wurden solche Kontrollen auf der Ebene von einzelnen und zusammenhängenden Eingabefeldern (Inputs und InputSets) angesiedelt. Aus dem implementierten Mechanismus wurde zu einem späteren Zeitpunkt das völlig objektorientierte und sprachunabhängige Input Validator Pattern abgeleitet, das in Kapitel 5 dokumentiert ist. Das Input Validator Pattern vermittelt einen Weg zur zentralen und konsistenten Überprüfung von Benutzereingaben. Eine ähnliche Vorgehensweise war zum Zeitpunkt der Implementierung des Institutsinformationssystems in keinem der untersuchten PHP Frameworks vorzufinden.

Wie bereits in den Ausführungen von Kapitel 4 erwähnt, ist die Entscheidung für ein bestimmtes PHP Framework nicht unbedingt eine einfache. Nachdem die Entscheidung für das XAO Framework gefallen und seiner Erweiterung implementiert war, wurde das Mojavi Framework entdeckt (http://www.mojavi.org). Mojavi ist ebenfalls ein Open-Source MVC Framework. Mit der Veröffentlichung von PHP 5 wird Mojavi momentan völlig überarbeitet. Besonders hervorzuheben ist ein Mechanismus zur Überprüfung von Benutzereingaben, der dem in dieser Arbeit entwickelten Input Validator Pattern in weiten Teilen ähnelt. Abgesehen davon verwendet Mojavi auch keine leistungsmindernden HTTP-Redirects (vgl. Phrame in Abschnitt 4.2.2). Inwiefern die XML-Technologien mit Mojavi einsetzbar sind, kann an dieser Stelle nicht beantwortet werden und bleibt offen für weitere Nachforschungen.

Was die Zukunft für LAMP betrifft, geht PHP mit Version 5 einen großen Schritt in Richtung professioneller, objektorientierter Web-Entwicklung [PHP04]. Das völlig überarbeitete Objektmodell unterstützt viele Merkmale, die man in PHP 4 während der Entwicklung vermisst hat. Dazu gehören

- Abstrakte Klassen und Methoden
- Interfaces
- Statische, protected und private Eigenschaften und Methoden
- Exceptions, u. v. m.

Die Neuerungen im Bereich der XML-Erweiterungen wurden bereits in Kapitel 4 ausführlich erwähnt und lassen für die Zukunft hoffen, dass viele Frameworks den Einsatz von XML unterstützen werden. Das in dieser Arbeit eingesetzte XAO Framework ist noch in PHP 4 implementiert. Zur Zeit gibt es keine Angaben, ob das XAO Framework in einer neuen Version basierend auf PHP 5 veröffentlicht wird. Obwohl laut [Stoc04] eine Portierung nach PHP Version 5 für die XML-Erweiterung leicht getan ist, darf man die Verbesserungen am Objektmodell von PHP 5 nicht vergessen. Vieles musste in PHP 4 aufgrund fehlender objektorientierter Eigenschaften grundlegend anders programmiert werden, sodass zu befürchten ist, das eine Portierung der meisten in PHP 4 geschriebenen Web-Anwendungen mit nicht mit wenig Aufwand verbunden ist. Für die Zukunft von LAMP ist außerdem wünschenswert, dass sich einige gute Frameworks weiterentwickeln und einen Bekanntheitsgrad erreichen, ähnlich dem zu Struts und Cocoon für die Java-Welt. Momentan geht der Trend in der PHP-Entwicklergemeinde noch in Richtung Eigenentwicklungen.

Abkürzungsverzeichnis

API.....Application Programming Interface
ARPA .....Advanced Research Projects Agency
ASP.....Active Server Pages
CERN .....Centre European pour la Recherche Nucleaire
CGI .....Common Gateway Interface
CSS.....Cascading Style Sheets
DCOM .....Distributed Component Object Model
DOM.....Document Object Model
DTD.....Document Type Definition
GUI .....Graphical User Interface
HTML.....Hypertext Markup Language
HTTP .....Hypertext Transfer Protocol
HTTPS.....Hypertext Transfer Protocol Secure
IIOP.....Inter-ORB Protocol
J2EE.....Java 2 Platform Enterprise Edition
JSP .....JavaServer Pages
LAMP .....Linux, Apache, MySQL, PHP
MIME .....Multipurpose Internet Mail Extensions
MVC .....Model View Controller
PDF.....Portable Document Format
PEAR.....PHP Extension and Application Repository
PHP.....PHP Hypertext Preprocessor
RMI.....Remote Method Incovation
RPC.....Remote Procedure Call
SAN .....Sascha, Andrea, Nenad
SQL.....Structured Query Language
URL .....Uniform Resource Locator
W3C.....World Wide Web Consortium
WML .....Wireless Markup Language
WWW.....World Wide Web
XAO .....XML Application Objects
XML .....Extensible Markup Language
XSL.....Extensible Stylesheet Language
XSLT .....XSL Transformations

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[ABD+04] Mehdi Achour, Friedhelm Betz, Antony Dovgal, u. v. m., *PHP-Manual*, http://www.php.net/manual/en/, 28. August 2004, [2004-10-07]

[AIS+77] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel, *A Pattern Language – Towns, Buildings, Construction*, Oxford University Press, New York, 1977

[AlCM01a] Deepak Alur, John Crupi, Dan Malks, *Core J2EE Pattern Catalog: Front Controller*, http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html, [2004-09-05]

[AlCM01b] Deepak Alur, John Crupi, Dan Malks, *Core J2EE Pattern Catalog: Intercepting Filter*, http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html, [2004-09-05]

[AnRo01] Michalis Anastopoulos, Tim Romberg, *Referenzarchitekturen für Web-Applikationen*, Projekt Application2Web, Forschungszentrum Informatik (FZI), 2001, http://app2web.fzi.de/, [2004-09-05]

[Benk03] Siegfried Benkner, *Software Engineering: IT/SWE Vorlesung – WS2003/2004*, Institut für Softwarewissenschaften, Universität Wien, http://www.par.univie.ac.at/teach/03W/SWE/vo8.pdf, [2004-09-28]

[BMR+01] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, *Pattern-Oriented Software Architecture - A System of Patterns*, John Wiley & Sons, Chichester, 2001

[Cona00] Jim Conallen, *Building web applications with UML*, Addison-Wesley, Massachusetts [u. a.], 2000

[Doug01] Dale Dougherty, *LAMP: The Open Source Web Platform*, O'Reilly ON-Lamp.com, http://www.onlamp.com/pub/a/onlamp/2001/01/25/lamp.html, [2004-09-15]

[DuGH03] Schahram Dustdar, Harald Gall, Manfred Hauswirth, *Software-Architekturen für Verteilte Systeme: Prinzipien, Bausteine und Standardarchitekturen für moderne Software*, Springer-Verlag, Berlin, Heidelberg, 2003

[Eich04] Christian Eichinger, *Architektur von Web-Anwendungen*, in Gerti Kappel, Birgit Pröll, Siegfried Reich, Werner Retschitzegger, *Web Engineering: Systematische Entwicklung von Web-Anwendungen*, dpunkt.verlag, Heidelberg, 2004

[Fowl03a] Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, Boston, 2003

[Fowl03b] Martin Fowler, *Who Needs an Architect?*, IEEE Software, 20 (4) Juli/August, 2003, pp. 2-4

[Fuec02a] Harry Fuecks, p*hpPatterns* http://www.phppatterns.com/index.php/article/articleview/10/1/5/, 2002, [2004-09-22]

[Fuec02b] Harry Fuecks, *Templates and Template Engines*, http://www.phppatterns.com/index.php/article/articleview/4/1/1/, 2002, [2004-09-16]

[Fuec03a] Harry Fuecks, *The Front Controller and PHP*, http://www.phppatterns.com/index.php/article/articleview/81/1/1/, 2003, [2004-09-05]

[GHJV04] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Entwurfsmuster: Elemente wiederverwenbarer objektorientierter Software*, Addison-Wesley, München, 2004

[Goll02] Heiko Goller, *Das LAMP Buch: Webserver mit Linux, Apache, MySQL und PHP*, SuSE-Press, Poing, 2002

[Heum04] Alexander Heumayer, *Entwicklung eines webbasierten Institutsinformationssystems unter besonderer Berücksichtigung des Multi Delivery*, Diplomarbeit, Business Informatics Group – Institut für Softwaretechnik und Interaktive Systeme, Technische Universität Wien, 2004

[Jova04] Nenad Jovanovic, *Entwicklung eines webbasierten Institutsinformationssystems unter besonderer Berücksichtigung der Websicherheit*, Diplomarbeit, Business Informatics Group – Institut für Softwaretechnik und Interaktive Systeme, Technische Universität Wien, 2004

[KRP+03] Gerti Kappel, Werner Retschitzegger, Birgit Pröll, Rainer Unland, Bahram Vojdani, *Architektur von Web-Informationssystemen*, in Erhard Rahm, Gottfried Vossen, *Web & Datenbanken: Konzepte, Architekturen, Anwendungen*, dpunkt.verlag, Heidelberg, 2003

[Kunz98] Michael Kunze, *Laßt es leuchten - LAMP: ein datenbankgestütztes Web-Publishing-System mit Freeware*, Heise Zeitschriften Verlag, Hannover, c't 12/98, Seite 230

[Mazz04] Stefano Mazzocchi, *Introducing C*ocoon, http://cocoon.apache.org/2.1/introduction.html, 04.September 2004 [2004-10-26]

[Netc04] *Web Server Survey*, http://news.netcraft.com/archives/web_server_survey.html, 1. Oktober 2004, [2004-10-07]

[OZHS04] Monte Ohrt, Andrei Zmievski, Andreas Halter, Thomas Schulz, *Smarty – die kompilierende PHP Template-Engine*, http://smarty.php.net/manual/de/, ispi of Lincoln Inc., 2004, [2004-09-15]

[Rols03] David Rolston, *LAMP, MySQL/PHP Database driven website design and development, and the List-Detail-Post paradig*m, http://www.phpfreaks.com/tutorials/89/0.php, 2003, [2004-10-07]

[PHP04] *Changes in PHP 5/Zend Engine II* http://www.zend.com/php5/articles/engine2-php5-changes.php, [2004-10-23]

[Sado97] Darleen Sadoski, *Two Tier Software Architectures*, Carnegie Mellon University, Software Engineering Institute, http://www.sei.cmu.edu/str/descriptions/twotier.html, 1997, [2004-09-10]

[SaCo00] Darleen Sadoski, Santiago Comella-Dorda, *Three Tier Software Architectures*, Carnegie Mellon University, Software Engineering Institute, http://www.sei.cmu.edu/str/descriptions/threetier.html, 2000, [2004-10-07]

[Schu95] George Schussel, *Client/Server: Past, present, and future*, http://www.dciexpo.com/geos/, 1995, [2004-04-10], (Link nicht mehr vorhanden, ein persönliche Kopie ist über den Autor erhältlich)

[Stan99] *Internet feiert Geburtstag: Grundstein vor 30 Jahren gelegt*, http://derstandard.at/?url=/?id=1521697, [2004-10-10]

[Stoc04] Christian Stocker, *XML in PHP5 – What's New?*, http://www.zend.com/php5/articles/php5-xmlphp.php, [2004-09-21]

[TaSt02] Andrew S. Tanenbaum, Maarten van Steen, *Distributed Systems: Principles and Paradigms*, Prentice Hall, Upper Saddle River, New Jersey, 2002

[WeLa04] Matthias Weigel, Oliver Lau, *Schablonenzauber: Arbeitsteilige PHP-Programmierung mit Hilfe von Templates*, Heise Zeitschriften Verlag, Hannover, c't 11/04, Seite 216

[Wiki04] *LAMP*, Wikipedia - The free encyclopedia, http://en.wikipedia.org/wiki/LAMP, [2004-09-15]

[Zako04] Robert H. Zakon, *Hobbes' Internet Timeline v7*.0, http://www.zakon.org/robert/internet/timeline/, [2004-10-10]

Anhang

Inhalt der beiliegenden CD

Die in Kapitel 5 angeführten Beispiele zur Systemdokumentation, mussten aus Platzgründen meist in vereinfachter und gekürzter Form dargestellt werden. Sie wurden im wesentlichen dem implementierten Web-Informationssystem entnommen und können auf der beiliegenden CD in vollständiger und dokumentierter Form betrachtet werden. Detaillierte Ausführungen und Beispiele zu den innerhalb des SAN Frameworks implementierten Komponenten, Front Controller und Input Validator, sind in den Klassendokumentationen, ebenfalls auf der CD, zu finden.

README	Beschreibung des CD-Inhalts
xampp-linux-1.4.8.tar-gz	Aktuelle LAMP-Version der Apache Friends
secure	Die in [Jova04] entwickelten Installations- und Update-Skripts zur Absicherung einer LAMP-Version der Apache Friends sind in diesem Verzeichnis untergebracht.
xao	Enthält die offizielle Version des XAO Frameworks inklusive Dokumentation und Demo-Beispielen. Außerdem ist die aktuellste (eine inoffizielle, vom Autor erhaltene) Arbeitsversion beigefügt. Das Demo-Beispiel aus Abschnitt 4.2.3 ist im Unterverzeichnis „demo“ zu finden.
san	Beinhaltet die Klassen der Komponenten Front

	Controller und Input Validator.
web-infosys	Das implementierte Institutsinformationssystem (in Zusammenarbeit mit Alexander Heumayer und Nenad Jovanovic)
diplomarbeit_schauerhuber_v1.pdf	Vorliegende schriftliche Arbeit im PDF-Format

Tabelle 1: Beschreibung des CD Inhalts

Verwendete Hard- und Software

Im Rahmen der Diplomarbeit wurden folgende Rechner eingesetzt. Die Entwicklung des Institutsinformationssystems fand dabei ausschließlich auf dem TP23 unter der XAMPP-Version 1.4.6 statt. Die anderen Rechner wurden zu Testzwecken des Web-Informationssystems über ein Netzwerk und zur Verfassung des vorliegenden Schriftstücks herangezogen.

Computerbezeichnung	IBM ThinkPad 23 (T23)
Prozessor	Intel Pentium III Mobile 1133 MHz
Hauptspeicher	512 MB
Festplatte	HITACHI DK23FB-60B 80 GB
Netzwerkkarte	Intel PRO/100 VE Network Connection IBM High Rate Wireless LAN MiniPCI Combo Card
Betriebssystem	Windows 2000 Professional Knoppix 3.1 (Kernel Version 2.4)

Tabelle 2: Hardwarebeschreibung T23

Computerbezeichnung	Acer TravelMate 8003LMi (Travelmate 8000)
Prozessor	Intel Pentium M 1.6GHz
Hauptspeicher	512MB DDR
Festplatte	60GB Ultra ATA/100 HDD
Netzwerkkarte	Broadcom NetXtreme Gigabit Ethernet Intel PRO/Wireless 2200BG Network Connection
Betriebssystem	Windows XP Professional

Tabelle 3: Hardwarebeschreibung Travelmate 8000

Computerbezeichnung	Medion 8008
Prozessor	Intel Pentium 4 2,6 GHz
Mainboard	MD-5000 Version 1.2
Hauptspeicher	512 MB DDR
Festplatte	SiS 160GB
Netzwerkkarte	SiS 900-basierte PCI-Fast Ethernet-Adapter
Betriebssystem	Windows XP Home

Tabelle 4: Hardwarebeschreibung Medion