Fakultät für Informatik
und Automatisierung

Vergleich unterschiedlicher Web-Technologien anhand der prototypischen Implementierung einer Einschreibeplattform

Studienjahresarbeit
November 2002

Kai Stammerjohann
Studiengang Informatik
Matrikelnummer 25994

Betreuer
Dipl.-Inf. Thorsten Strufe

Inhaltsverzeichnis

1 Einleitung ... 3
2 Aufgabenstellung ... 4
2.1 Praktika ... 4
2.2 Hauptseminare ... 4
3 Prozessmodell ... 5
3.1 Rollen ... 5
3.2 Prozesse ... 7
4 Datenbank ... 11
4.1 Grundlagen ... 11
4.2 Allgemeine Datenbankdetails ... 12
4.3 Hauptseminar Datenbank ... 14
4.4 Praktikum Datenbank ... 15
5 JSP Model 2 Implementierung ... 17
5.1 Grundlagen ... 17
5.2 Java Server Pages Model 2 ... 19
5.3 Implementierung ... 20
6 .NET Implementierung ... 26
6.1 Grundlagen ... 26
6.2 Implementierung ... 33
7 Vergleich ... 36
7.1 Grundlegende Merkmale ... 36
7.2 Implementierungsmerkmale ... 38
7.3 Schlussfolgerung ... 38
8 Installation und Konfiguration ... 39
8.1 Datenbankinstallation ... 39
8.2 Java Umgebung ... 39
8.3 .NET Umgebung ... 40
9 Links ... 41

1 Einleitung

Das Internet gewinnt immer mehr an Bedeutung. Die Zahlen der Web-Seiten und der Internetnutzer steigen stetig an. Bald ist das Internet nicht mehr aus unserem Leben wegzudenken. Logische Schlussfolgerung ist, dass wir immer mehr Aufgaben mit Hilfe des Internets lösen. Umso automatisierbarer die Aufgaben dabei sind, umso besser lassen sie sich mit Hilfe des Internets umsetzen. Generell lassen sich organisatorische Aufgaben einfach automatisieren und eignen sich daher gut für eine Internetlösung.

Diese Arbeit beschäftigt sich mit der Umsetzung einer internetbasierten Einschreibeplattform. Dazu wird anfangs auf die Funktionalität des Systems eingegangen und die einzelnen Funktionen werden kurz vorgestellt. Im nächsten Teil geht es um die Benutzung. In diesem Abschnitt werden die einzelnen Rollen, die dazugehörigen Rechte und daraus resultierenden Fähigkeiten erläutert.

Mittlerweile gibt es sehr viele verschiedene Umgebungen, die als Grundlage für Internetanwendungen benutzt werden können. Stellvertretend für diese Möglichkeiten soll im Laufe dieser Arbeit zwei verschiedenen Lösungsansätzen nachgegangen werden. Das Ergebnis ist jeweils eine prototypische Implementierung eines Teils der geforderten Funktionalität des Einschreibesystems. Für die Umsetzungen kommen Java JSP (Model2) und Microsoft .NET zum Einsatz. Auf die für die jeweilige Umsetzung notwendigen Grundlagen wird genauso eingegangen wie auf die Details die letztlich die Umsetzung darstellen.

Eine zentrale Rolle aller Internetanwendungen spielen die Datenbanken. Aus diesem Grund widmet sich ein Kapitel diesem Thema. Es werden die notwendigen Grundlagen und die Umsetzungen die zur Lösung der Anforderungen notwendig sind erläutert. Beide Realisierungen benutzen dabei aus Gründen der Komplexität eine gemeinsame Datenbankstruktur.

2 Aufgabenstellung

Die TU-Ilmenau bietet unter anderem im Fachgebiet Telematik verschiedene Praktika und Hauptseminare an. Aus verschiedenen Gründen ist es zwingend notwendig, dass sich interessierte Studenten vor einer Teilnahme an einer solchen Lehrveranstaltung entsprechend anmelden bzw. eintragen. Diese Einschreibeprozesse durch weitgehende Automatisierung effizienter für Studenten und Mitarbeiter zu gestalten, ist das Ziel dieser Einschreibeplattform.

2.1 Praktika

Unter Praktika sind einmalige Übungen mit verstärkt praktischen Charakter zu verstehen. Analog zu normalen Übungen gibt es einen Betreuer der das Praktikum leitet und mehrere Studenten die an diesem Praktikum teilnehmen. Aufgrund der stark begrenzten Platz- und Raumkapazitäten und ständig zunehmender Studentenzahlen ist es zwingend erforderlich, dass sich alle Praktika periodisch wiederholen, also beispielsweise im Wochenrhythmus stattfinden. Je mehr Studenten durch ein Praktikum angesprochen werden desto schwieriger ist es, einen gemeinsamen Termin zu finden, an dem möglichst viele Interessenten teilnehmen könnten. Aus diesem Grund gibt an jedem Semesterbeginn die Veranstaltungsplanung Zeiträume vor, an denen Praktika stattfinden dürfen. Außerhalb dieser Zeiträume würden Praktika dementsprechend keinen Sinn ergeben, da Studenten nicht teilnehmen könnten.

2.2 Hauptseminare

Hauptseminare sind Veranstaltungen, an denen Studenten für Studenten und Universitätsangestellte wissenschaftliche Vorträge halten bzw. Diskussionen führen sollen. Dazu nehmen an einer Hauptseminarveranstaltung mehrere Studenten mit ihren Betreuern teil. Jeder teilnehmende Student bekommt ein eigenes Thema, wozu er eine wissenschaftliche Ausarbeitung entwickeln und einem Vortrag halten muss. In den meisten Fällen können sich Studenten das Thema über das sie referieren möchten aus einem größeren fakultätsgebundenen Themenkatalog aussuchen.

3 Prozessmodell

In diesem Kapitel werden alle vorkommenden Rollen beschrieben. Dazu werden zu jeder Rolle die zugehörigen Teilaufgaben aufgezählt. Folgende Rollen werden beschrieben: *Planer, Betreuer, Administrator, Teilnehmer*, *Benutzer* und *Anonym*.

Basierend auf den verschiedenen Rollen werden im darauf folgenden Abschnitt die wichtigsten Prozesse erklärt. An den Stellen wo sich die Praktikaeinschreibung von der Hauptseminareinschreibung unterscheidet, wird getrennt auf die Merkmale eingegangen.

3.1 Rollen

3.1.1 Übersicht

Abb. 3-1 stellt die Mitgliedschaften der einzelnen Rollen dar. Die Rollendetails werden in den folgenden Kapiteln erläutert. Unmittelbar abhängig von den Mitgliedschaften sind die Rechte. Aus **Abb. 3-1** geht dementsprechend hervor, dass z.B. *Teilnehmer* neben den Teilnehmerrechten auch alle Rechte von *Benutzer* und *Anonym* hat.

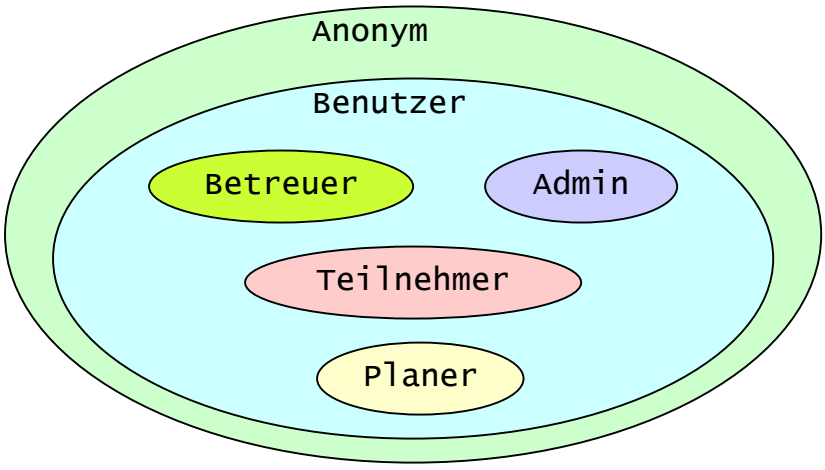

Abb. 3-1: Mitgliedschaften und Rechteverteilung

Abb. 3-2 stellt dar, wie Gruppenzugehörigkeiten entstehen. Der Gruppe *Teilnehmer* kann nur angehören, wer sich vorher für ein Praktikum oder ein Hauptseminar eingetragen kann. Zu den Gruppen *Admin*, *Betreuer* und *Planer* gehören alle Benutzer, die von einem Administrator der jeweiligen Gruppe zugeordnet wurden.

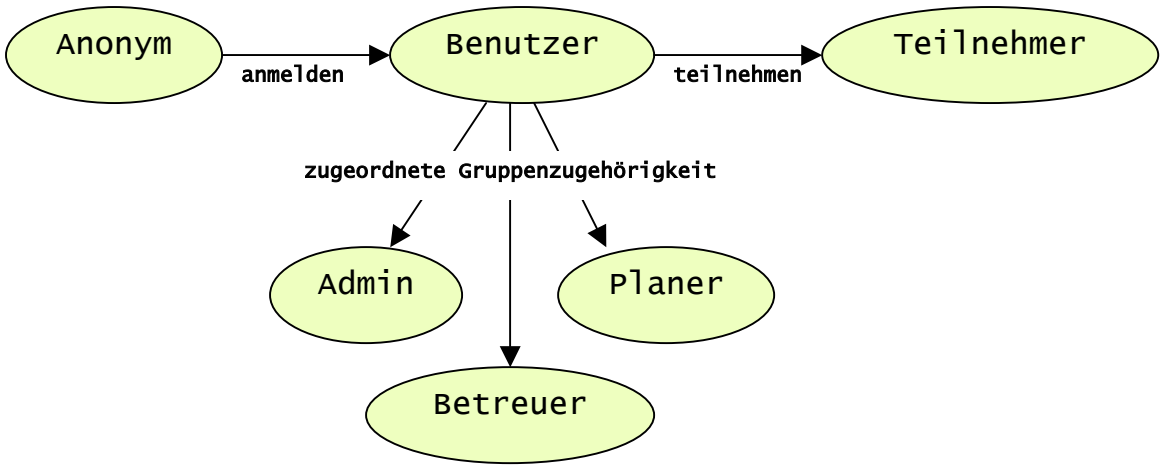

Abb. 3-2: Gruppenzugehörigkeitsübergänge

3.1.2 Administrator

Administratoren sind für die Rechtevergabe verantwortlich. Ein Administrator kann Benutzer zu Gruppen hinzufügen bzw. Benutzer aus Gruppen entfernen.

3.1.3 Planer

Praktika können nur zu ganz bestimmten Uhrzeiten, an ganz bestimmten Wochentagen (beispielsweise montags zwischen 7 und 15Uhr) stattfinden. Benutzer die Mitglied in der *Planer* Gruppe sind können diese Zeitspannen in das System eintragen. Anschließend können innerhalb dieser Zeitspannen Praktika erstellt werden. Die Planerrolle existiert nur für den Praktikaeinschreibeteil.

3.1.4 Betreuer

Betreuer sind in der Lage Praktika oder Hauptseminare zu erstellen, sie zu editieren oder sie zu löschen. Dabei kann jeder Betreuer nur seine eigenen Praktika bzw. Hauptseminare beeinflussen.

3.1.5 Teilnehmer

Benutzer die sich für ein oder mehrere Praktika eingetragen haben, werden in die Gruppe *Teilnehmer* aufgenommen. Damit bekommen sie das Recht, ihre Eintragungen zu editieren oder gegebenenfalls zu entfernen.

3.1.6 Benutzer

Jeder der sich bei der Einschreibeplattform registriert hat, ist automatisch in der Gruppe der *Benutzer*. Benutzer sind dem System also bekannt. Sie können sich für Hauptseminare oder Praktika eintragen, wodurch sie Teilnehmerrechte erlangen würden.

3.1.7 Anonym

In der Anonymgruppe sind alle Benutzer, die kein Login besitzen oder sich noch nicht angemeldet haben, dem System also nicht bekannt sind.

3.2 Prozesse

Abb. 3-3 visualisiert einen Teil der Prozesse mittels Use Cases. Der Einfachheit halber werden nur die wichtigsten Prozesse dargestellt: administrative Prozesse wurden außen vor gelassen.

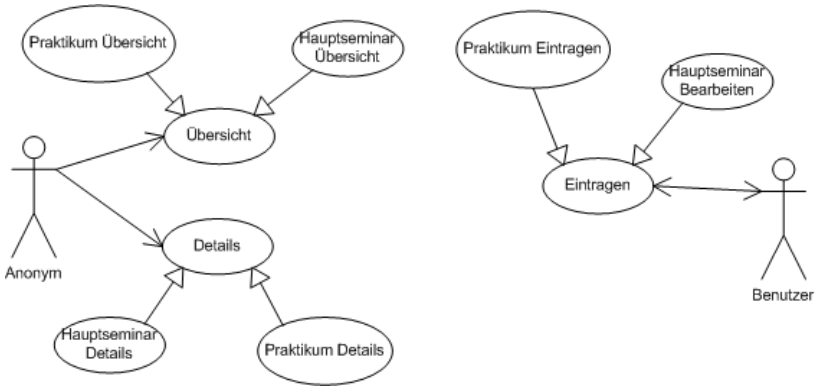

Abb. 3-3: Benutzerprozeß Use Cases

3.2.1 Übersicht

In der Praktika bzw. Hauptseminar Übersicht werden alle verfügbaren Praktika bzw. Hauptseminare mit ihrem Titel und einer gekürzten Inhaltsangabe nach Fachgebieten geordnet aufgelistet. Sowohl die Praktika als auch die Hauptseminarübersicht kann von allen Gruppen benutzt werden. Eine Anmeldung ist nicht notwendig. **Abb. 3-4** stellt eine Implementierungen der Übersichtsseite dar.

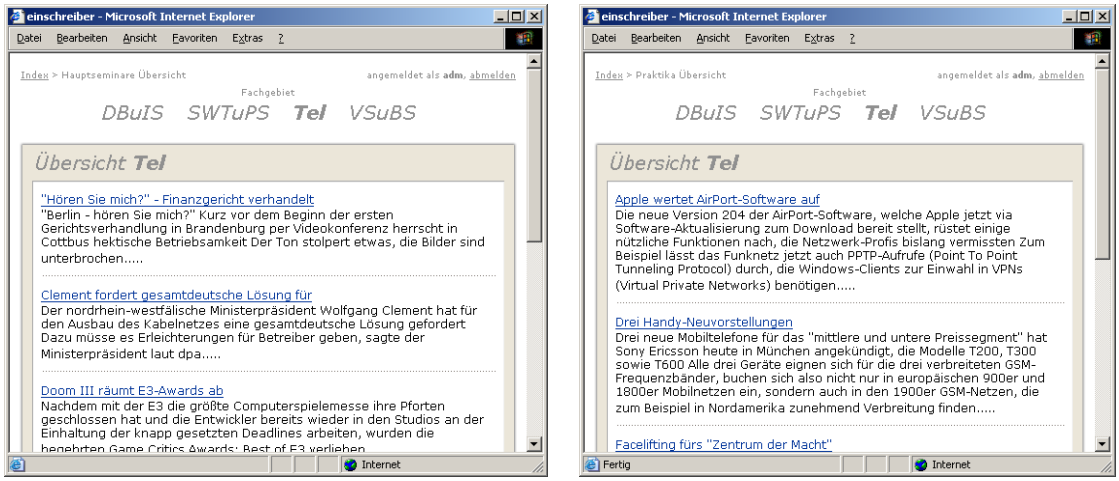

Abb. 3-4: Übersichtsseite (links Hauptseminare, rechts Praktika)

3.2.2 Details

Die Detailansicht liefert genauere Informationen zu einem Hauptseminar bzw. Praktikum. Neben dem Titel und dem gesamten Inhalt werden Termine, Ortsangaben und andere organisatorische Details gezeigt. Die Detailansicht kann von allen Gruppen benutzt werden. **Abb. 3-5** zeigt eine Implementierung einer Detailansicht.

Abb. 3-5: Detailansicht (links Praktika, rechts Hauptseminar)

3.2.3 Eintragung

Für eine Eintragung in ein Hauptseminar oder Praktikum ist es notwendig, dass man sich angemeldet hat, sich also nicht mehr in der Anonym Gruppe befindet. In welchen Gruppen man nach dem Anmelden ist, spielt dabei keine Rollen. Bis auf die Anonymgruppe haben alle Gruppen haben das Eintragungsrecht.

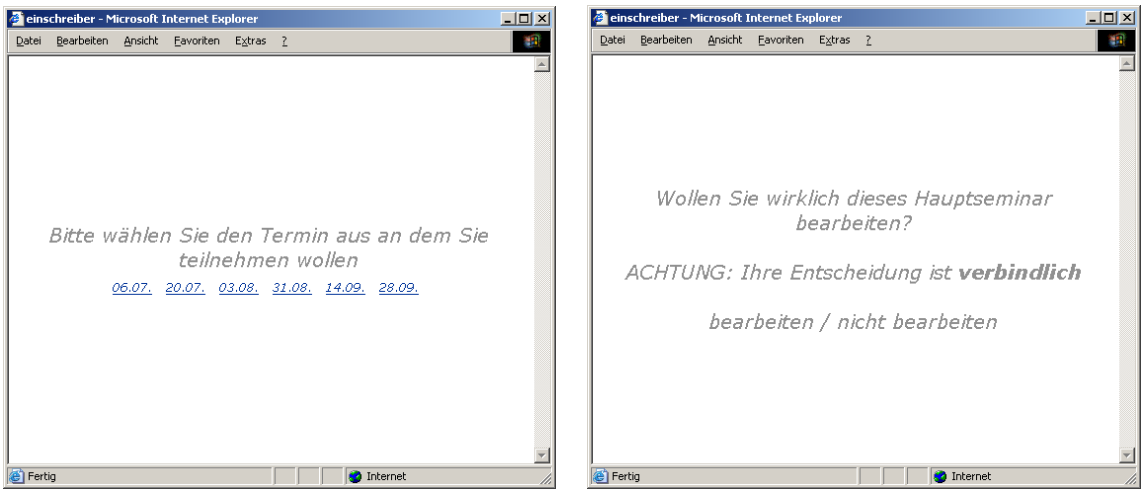

Abb. 3-6: Eintragung in Praktikum(links) und Hauptseminar(rechts)

3.2.4 Registrierung

Die Registrierung dient der eindeutigen Identifikation eines Benutzers. Nach einem erfolgreichen Login hat man Benutzerrechte und kann sich so z.B. für Praktika eintragen, was ohne Registrierung nicht möglich wäre.

Zu jedem Login gehört eine E-Mail Adresse. Damit sichergestellt werden kann, dass eine gültige E-Mail Adresse eingegeben wurde, wird nach der Eingabe der Benutzerdaten eine Bestätigungsemail an die angegebene Adresse geschickt. Die Bestätigungsemail enthält einen zu dem neu erstellten Login passenden Aktivierungslink. Durch Aufruf dieses Links wird das Login aktiviert. Solange der Link nicht ausgeführt wird bleibt das Login deaktiviert und kann nicht genutzt werden.

3.2.5 Login

Über die Login Seite können sich Benutzer anmelden. Die Rechte die sie nach einer erfolgreichen Anmeldung bekommen ist abhängig von den Gruppen, denen sie angehören.

Alle Nutzer haben nach dem anmelden mindestens die Rechte der Benutzergruppe. Über die Login Seite ist es weiterhin möglich eine bestehende Anmeldung zu beenden. Eine Loginaufforderung kommt an allen Stellen wo Aktionen ausgeführt werden sollen, die die aktuelle Gruppenzugehörigkeit nicht zulässt.

3.2.6 Administration

Verschiedene Logins können verschiedenen Rollen zugeordnet werden. Genau diese Zuordnungen der Logins zu den Gruppen (entspricht den Rollen) kann über die Loginadministrationsschnittstelle durchgeführt werden. Neben der Vergabe der Gruppenzugehörigkeiten ist es auch möglich, neue Benutzer zu erzeugen. Diese Funktionalität kann nur von Administratoren ausgeführt werden.

Neben den allgemeinen organisatorischen Aufgaben können Benutzer mit Betreuerrechten neue Praktika erstellen, alle praktikabezogenen Angaben editieren, oder es komplett aus dem Einschreibesystem entfernen. Aus nahe liegenden Gründen ist es Betreuern nur möglich, eigene Praktika zu administrieren.

Weiterhin sind Mitglieder der Planergruppe in der Lage, neue Praktikatermine freizugeben, vorhandene zu verändern oder zu löschen.

Über die Hauptseminaradministrationsseiten haben Mitglieder der Betreuergruppe die Möglichkeit, Hauptseminare zu erstellen oder zu editieren. Außerdem ist es ihnen möglich, den Status eines Hauptseminares zu beeinflussen.

4 Datenbank

Grundlage aller Verwaltungsaufgaben sind ein oder mehrere Datenbanken beziehungsweise Tabellen. Die für die Einschreibeplattform benutzte Datenbankstruktur wird in diesem Kapitel erklärt. Dazu wird auf die notwendigen Grundlagen sowie auf die Details eingegangen.

4.1 Grundlagen

Eine Datenbank ist eine Ansammlung von Daten, ein Datenbankmanagementsystem (DMBS) verwaltet diese Daten. Zur Abfrage von Daten bieten Datenbankmanagementsysteme eine standardisierte Abfragesprache: Structured Query Language (SQL). Mit SQL können auf einfache Weise komplexe Suchanfragen realisiert werden. SQL Anfragen beschreiben also eine Teilmenge der verwalteten Daten.

Neben der Zugriffsschnittstelle ist ein DBMS unter anderem auch für die Konsistenz der Daten verantwortlich. Daten werden inkonsistent wenn sie sich gegenseitig widersprechen. Zu solchen Situationen kommt es, wenn mehrere Benutzer gleichzeitig die Tabellen beeinflussen. Moderne DBMS sind in der Lage die Datensätze automatisch konsistent zu halten. Eine Möglichkeit Inkonsistenzen zu verhindern sind Relationen zwischen Tabellen. Relationen sind Abhängigkeiten zwischen einzelnen Spalten. Ein Beispiel für eine Relation ist eine Eltern-Kind Beziehung[1]. Ein Kind gibt es nur mit Eltern, ein Kind hat genau ein Elternpaar, ohne Eltern gibt es auch kein Kind. Genau solche Relationen gibt es auch zwischen Datenbanktabellen – wenn es den einen Datensatz nicht gibt, dann gibt es den anderen auch nicht (bzw. er wird entfernt).

4.1.1 Primärschlüssel

Ein Primärschlüssel ist ein Feld einer Tabelle, dass den Datensatz zudem er gehört eindeutig beschreibt. Es gibt keine zwei Datensätze die denselben Primärschlüsselwert besitzen. Somit ist ein Datensatz ist über seinen Primärschlüssel eindeutig identifizierbar. Eine Umsetzung für einen Primärschlüssel wäre eine Durchnummerierung aller Datensätze. Jeder Datensatz erhält eine andere Nummer und wird dadurch identifizierbar.

[1] mal von den technischen Einzelheiten abgesehen

4.1.2 Foreign Keys

Foreign Keys sind Felder einer Tabelle die Zeiger auf Primärschlüssel einer anderen Tabelle enthalten. Auf diese Weise wird eine n:1 Beziehung zwischen den Datensätzen hergestellt. Ein Beispiel für Foreign Keys ist folgendes Szenario: eine Tabelle enthält verschiedene Produkte und jedes Produkt hat eine eindeutige Nummer. Neben der Produkttabelle existiert eine Warenkorbtabelle, wobei die Warenkorbtabelle ein oder mehrere Produkte enthalten kann. Anstelle die komplette Produktbeschreibung in der Warenkorbtabelle zu speichern wäre es angebracht, nur die Produktnummern zu vermerken. Damit ist die Produktnummer Spalte der Warenkorbtabelle ein Foreign Key: sie enthält einen Zeiger auf einen Datensatz einer anderen Tabelle.

4.1.3 Constraints

Constraints sind Zwänge, die von Datensätzen erfüllt werden müssen. Falls sie nicht erfüllt werden können werden alle betroffenen Datensätze ungültig und ungültig gewordene Datensätze machen die Datenbank inkonsistent. Um das zu verhindern sind Constraints in der Lage ungültige Datensätze zu entfernen damit die Datenbankkonsistenz erhalten bleibt. In den folgenden Kapiteln wird immer von Constraints ausgegangen die diese automatische Konsistenzerhaltung einsetzen. Bezogen auf das Produkt – Warenkorb Beispiel bedeutet das, dass wenn ein Produkt aus der Produkttabelle gelöscht wird, automatisch alle Datensätze aus der Warenkorbtabelle entfernt werden, die das gelöschte Produkt referenzierten. Um Constraints einsetzen zu können werden Foreign Key Felder benötigt, die die entsprechenden Tabellen in eine Beziehung stellen.

4.2 Allgemeine Datenbankdetails

Alle verwendeten Tabellen haben ohne Ausnahme eine ID (*id*) Spalte. Die ID Spalte enthält für jeden Tabelleneintrag einen eindeutigen Wert und eignet sich aus diesem Grund als Primärschlüsselspalte. Über diese Werte lassen sich die Datensätze der Tabellen genau adressieren. **Abb. 4-1** stellt die in den folgenden Kapiteln genannten Abhängigkeiten zwischen den Tabellen grafisch dar.

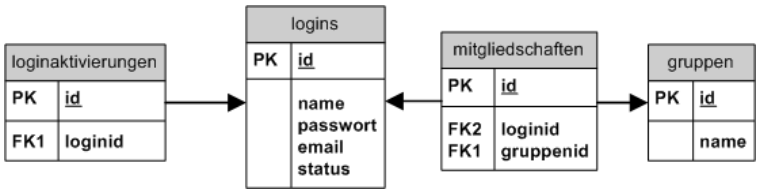

Abb. 4-1: Abhängigkeiten zwischen den allgemeinen Tabellen

4.2.1 Logins

Die zentralste Tabelle ist die Logins Tabelle. Sie enthält für jeden dem System bekannten Benutzer einen Namen (*name*), eine E-Mail Adresse (*email*), sein Passwort (*passwort*) sowie seinen Status (*status*). Von der *Logins* Tabelle sind direkt oder indirekt alle anderen Tabellen abhängig.

4.2.2 Gruppen

Jeder Benutzer (bzw. jedes Login) kann in einer oder mehreren Gruppen sein. Welche Gruppen es gibt wird in der Gruppen Tabelle festgelegt. Abhängig von den Gruppen sind die Rechte die ein Benutzer hat. Beispielsweise sind Mitglieder der Gruppe *Betreuer* in der Lage neue Praktika oder Hauptseminare zu erstellen.

4.2.3 Mitgliedschaften

Die Zuordnungen von Benutzern zu Gruppen stehen in der Mitgliedschaften Tabelle. Mitgliedschaften sind somit abhängig von Gruppen (*gruppenid*) und Logins (*loginid*): es gibt eine Mitgliedschaft nur wenn es ein bestimmtes Login und eine bestimmte Gruppe gibt. Falls das entsprechende Login ungültig wird, ergibt auch die dazu passende Mitgliedschaft keinen Sinn mehr und kann zerstört werden, oder andersrum: gibt es die entsprechende Gruppe nicht mehr, wird auch die Mitgliedschaft ungültig. Diese Abhängigkeit kann von einem DBMS mit Hilfe von Constraints[2] umgesetzt werden. Wenn ein Login gelöscht wird, kann die Datenbank automatisch alle abhängigen Mitgliedschaften löschen (ohne Login keine Mitgliedschaft).

[2] Vgl. Kap. 2.1.3

4.2.4 Loginaktivierungen

Bevor ein Benutzer dem System bekannt ist muss er registriert werden. Da neue Logins erst aktiviert werden müssen bevor sie benutzt werden können wird eine zusätzliche Tabelle gebraucht. Diese Tabelle heißt *Loginaktivierungen.* Sie enthält ein Constraint auf ein Login (*loginid*) und eine eindeutige Identifikation. Damit existieren Loginaktivierungen nur wenn es das entsprechende Login gibt. Die Bestätigungsemail[3] enthält in Form eines Links die Loginaktivierungsid. Das Login wird erst bei der Ausführung dieses Bestätigungslinks aktiviert und kann benutzt werden.

4.3 Hauptseminar Datenbank

In den Hauptseminartabellen werden die Hauptseminare und ihre Bearbeitungen verwaltet. **Abb. 4-2** zeigt die Abhängigkeiten.

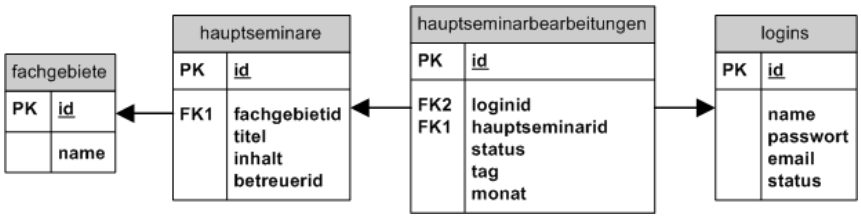

Abb. 4-2: Hauptseminartabellen und wovon sie abhängig sind

4.3.1 Hauptseminare

Alle Hauptseminare werden in der Hauptseminare Tabelle gespeichert. Sie enthält für jedes Hauptseminar das Fachgebiet (*fachgebietid*) in dem es stattfindet, den Titel (*titel*), den Inhalt (*inhalt*) und den Betreuer (*betreuerid*). Die Hauptseminare Tabelle nimmt innerhalb der Hauptseminardatenbank eine zentrale Rolle ein, da direkt oder indirekt alle anderen Tabellen von ihr abhängig sind.

An dieser Stelle sticht *sofort* ein Problem ins Auge: es ist bei den obigen Abhängigkeiten nicht mehr möglich ein Constraint zwischen der Hauptseminarbetreuerid und einer entsprechenden Loginid zu erstellen. Gäbe es dieses Constraint dann würde ein Constraintring entstehen. Bei Constraintringen können die automatischen Konsistenzerhaltungsmechanismen

[3] Vgl. Kap. 3.2.4

nicht mehr eindeutig auf die Datensätze angewendet werden, da sie sich gegenseitig beeinflussen würden.

4.3.2 Hauptseminarbearbeitungen

Für gewöhnlich können Hauptseminare von Studenten bearbeitet werden. Diese Bearbeitungen werden in der Hauptseminarberarbeitungen Tabelle gespeichert. Dazu enthält sie neben der obligatorischen ID Spalte einen Constraint auf eine Loginid (*loginid*), einen Status (*status*) der Aussagen über die Bearbeitungen zulässt und das Eintragedatum (*monat*, *tag*). Damit sind Hauptseminarbearbeitungen nur gültig wenn sie ein gültiges Login und ein gültiges Hauptseminar referenzieren.

4.4 Praktikum Datenbank

In den Praktika Tabellen stehen alle notwendigen Verwaltungsinformationen für den Praktikateil der Einschreibeplattform. **Abb. 4-3** visualisiert zum besseren Verständnis die dafür notwendigen Tabellen und ihre Beziehungen untereinander.

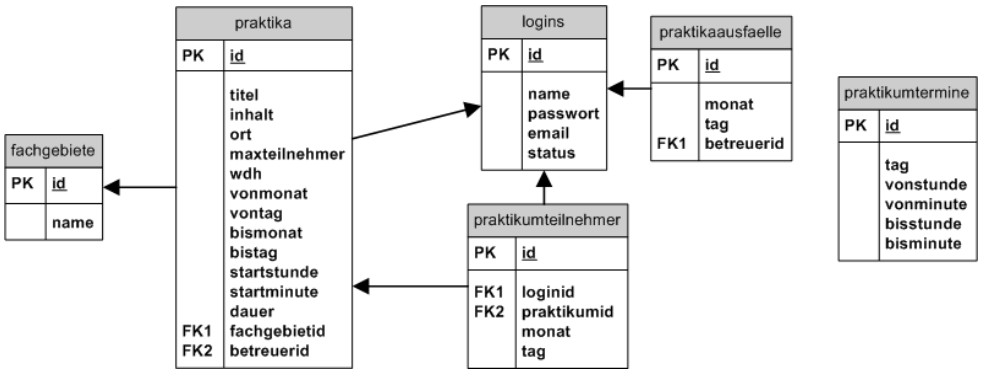

Abb. 4-3: Praktika Tabellen

4.4.1 Praktika

Die Praktika Tabelle enthält für jedes bekannte Praktikum den Titel (*titel*), den Inhalt (*inhalt*), den Ort (*ort*) wo es stattfindet, die maximale Teilnehmeranzahl (*maxteilnehmer*) pro Termin, in welchen Rhythmus die Praktikas stattfinden (*wdh*), in welchem Zeitraum die Praktika stattfinden (*vontag, vonmonat, bistag, bismonat*), wann es startet (*startstunde, startminute*), wie lange es geht (*dauer*), in welchem Fachgebiet es stattfindet (*fachgebietid*) und wer der

Betreuer ist (*betreuerid*). Der Verweis auf das Fachgebiet und den Betreuer ist als Constraint umgesetzt.

4.4.2 Praktikumteilnehmer

An einem Praktikum können an den verschiedenen Terminen an denen es stattfindet mehrere Teilnehmer teilnehmen. Die maximale Anzahl der Teilnehmer pro Termin wird durch das *maxteilnehmer* Feld in der Praktika Tabelle festgelegt. Zu jeder Teilnahme wird in der Praktikumteilnehmer Tabelle die ID des Teilnehmenden (*loginid*), die Praktikumid (*praktikumid*) sowie der spezielle Termin (*tag, monat*) an dem teilgenommen werden soll vermerkt. *Loginid* und *betreuerid* sind Constraints auf die entsprechenden Tabellen.

4.4.3 Praktikaausfaelle

Alle Praktika finden innerhalb eines festgelegten Zeitraums statt. Da es vorkommen kann, dass dieser Zeitraum Termine enthält, an denen das Praktikum nicht stattfinden kann gibt es die *Praktikaausfaelle* Tabelle. Sie enthält für jeden Termin an dem ein Praktikum nicht stattfinden darf einen Eintrag. Dazu werden der Tag (*tag*), der Monat (*monat*) sowie der Betreuer (*betreuerid*) abgespeichert. Alle Praktika die an diesem Termin von diesem Betreuer stattfinden würden fallen damit aus und werden nicht zum Teilnehmen angeboten.

4.4.4 Praktikumtermine

Wie bereits erwähnt wurde, dürfen Praktika nur an ganz bestimmten Wochentagen bzw. Uhrzeiten stattfinden. Diese legalen Praktikatermine werden in der *Praktikumtermine* Tabelle abgelegt. Alle stattfindenden Praktika müssen innerhalb eines dieser Termine liegen. Dieser Umstand ist nicht mittels SQL ausdrückbar, was der Grund dafür ist, dass die *Praktikumtermine* Tabelle keinerlei Referenzen auf andere Tabellen hat.

5 JSP Model 2 Implementierung

In diesem Kapitel geht es um die Umsetzungen der Theorie in die Praxis mit Hilfe von Javas Webtechnologien. Dabei wird neben den eigentlichen Implementierungsdetails auch auf die Grundlagen der jeweiligen verwendeten Technologie eingegangen.

5.1 Grundlagen

5.1.1 Servlets

Servlets ([SERVLET]) sind vergleichbar mit Common Gate Interfaces (CGIs), sie laufen allerdings in einer Java Virtual Machine (JVM) Umgebung innerhalb eines Webservers. Aus diesem Grund ist es zwingend erforderlich, dass Servlets Java Klassen sein müssen. Damit diese Klassen Servletfunktionalitäten bekommen, müssen sie von einer speziellen Servletklasse abgeleitet werden.

Servlets werden im Gegensatz zu anderen CGI Technologien nur einmal Instanziiert, was bedeutet dass alle Benutzer dieselbe Servletinstanz verwenden. Eine Folge aus diesem Prinzip ist, dass Servlets zustandslos sind. Da in vielen Fällen nicht ohne verschiedene Zustände gearbeitet werden kann, bieten Servlets so genanntes Sessionmanagement an. Dazu wird jedem Benutzer eine eigene bzw. eindeutige Session[4] zugeordnet. Sessions können unter anderem den Zustand eines Benutzers speichern[5].

Der Benutzer eines Servlets ist ein Browser, der direkt mit dem Servlet kommuniziert. Zusammen mit dem oben erwähnten Sessionmanagement ist ein Servlet in der Lage, für den jeweiligen Benutzer situationsabhängige Seiten zu generieren. **Listing 5-1** zeigt eine einfache Servletklasse.

```
public class Servlet extends HttpServlet
{
        public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException
        {
                Printwriter out = response.getWriter();
                out.println("<html><body><b>" + new Date() + "</b></body></html>");
        }
}
```

Listing 5-1: einfache Servletklasse

[4] Session (engl.) bedeutet in diesem Zusammenhang Sitzung

[5] ein Zustand wird im allgemeinen durch Variablen repräsentiert

Für die Erzeugung der passenden HTML Seiten müssen Servlets zwangsläufig den dazu notwendigen HTML Code enthalten. Folge dieser Tatsache ist eine Mischung von Anwendungslogik und Darstellung. Diese Untrennbarkeit macht die Wartung eines Servlets sehr problematisch. Änderungen am Aussehen der Seite sind nur durch Änderungen am Servletcode realisierbar. Eine Lösung dieses Fiaskos ist die Trennung von Anwendungslogik und Darstellung.

5.1.2 Java Server Pages

Servlets enthalten im Anwendungscode den Darstellungscode, bei Java Server Pages ist es genau umgedreht: der Darstellungscode enthält den Anwendungscode. Java Server Pages sind damit sehr vergleichbar zur ASP ([ASP]) Technologie von Microsoft. Aufgrund dieser Tatsache haben Java Server Pages dieselben Nachteile wie Servlets: Anwendung und Darstellung sind nicht ausreichend voneinander getrennt, was jegliche Wartung verkompliziert. **Listing 5-2** zeigt eine einfache Java Server Page.

```
<%@page import="java.util.Date"%>

<html>
          <body>
                    <b>
                              <%= new Date() %>
                    </b>
          </body>

</html>
```

Listing 5-2: einfache JSP

Interessant an Java Server Pages ist, dass sie ähnlich wie normaler Sourcecode von einem Java Server Page Compiler übersetzt werden müssen bevor sie benutzt werden können. Intern erzeugen Java Server Page Compiler aus einer .JSP Quelldatei eine Servletklasse. Diese generierte Servletklasse wird von einem herkömmlichen Javacompiler zu einem ausführbaren Javabinary übersetzt um dann von der im Webserver laufenden Servletengine ausgeführt zu werden.

5.2 Java Server Pages Model 2

JSP Model 2 ([JSP2]) versucht die Nachteile die Servlets und Java Server Pages haben zu umgehen. Dazu verwendet es beide Technologien zusammen: die Anwendungslogik wird innerhalb von Servlets realisiert, für die Darstellung der resultierenden Daten kommen Java Server Pages zum Einsatz. Auf diese Weise wird eine Trennung von Darstellung und Anwendungslogik erreicht. Beispielsweise kann das Aussehen einer Seite nun unabhängig vom Anwendungscode beeinflusst werden. Das Servlet spielt die Rolle des Controllers, die zugehörige JSP die Rolle der View. Mit dieser Aufgabenteilung wird das Model View/Controller Prinzip umgesetzt. Daraus Folgt, dass zu jedem Servlet mindestens eine Java Server Page existieren muss, die die Darstellung übernimmt. In **Abb. 5-1** wird das Zusammenspiel von Servlet und Java Server Page illustriert.

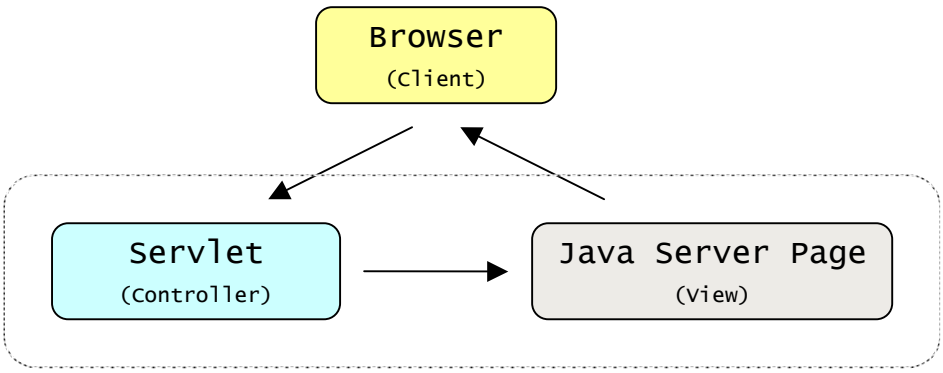

Abb. 5-1: Aufgabenverteilung JSP Model 2

Ein Browser startet den Kreislauf indem es einen Seitenrequest an das Servlet sendet. Das Servlet berechnet daraufhin (möglicherweise situationsabhängig) notwendige Daten und übergibt sie der passenden Java Server Page. Die Java Server Page kennt die Daten die es bekommt und weiß wie sie darzustellen sind, es kann die Seite rendern. Abschließend muss die so erzeugte Seite noch zum Client gesendet werden.

Für den Datenaustausch zwischen dem Servlet und der JSP gibt es verschiedene Möglichkeiten. Eine Möglichkeit ist, die Daten innerhalb des Servlets in die zugehörige Session zu schreiben. Da die JSP Zugriff auf dieselbe Session hat kann sie die Daten dort wieder auslesen um sie anschließend darzustellen. Für den Servlet – JSP Datenaustausch ist es leider oft unumgänglich Java Code[6] in die JSP Seiten einzubetten, was zur Folge hat, dass die

[6] so genannte Scriplets

Trennung von Daten und Darstellung wieder aufgeweicht wird. Um Code innerhalb Java Server Pages zu minimieren, ist es möglich Java Beans zu benutzen. Java Beans sind Objekte die von Java Server Pages dargestellt werden können, sie müssen dazu eine spezielle Schnittstelle aufweisen. Java Beans werden aufgrund der Komplexität nicht bei der Einschreibeplattformimplementierung eingesetzt.

5.3 Implementierung

Die JSP Model 2 Implementierung der Einschreibeplattform verwendet die Model 2 Vorgaben: in Servlets werden die Daten erzeugt, die daraufhin von Java Server Seiten dargestellt werden. Eine Trennung von Anwendungslogik und Darstellung ist somit gewährleistet.

5.3.1 Architektur

Die Einschreibeplattform besteht aus verschiedenen Schichten. Sie hat allgemeine Schichten und Schichten für die speziellen Einschreibekategorien.

Abb. 5-2 zeigt die einzelnen Teile aus denen die Einschreibeplattform besteht.

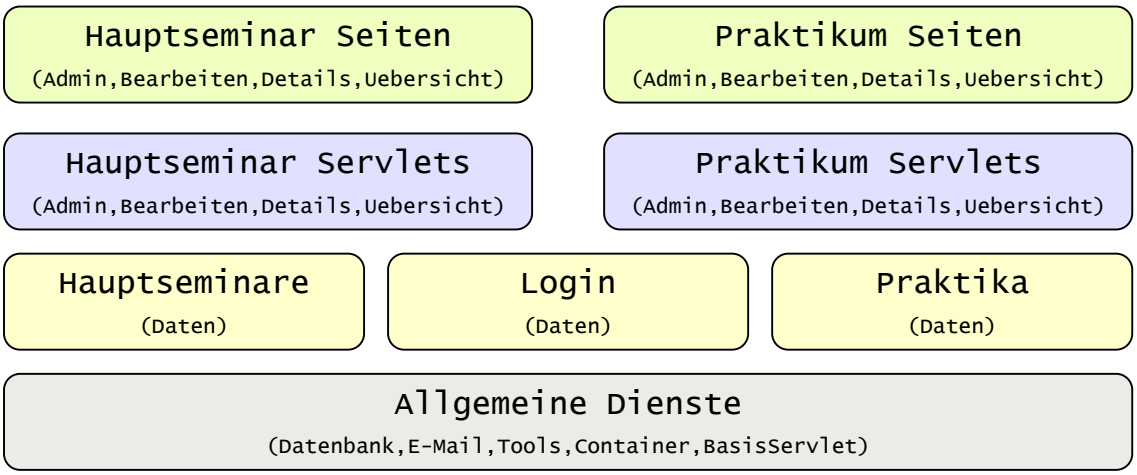

Abb. 5-2: Einschreibeplattformmodule

5.3.2 Allgemeine Bestandteile

Die Basisschicht beinhaltet allgemeine Dienste bzw. allgemeine Klassen, die keinerlei Spezialisierungen auf eine Einschreibekategorie unterliegen. All diese Klassen stehen im

`de.tuilmenau.einschreiben` Package. Alle verwendeten Servlets sind von der `de.tuilmenau.einschreiben.Servlet` Klasse abgeleitet. Sie erben dadurch Funktionen zum Umgang mit JSPs sowie eine allgemeine Fehlerbehandlungslogik.

Eine wichtige allgemeine Klasse ist die Datenbankklasse (`de.tuilmenau.einschreiben.Datenbank`). Sie enthält Methoden für den Umgang mit Datenbanken. Dazu gehören Methoden für den Verbindungsaufbau und Methoden zur Abfrage der Datenbank. Grundsätzlich gibt es zwei verschiedene Datenbankabfragetypen: Abfragen die Datensätze als Ergebnis zurückliefern und Abfragen die den Datenbankinhalt verändern. Der Zweite Abfragetyp liefert keine Datensätze zurück. Die Funktion für die Abfrage von Datensätzen ist die `query` Methode. Sie bekommt als Argument ein SQL[7] Statement und liefert das Ergebnis mittels eines `Query` Objektes zurück. Die `Query` Klasse bietet Zugriff auf die zurückgegeben Datensätze. Für das updaten von Datensätzen gibt es die `execute` Methode. Sie bekommt ebenfalls ein SQL Statement, liefert aber im Unterschied zur `query` Methode nichts zurück. Eine zusätzliche Aufgabe der Datenbankklasse ist die Sicherstellung der Verbindung zum Datenbankserver. Da eine Verbindung jederzeit unterbrochen werden kann, prüft sie vor jeder Datenbankabfrage ob die Verbindung noch gültig ist. Falls die Verbindung unterbrochen wurde, wird eine neue Verbindung aufgebaut.

Der Verbindungsaufbau zu einer Datenbank ist ein zeitaufwendiger Prozess. Aus diesem Grund ist es nicht ratsam für jede Datenbankoperation eine neue Verbindung aufzubauen. Es gibt verschiedene Lösungen um den Verbindungsaufbauzeitbedarf zu minimieren. Eine Möglichkeit sind Database Connection Pools. Ein Connection Pool enthält mehrere geöffnete Verbindungen zu einer Datenbank. Benötigt die Anwendung eine Verbindung wird sie dem Pool entnommen und kann genutzt werden. Der Öffnungsprozess wird dadurch umgegangen bzw. zu einem anderen (günstigeren) Zeitpunkt durchgeführt. Die Einschreibeplattform benutzt eine andere Methode zur Vermeidung von Verbindungsaufbauverzögerungen: es wird vom gesamten System immer nur genau eine Verbindung genutzt, alle SQL Operationen laufen über diese eine Verbindung. Es wird an keiner Stelle eine neue Verbindung benutzt, dadurch kommt es an keiner Stelle zu Verzögerungen. Um dies zu erreichen ist es notwendig,

[7] Vgl. Kap. 4.1

dass alle Methoden der Datenbankklasse statisch[8] sind. Dadurch wird erreicht dass alle Objekte auf derselben Datenbankinstanz[9] operieren.
Neben der Datenbankklasse gibt es noch Datenzugriffsklassen. Es gibt eine allgemeine (`de.tuilmenau.einschreiben.Daten`) und spezialisierte Datenzugriffsklassen. Die Datenklassen kapseln die 'Datenbeschaffung'. Benutzer der Datenklassen bekommen auf diese Weise keinen direkten Kontakt mit der Datenbank. Theoretisch könnten verschiedene Backends eingesetzt werden, jediglich die Datenklassen müssten intern darauf abgestimmt werden.

Für den Datenaustausch mit den JSP Seiten gibt es verschiedene Containerklassen. Containerklassen (`de.tuilmenau.einschreiben.Content`) enthalten ein oder mehrere Datensätze (`de.tuilmenau.einschreiben.Row`). Zum durchlaufen der Datensätze existieren verschiedene Zugriffsmethoden.

Beispielsweise wird für die Übersichtsansicht ein Container benutzt, der zu jedem Eintrag einen Titel, eine Beschreibung und einen Link für Details enthält. Diese Containerklassen sind also vergleichbar mit mehrdimensionalen Arrays.

In **Abb. 5-2** wird das Zusammenspiel der Komponenten am Beispiel der Praktikaübersichtsansicht[10] dargestellt. Alles fängt mit dem Aufruf der Übersichtsseite eines Browsers an (1.). Die Übersichtservletklasse erstellt daraufhin ein `Content` Objekt welches die Übersichtlinks enthält. Dieses Objekt wird von der `getLinks` Funktion der spezialisierten `Daten` - klasse generiert. Sie bekommt als Parameter die Fachgebietsid, damit nur Praktika die dem ausgewählten Fachgebiet angehören dargestellt werden. Die `getLinks` Funktion wiederum führt eine SQL Abfrage mithilfe der `Datenbank` – Klasse durch, das Ergebnis sind die Übersichtsdaten. Anschließend werden diese Daten in die Containerklasse kopiert. Das Servlet besitzt nun alle Daten zum Anzeigen der Übersichtsseite und ruft aus diesem Grund mittels `include` die passende JSP auf, die die Darstellung übernimmt. Das Ergebnis wird zurück zum Browser gesendet.

[8] erkennbar an `static` Eigenschaft bei Funktionsdeklaration

[9] keine echte Objektinstanz, Vgl. Singleton Pattern

[10] Vgl. Abb. 3-4

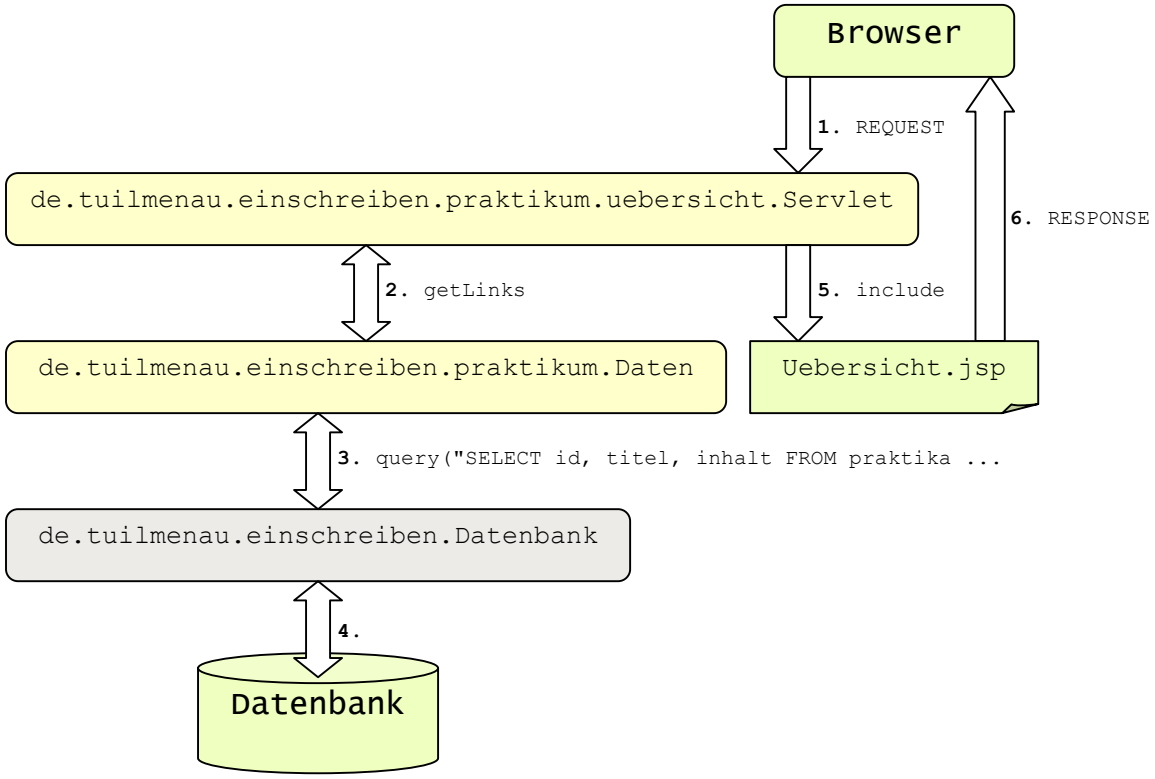

Abb. 5-2: Informationsfluss Beispiel

5.3.3 Spezielle Bestandteile

Die Einschreibeplattform besteht neben den allgemeinen Teilen aus 3 speziellen Teilen. Diese Teile realisieren die Einschreibungen für Hauptseminare, Praktika und die Erstellung von Logins. Jede dieser 3 Teile besteht aus verschiedenen Unterbereichen. In **Abb. 5-2** sind die 3 Hauptteile mit ihren Unterbereichen dargestellt. Jeder Unterbereich (Admin, Details, …) wird durch ein Servlet und mindestens einer JSP repräsentiert.

Die Aufgaben der aufgezählten Hauptseminar und Praktika Seiten entsprechen im Wesentlichen dem, was in Kapitel 3.2 erläutert wurde. Über den *Login* - Bereich können sich Benutzer anmelden oder neue Logins erstellen. Das Erstellen eines Logins wird von der *Erstellen* Seite realisiert. Der Benutzer ruft über die automatisch generierte Aktivierungs- E-

Mail[11] das *Aktivieren* Servlet auf. Daraufhin wird das vorher erstellte Login aktiviert und kann zur Anmeldung genutzt werden.

[11] Vgl. Kap. 3.2.4

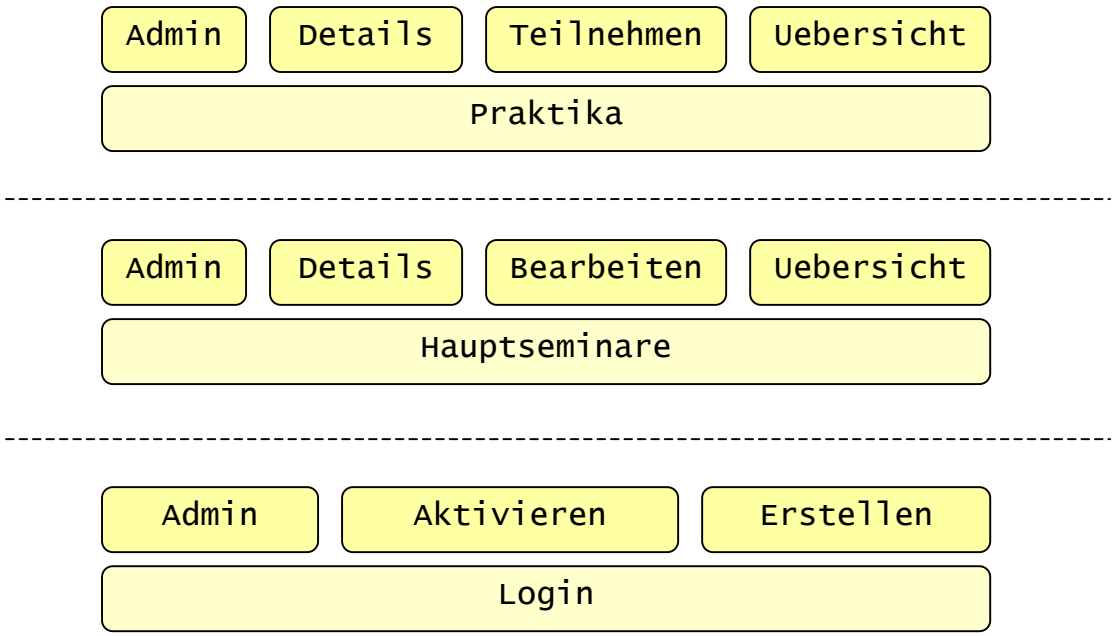

Abb. 5-2: Bestandteile der einzelnen Kategorien

6 .NET Implementierung

In diesem Teil der Ausarbeitung wird die Implementierung der Einschreibeplattform mittels .NET behandelt. Dazu wird zunächst auf .NET Grundlagen eingegangen. Im darauf folgenden Abschnitt werden neben den Architekturmerkmalen auch speziellere Implementierungsdetails erläutert. Randzubemerken wäre dass die .NET Implementierung nicht denselben kompletten Funktionsumfang enthält wie es bei der Java Version der Fall ist.

6.1 Grundlagen

Ähnlich zu SUN's J2EE Plattform stellt das Microsoft .NET Framework ([NET]) eine standardisierte[12] Infrastruktur für die Entwicklung von Internetapplikationen zur Verfügung. Das Framework besteht neben .NET Sprachen, Übersetzern und einer Laufzeitumgebung auch aus einer umfangreichen Klassenbibliothek. **Abb.** 6-1 stellt den Zusammenhang zwischen .NET Sprache, Übersetzer, Assembly und CLR dar.

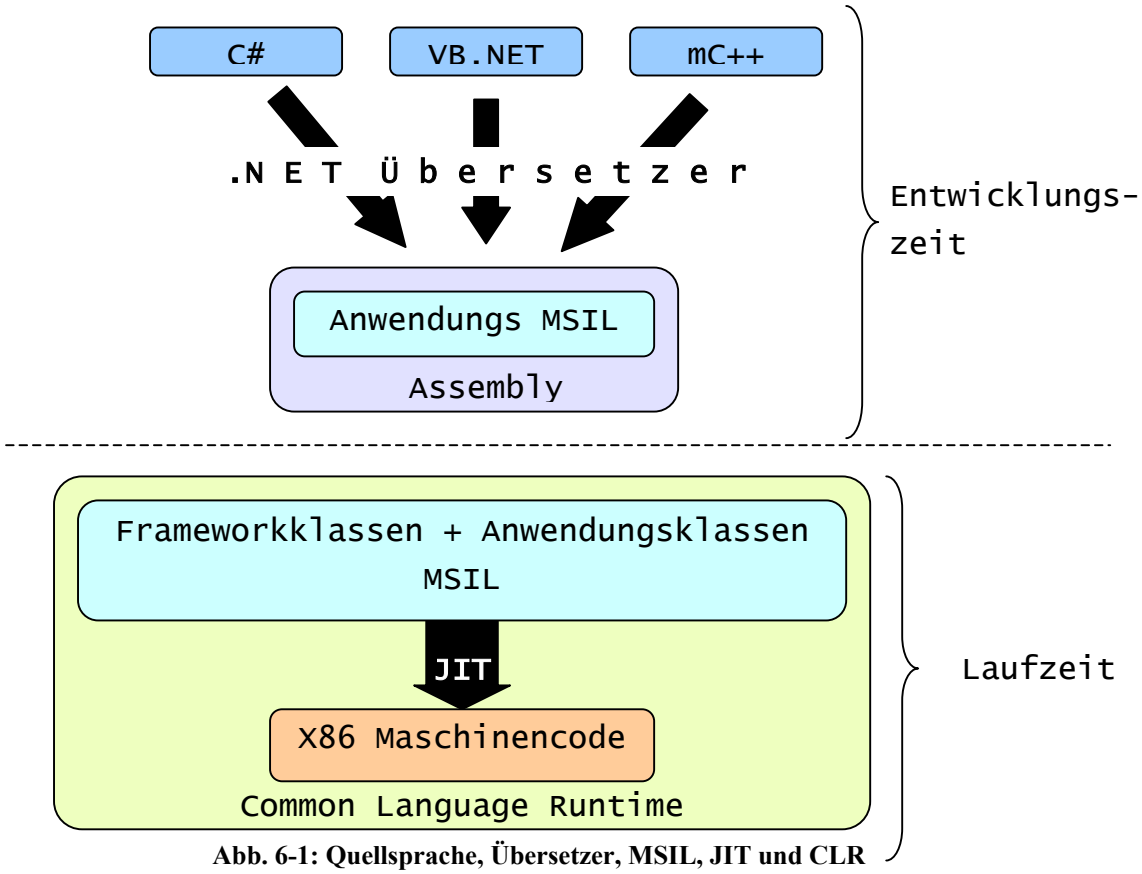

Abb. 6-1: Quellsprache, Übersetzer, MSIL, JIT und CLR

[12] die .NET CLR ist ein ECMA Standard

Für die Erstellung von dynamischen Webseiten stellt das .NET Framework verschiedene Möglichkeiten zur Verfügung. Eine Möglichkeit sind Webforms mit Webcontrols auf die im Verlaufe dieses Kapitels eingegangen wird.

Eine weitere wichtige .NET Technologie sind Webservices. Da Webservices bei der .NET Implementierung der Einschreibeplattform eine nicht unwichtige Rolle spielen wird zusätzlich auf die Erzeugung und Benutzung Derselben eingegangen.

6.1.1 Sprachen

Zusätzlich zu den .NET Standartsprachen Visual Basic.NET, C# und C++ gibt es Sprachen von Drittanbietern. Dazu zählen unter anderem Forth oder Pascal.

Alle .NET Sprachen besitzen einen Garbage Collector Mechanismus, der automatisch für die Deallokierungen[13] nicht benutzter Objekte sorgt.

Des Weiteren versuchen verschiedene Sicherheitsmechanismen .NET Anwendungen so sicher zu machen wie möglich. Dazu zählen Funktionen die zur Laufzeit Pufferüberläufe genauso erkennen wie nicht initialisierte Variablen. Ein komplexer Sicherheitsmechanismus ermöglicht die Vergabe von Rechten auf Codeebene, so dass Dieser nur von privilegierten Benutzern ausgeführt werden kann.

6.1.2 Übersetzer

Zu jeder .NET Sprache gibt es einen .NET Übersetzer (Compiler). .NET Übersetzer erzeugen aus der Quellsprache Microsoft Intermediate Language Code (MSIL). MSIL Code ist nicht direkt ausführbar und muss daher vor der Ausführung in ausführbaren Maschinencode übersetzt werden[14]. Diese Aufgabe wird von der .NET Laufzeitumgebung (Common Language Runtime – CLR) übernommen.

Das Übersetzen aller Quellsprachen in die gemeinsame Zwischensprache MSIL ermöglicht eine sprachübergreifende Anwendungsentwicklung: Module aus einer Quellsprache können

[13] Deallokierung entspricht der Freigabe des durch das Objekt belegten Speicher

[14] ähnlich dem Java Übersetzungsprozess inklusive der Java Virtual Machine

mit Modulen aus anderen Quellsprachen zusammenarbeiten. Neben der gemeinsamen MSIL Teilen sich alle .NET Programmiersprachen ein gemeinsames Typsystem (Common Type System, CTS). Das CTS legt für alle Sprachen gleichermaßen fest, wie Klassen und primitive Datentypen im MSIL repräsentiert werden, was eine wichtige Grundlage für die Sprachunabhängigkeit darstellt.

.NET Compiler erzeugen beim Übersetzen des Quellcodes so genannte Assemblies. Assemblies sind kleine Archive die neben dem zugehörigen MSIL Code noch Funktionalität zur Wahrung der Sicherheit und der Konsistenz beinhalten. Zum einen werden Assemblies signiert und sind dadurch gegen Veränderung geschützt, zum anderen enthalten sie Versionisierungsinformationen, die verhindern, dass Probleme bei der Zusammenarbeit von Komponenten mit verschiedenen Versionen auftreten.

6.1.3 Laufzeitumgebung

Im vorangegangen Absatz wurde erwähnt, dass der von den Compilern erzeugte MSIL Code nicht direkt ausführbar ist. Daraus folgt, dass er vor seiner Ausführung ausführbar gemacht werden muss. Diese Übersetzung von MSIL in (optimierten) x86er Maschinencode wird zur Laufzeit von der Common Language Runtime (CLR) durchgeführt.

Da Übersetzungsvorgänge zeitaufwendige Prozeduren sein können, wird von der CLR nur der MSIL Code übersetzt, der unmittelbar zur Ausführung benötigt wird. Diese Vorgehensweise wird als Just In Time Übersetzung (JIT) bezeichnet.

Der Sinn des Umweges über die MSIL Zwischensprache liegt in der dadurch erreichten Plattformunabhängigkeit: MSIL Code enthält keine plattformspezifischen Fragmente. Ein anderer Vorteil dieser Vorgehensweise sind die Optimierungsmöglichkeiten der Laufzeitumgebungen, so können sie in Abhängigkeit von ihrer Plattform den MSIL Code angepasst optimieren.

6.1.4 Klassenbibliothek

Die funktionelle Grundlage der .NET Plattform stellt die umfangreiche Klassenbibliothek dar. Sie besteht aus vielen kleineren oder größeren Werkzeugen, die die normale Anwendungserstellung um ein vielfaches erleichtern und die Entwicklungszeit dadurch

wesentlich verkürzen können. Die Klassenbibliothek ist damit vergleichbar zur Java Development Kit Klassenbibliothek. Ein grossteil dieser Klassen lässt sich über die Entwicklungsumgebung visuell bearbeiten und an die speziellen Gegebenheiten anpassen, was die Benutzung zusätzlich vereinfacht.

6.1.5 Internetanwendungen

.NET Internetanwendungen bestehen aus Webforms und Webcontrols. Webforms sind herkömmliche Internetseiten, Webcontrols sind Steuerelemente die in Webforms enthalten sein können.

Zu jedem Webcontrol gehört eine .NET Klasse[15]. Diese Webcontrolklasse repräsentiert das Webcontrol, steuert sein Verhalten und bietet Schnittstellen für einen Datenaustausch an. Während dem Entwurf des Webforms wird parallel eine entsprechende Webformklasse erzeugt. Sie bekommt für jedes enthaltende Webcontrol die zugehörige Webcontrolklasse als Variable hinzugefügt. Auf diese Weise entsteht ein komplett objektorientiertes Abbild der erzeugten Internetseite. Im Normalfall werden die Webformklassen mit der Anwendungslogik ergänzt.

Für die grafischen Eigenschaften der Seiten existiert parallel zu jedem Webform eine ASP.NET (.aspx) Datei. Diese Dateien enthalten Angaben zur Größe, Position oder anderen grafischen Eigenschaften der Steuerelemente. ASP.NET Dateien sind dazu in der Lage mit der Webformklasse zu kommunizieren bzw. Daten auszutauschen. ASP.NET Dateien sind in der Semantik sowie in der Syntax sehr ähnlich zu Java's JSP Seiten. Mit anderen Worten enthält die ASP.NET Datei die Darstellungseigenschaften, die Webformklasse hingegen enthält die Anwendungslogik. Eine Trennung von Daten und Darstellung wird somit gewährleistet.

Der Zusammenhang zwischen Webform ASP.NET Datei, Webformklasse und Webformcontrol wird **Abb. 6-2** in dargestellt. Das linke obere Bild stellt dabei die ASP.NET Datei dar, sie wird visuell entworfen und enthält die grafischen Einstellungen für die Benutzten Webcontrols (in diesem Beispiel eine Tabelle, ein Knopf und ein Schild). Das rechte obere Bild zeigt die zugehörige Webformklasse. Sie enthält entsprechend der

[15] Standardwebcontrolklassen (Button, Label, Image, Table, …) sind in der .NET Klassenbibliothek enthalten

ASP.NET Datei 3 Variablen: eine Variable für die Tabelle (`System.Web.UI.WebControls.Table`), eine für den Knopf (`System.Web.UI.WebControls.Button`) und eine für das Schild (`System.Web.UI.WebControls.Label`). Die dargestellte `Page_Load` Funktion wird bei jedem Seitenaufruf aufgerufen. In diesem Beispiel füllt sie die Tabelle indem sie ein `TableRow` und ein `TableCell` Objekt zu dem Tabellen Objekt hinzufügt. Des Weiteren wird die Texteigenschaft des Labels mit der aktuellen Uhrzeit gefüllt. Zusätzlich gibt es für den Button eine Funktion, die ausgeführt wird, wenn der Button betätigt wird. Diese Funktion (`Button1_Click`) ändert die Texteigenschaft des Buttons. Das Ergebnis wird im unteren Bild gezeigt. Zusätzlich soll die objektorientierte Programmierbarkeit (rechtes oberes Bild: Quellcode der Seite) eines Webforms dargestellt werden.

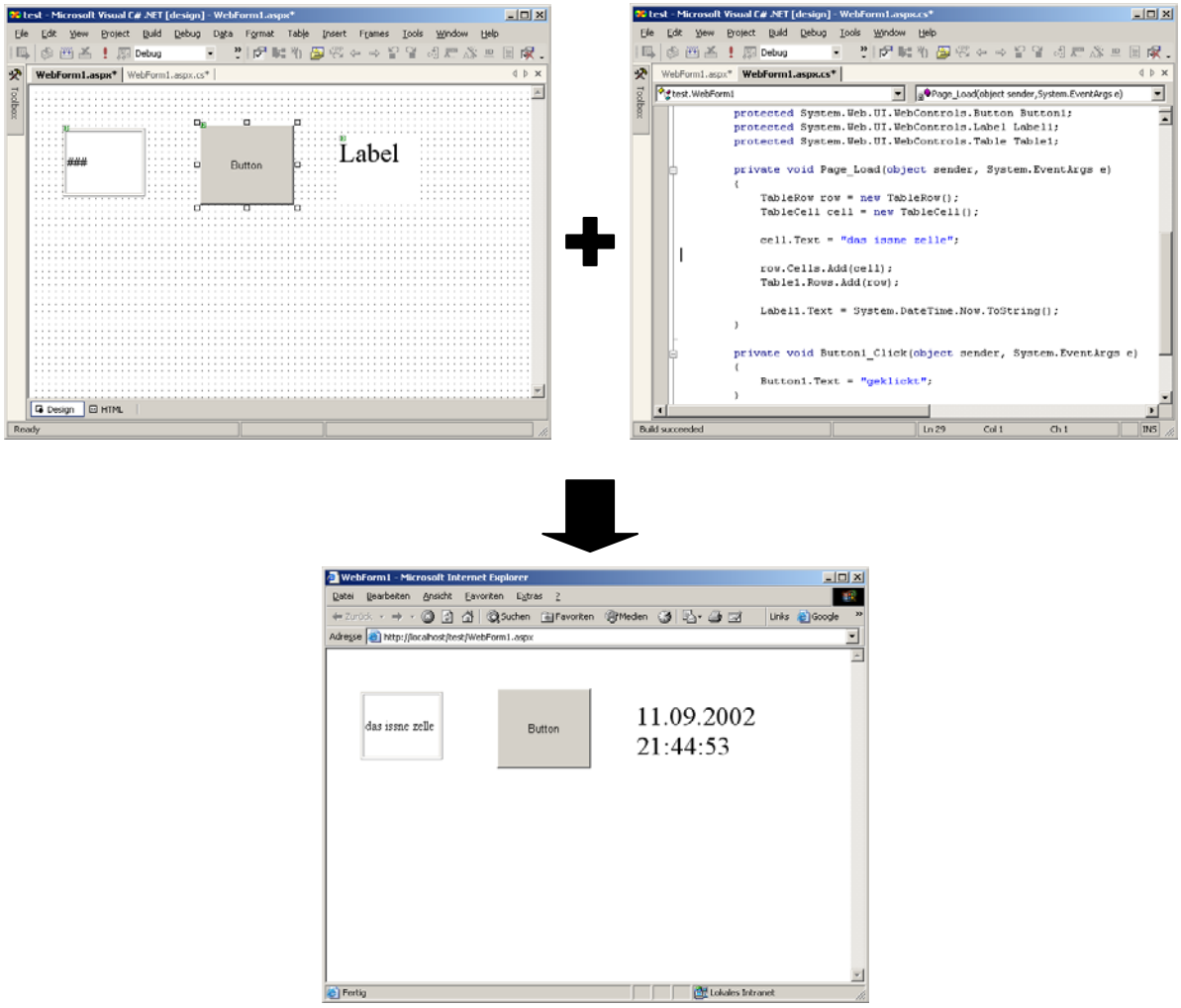

Abb. 6-2: ASP.NET Datei + Webformklasse = Internetseite

6.1.5.1 Repeater Webcontrol

Stellvertretend für Webcontrols und wegen seiner Wichtigkeit wird näher auf den *Repeater* eingegangen. Das Repeater Control dient der Visualisierung von sich wiederholenden[16] Daten. Es eignet sich daher besonders für die Darstellung von datenbankbasierten Inhalten. Dazu wird der Repeaterinstanz die Datenquelle in zugewiesen. Zur Laufzeit wird diese Datenquelle zeilenweise ausgelesen und dargestellt.

Die Darstellungform der einzelnen Datenzeilen wird mittels Templates[17] in der zum Webform gehörenden ASP Datei festgelegt. Dazu gibt es mehrere verschiedene Templates, die jeweils einen bestimmten Datenzeilentyp beschreiben. Beispielsweise gibt es Templates für die Kopfzeilen, Fußzeilen oder die eigentlichen Datenzeilen. Das Control wählt zur Laufzeit das passende Template aus, und stellt die Zeile dementsprechend formatiert dar. Diese templatebasierte Vorgehensweise ist damit sehr ähnlich zur XSL Transformation, wo ebenfalls verschiedene Templates für verschiedene Knoten existieren.

6.1.6 Webservices

Webservices sind Dienste, die über das Internet benutzt werden können. Die Kommunikation mit einem Webservice muss über das SOAP ([SOAP]) Protokoll erfolgen. Das bedeutet, dass es keine Rolle spielt, wer[18] der Dienstbenutzer ist, solange das SOAP Protokoll korrekt umgesetzt wird. Webservices gibt es nicht nur (bzw. seit) .NET, Webservices sind vielmehr ein offizieller Standart, geprägt und implementiert von SUN.

.NET Webservice Klassen sind bis auf wenige Unterschiede traditionelle Klassen. Zum einen müssen sie von einer speziellen Webservice Klasse abgeleitet sein, zum anderen müssen alle zu veröffentlichenden Methoden ein spezielles Attribut[19] (`[WebMethod]`) bekommen. **Listing 6-3** ist ein Listing eines einfachen Webservices.

[16] engl. to repeat: wiederholen

[17] engl. template: Muster

[18] Dienstnutzer können Internetanwendungen oder Clientanwendungen in Java oder C# oder ähnliches sein

[19] .NET Attribute sind vergleichbar zu traditionellen Attributen wie public, private oder protected

```
public class Service1 : System.Web.Services.WebService     //Web Service Basisklasse
{
        [WebMethod]    //markiert Methode für öffentlichen Zugriff
        public string hallo(string s)
        {
                return "hallo " + s;
        }
}
```

Listing 6-3: einfacher Webservice

Damit ein Dienst benutzt werden kann, ist es zwingend erforderlich, genaue Informationen über seine Schnittstelle zu haben. Aus diesem Grund ist jeder Dienst automatisch in der Lage, eine ausführliche Auskunft über seine Signatur zu geben. Dazu gehört neben der Nennung der einzelnen Funktionen samt Parameter und Rückgabetyp auch die Beschreibung aller verwendeten Datentypen[20]. Dieses Beschreibungsformat wird Webservice Description Language (WSDL) genannt und basiert wie alle anderen Kommunikationsprotokolle auf XML. Ausgehend von der WSDL Beschreibung ist es möglich, eine Dienstproxyklasse zu generieren. Diese Proxyklasse übernimmt die Kommunikation mit dem eigentlichen Dienstserver und besitzt dazu dieselbe Schnittstelle wie der Server auch. Auf diese Weise werden die SOAP Kommunikationsmechanismen zwischen Dienstserver und Dienstnutzer vor dem Client verborgen.

Es kann nicht immer gewährleistet werden, dass jeder Client weiß, wo sich ein passender Webservice befindet. Um dieses Problem zu lösen gibt es ein zentrales Webservice Verzeichnis, wo sich Serviceanbieter registrieren können. Dieses Verzeichnis kann dem Client Auskunft über einen zur Sucheingabe passenden Webservice geben. Der Name dieses Verzeichnisses lautet Universal Description, Discovery and Integration ([UDDI]) und ist ebenfalls standardisiert.

Ein Beispiel für einen interessanten Dienst ist der Webservice von google.com ([GOOGLE]). Über seine Schnittstelle können Suchanfragen gestellt werden. Als Antwort erhält man eine detaillierte Liste von Treffern.

[20] ausgenommen sind Standarddatentypen wie integer, string, float usw.

6.2 Implementierung

6.2.1 Architekturüberblick

Grundsätzlich entspricht die .NET Architektur der Architektur der Java Implementierung. Der größte Unterschied besteht darin, dass der komplette Datenteil als Webservice[21] realisiert wurde. Das wiederum hat zufolge, dass neben einer möglichen Verteilbarkeit der Komponenten es auch egal ist, wie die Frontends[22] (bzw. die Clients) aussehen. Um diese Tatsache zu veranschaulichen wurde neben einem Internetfrontend eine (vereinfachte) Windowsanwendung realisiert. **Abb. 6-3** stellt die Zusammenhänge dar.

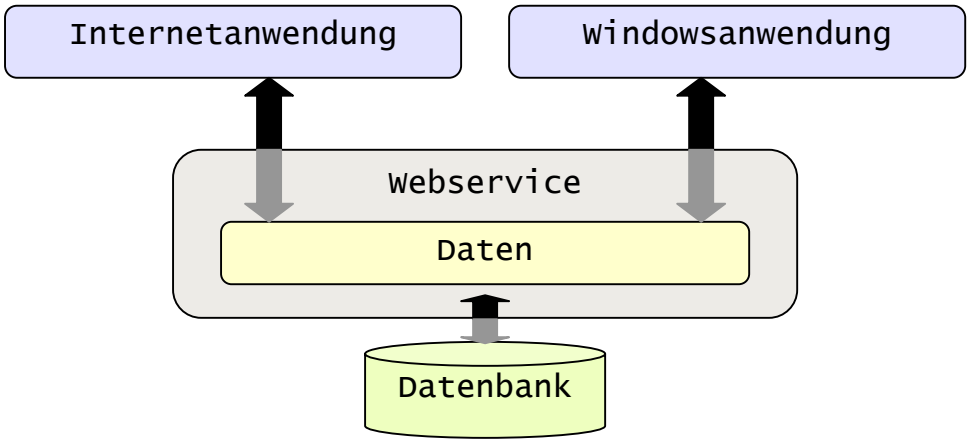

Abb. 6-3: Grundsätzliche Architektur

Wie bereits erwähnt wurde, besteht der Datenteil aus einem Webservice, der von einer Windowsanwendung sowie von einer Internetanwendung benutzt wird. Windows- und Internetanwendung bestehen analog zur Java Version aus verschiedenen Bestandteilen, jediglich der Datenteil wurde ausgelagert.

6.2.2 Datenbank

Naheliegenderweise orientiert sich der Umgang mit der Datenbank ebenfalls wie große Teile der .NET Implementierung am Java Pendant. Das bedeutet, dass es eine statische Datenbankabfrage Funktion gibt und alle Objektinstanzen somit auf einer gemeinsamen Datenbankverbindung operieren.

---

[21] siehe vorangeganger Absatz

[22] Frontend bezeichnet den sichtbaren Teil des Systems

6.2.3 Internetanwendung

6.2.3.1 Allgemeine Bestandteile

Bei den allgemeinen Steuerelementen handelt es sich zum einen um die Fachgebietsnavigationsleiste und zum anderen um das Loginstatussteuerelement. Beide Steuerelemente sind in **Abb. 6-4** dargestellt: der Loginstatus wird oben rechts angezeigt und die Navigationsleiste in der Mitte darunter. Das endgültige Aussehen dieser Steuerelemente steht erst zur Laufzeit fest, aus diesem Grund können sie nur symbolisch[23] angedeutet werden.

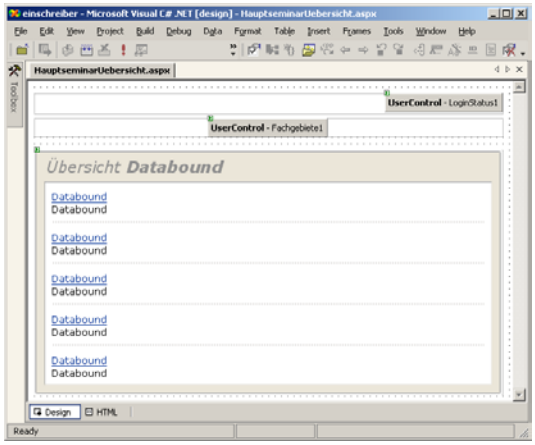

Abb. 6-4: Designansicht der Übersichtsseite

Neben den allgemeinen Steuerelementen existieren einige allgemeine Module, die wichtige Funktionalitäten enthalten. Dazu zählt die Datenwebserviceklasse. Sie enthält die Loginfunktionalität, die von allen speziellen Teilen gleichermaßen benötigt wird. Die dazu notwendigen Loginklassen sind ebenfalls Bestandteil der allgemeinen Module.

6.2.3.2 Spezielle Bestandteile

Grundsätzlich besteht jede Einschreibekategorie aus verschiedenen Webforms. Neben den einzelnen Webformklassen gibt es für jede Kategorie einen eigenen Webservice, der von den Webforms als Datenquelle genutzt wird. All diese Webservices sind vom allgemeinen Datenwebservice abgeleitet.

[23] gilt nur bedingt, mittels einer eigenen ControlDesigner Implementierung wäre auch ein beliebig korrektes Darstellen möglich

6.2.4 Windowsanwendung

Wie einführend erwähnt wurde, wurde neben der Internetanwendung eine in der Funktionalität vereinfachte Windowsanwendung implementiert. Die Windowsanwendung enthält passende Proxyklassen zu den verschiedenen Datenwebservices der Einschreibekategorien. Mit Hilfe dieser Proxyklassen kann die Anwendung alle notwendigen Informationen auslesen und darstellen. Die Darstellung wird von traditionellen Windowscontrols übernommen. In **Abb. 6-5** ist die Windowsanwendung dargestellt.

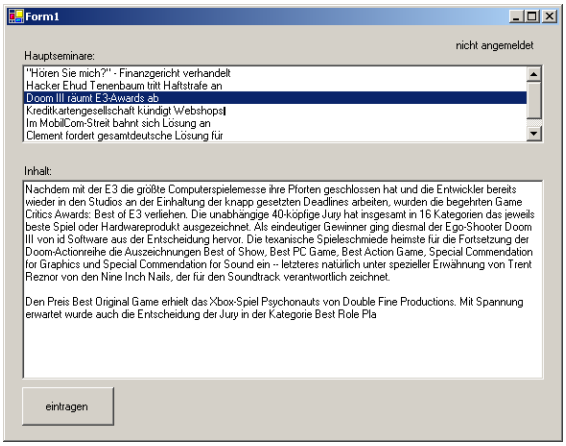

Abb. 6-5: Windowsanwendung, realisiert mit C#

7 Vergleich

In diesem Kapitel werden die beiden Implementierungen gegenübergestellt und verglichen. Aufgrund der sehr ähnlichen Architekturen ist ein Vergleich relativ gut möglich. Grundlage des Vergleichs sind die jeweiligen Vor- bzw. Nachteile oder andere Eigenheiten der Technologien.

Das problematische an Vergleichen ist immer, dass sie zum einen nie hundertprozentig umfassend sind und das teilweise nur ein subjektiver Eindruck vermittelt werden kann. Dieser Vergleich ist davon leider nicht ausgenommen.

7.1 Grundlegende Merkmale

Zweifelsfrei einer der größten Vorteile von Java ist die Plattformunabhängigkeit. Es stellt sich allerdings die Frage, wieviele serverseitige Anwendungen in ihrer Lebenszeit diese Eigenschaft benötigen, wie oft also die Serverplattform derart dramatisch geändert wird. Anders ausgedrückt: bei clientseitigen Javalösungen spielt dieses Argument eine essentielle Rolle, bei serverseitigen Anwendungen ist ein Wechsel des Betriebssystems hingegen unwahrscheinlich. Das genaue Gegenteil ist (noch[24]) die .NET Plattform: .NET Anwendungen sind zwangsläufig an ein windowsbasiertes Betriebssystem gebunden, was ebenfalls eindeutig gegen.NET sprechen würde, falls eine nicht-Windows basierte Umgebung benutzt werden muss. Es stellt sich also die Frage, was Plattformunabhängigkeit für eine Rolle spielt, falls es eine große Rolle spielt führt kein Weg an Java vorbei.

Der Umweg über Zwischensprachen, den sowohl Java[25] als auch .NET[26] gehen bringt zwar (potentielle) Plattformunabhängigkeit, aber auch Ineffizienz mit sich. Diese Ineffizienz äußert sich spätestens zur Laufzeit beim Ressourcenverbrauch. Da beiden Sprachen dasselbe Prinzip zugrunde liegt, ist es umso verwunderlicher, dass .NET Anwendungen generell ressourcensparender ausfallen als ihre Java - Pendants.

[24] das Mono Projekt ([MONO]) portiert die .NET Plattform auf andere Betriebssysteme

[25] Java compiliert zu .class Dateien, die von Java Virtual Machines interpretiert werden

[26] Vgl. Kap. 6.1.3

Ein anderer Unterschied betrifft die Offenheit beider Systeme. Beispielsweise existieren auf Javaseite dutzende völlig unterschiedliche Webpublishingframeworks[27] oder Komponentenmodelle[28]. Eine derartige Systemvielfalt ermöglicht eine abgestimmtere Umgebung für eigene Projekte. Das komplette Gegenteil stellt wiederum .NET dar: es gibt genau ein Publishingframework und genau ein Komponentenmodell. Positiv bleibt zu bemerken, dass Microsoft nicht vergebens bemüht ist, seine Systeme so offen und vielseitig wie möglich zu realisieren, was sich allerdings sehr negativ auf deren Komplexität auswirkt. Damit stellt sich folgende Frage: hohe Integration (woraus hohe Effizienz folgt) oder hohe Offenheit (woraus hohe Vielfältigkeit folgt).

Ausnahmsweise keinen Unterschied gibt es in Bezug auf den Funktionsumfang sowie die Werkzeugunterstützung. Sowohl .NET als auch Java stellen unzählige Werkzeuge und komplexe Entwicklungsumgebungen zur Lösung von Standartaufgaben zur Verfügung. Auch in der Benutzung der jeweiligen Klassen gibt es keine auffälligen Unterschiede.

Ein ebenfalls sehr wichtiger Unterschied stellt die Kostenfrage dar. Einerseits sind beide Plattformen kostenlos verfügbar, andererseits bringt .NET höhere Kosten für Betriebssysteme oder andere notwendige Lizenzen mit sich. Des Weiteren existieren für Java erstaunlich umfangreiche Entwicklungsumgebungen[29], die ein ähnliches Integrationsniveau wie Microsofts (sehr kostenintensive) .NET Entwicklungsumgebung Visual Studio.NET bieten, aber vollkommen kostenlos und frei verfügbar sind.

Der letzte Unterschied auf den eingegangen werden soll betrifft den Reifegrad. Die Javaplattform ist zweifellos wesentlich älter und hatte somit mehr Zeit zu ‚reifen'. Das äußert sich zum einen darin, dass es in Bezug auf die Benutzung einen relativ geschlossenen Eindruck macht, und zum anderen existieren genau die Werkzeuge, die gebraucht werden. Aussagen über Fehlerfreiheit sind allerdings gefährlich, obwohl es nahe liegend ist, dass reife Software grundsätzlich fehlerfreier ist.

[27] bspw. Cocoon, Turbine

[28] bspw. EJB verschiedener Anbieter (jBoss, Enhydra, Weblogic, Websphere)

[29] bspw. Netbeans

7.2 Implementierungsmerkmale

Aufgrund des objektorientierten Zugriffsmodells von ASP.NET Seiten gestaltet sich die Programmierung von dynamischen Verhalten relativ einfach und intuitiv. Dynamisches Verhalten mit javabasierten Mitteln zu realisieren ist etwas aufwendiger, aber genauso einfach realisierbar. Dieser Unterschied äußerte sich bei der Implementierung der Einschreibeplattform hauptsächlich im Zeitbedarf, der für die jeweilige Umsetzung verbraucht wurde. Da sich wie bereits erwähnt, die Klassenbibliotheken im Umfang nicht wesentlich unterscheiden, mussten weder für die .NET Lösung noch für die Java Lösung Standartklassen selbst programmiert werden. Auch der Umgang mit Datenbankbasierten Informationen gestaltet sich bei beiden Formen sowohl im Umfang als auch in der Umsetzung identisch.

7.3 Schlussfolgerung

Sowohl Java als auch .NET eignen sich gut für die Realisation von internetbasierten Anwendungen. Beide Plattformen haben sowohl Vor- als auch Nachteile. Daher ist es abgesehen von Plattform-, Betriebssystem- oder Kosteneinschränkungen eher eine Frage des persönlichen Geschmackes, welche Umsetzungsform gewählt wird.

8 Installation und Konfiguration

Dieser Abschnitt behandelt die Installation und Konfiguration der Einschreibeplattform auf Java und .NET Umgebung.

8.1 Datenbankinstallation

Es können sowohl Microsoft SQL 2000 Server als auch PostgreSQL als Datenbankmanagementsystem genutzt werden. Prinzipiell kann auch jede andere Datenbank benutzt werden, da für die Javaanwendung JDBC und für .NET ODBC als Datenbankabstraktionsschnittstellen genutzt wurde. Exemplarisch wird die Installation auf PostgreSQL und MSSQL2000 beschrieben.

8.1.1 PostgreSQL

Zuerst muss eine Datenbank erstellt werden. Folgende Anweisung eignet sich dafür am besten, da auch ein sonderzeichenkompatibler Zeichensatz eingestellt wird:

```
psql -d template1 -c "CREATE DATABASE "einschreiber" WITH TEMPLATE = template0 ENCODING = 'LATIN1'"
```

Nach dem die Datenbank erstellt wurde, kann das PSQL Createscript ausgeführt werden:

```
psql -d einschreiber -f dump
```

Auf diese Weise wird eine funktionsfähige Einschreiber Datenbank erzeugt.

8.1.2 Microsoft SQL 2000

Nach dem Erstellen einer leeren Datenbank können die notwendigen Tabellen mittels des Create-Skriptes *mssql_create_script.sql* erzeugt werden:

```
isql mssql_create_script.sql
```

Das Ergebnis ist eine funktionsfähige Datenbankumgebung.

8.2 Java Umgebung

Folgende Software mindestens benötigt:

- Java2 Platform Standard Edition: http://java.sun.com/j2se/
- Tomcat4: http://jakarta.apache.org/tomcat/

- für die schmerzfreiestes übersetzen der sourcen und deployen der Binaries eignet sich Netbeans ([NB]) am besten

JDBC Treiber für die zu benutzende Datenbank:

- PostgreSQL JDBC: http://jdbc.postgresql.org/
- MSSQL2000: http://msdn.microsoft.com/downloads/default.asp?URL=/downloads/sample.asp?url=/MSDN-FILES/027/001/779/msdncompositedoc.xml

8.2.1 Übersetzung

Die Binaries sind am einfach mittels des zugehörigen Netbeans ([NB]) Projektes erzeugbar. Alternativ dazu können die Javasourcen auch manuell zu Javabytecode übersetzt werden. Die so erzeugten Javaklassen müssen für die Benutzung als Tomcat Internetanwendung zusammen mit den zugehörigen JSP Dateien zu einem Webarchiv gepackt werden. Dafür kann entweder Netbeans oder ein anderer Jar Packager genutzt werden.

8.3 .NET Umgebung

Folgende Software wird benötigt:

- .NET Framework SDK: http://msdn.microsoft.com/netframework/productinfo/next/download.asp
- Internet Information Server 5 oder anderer ASP.NET kompatibler Webserver

8.3.1 Übersetzung

Übersetzt werden können die gesamten Quellen mittels des Visual Studio 7 Projektfiles, oder per Kommandozeilen Compiler CSC.EXE. Anschließend muss noch ein Link auf das Binary Verzeichnis im Webserver eingerichtet werden.

9 Links

[SERVLET] Java Servlets
http://www.ssw.uni-linz.ac.at/Teaching/Lectures/Sem/2000/Kotzmann/index.html

[ASP] The Official Microsoft ASP.NET Site
http://www.asp.net/

[JSP2] Understanding JavaServer Pages Model 2 architecture
http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

[NET] Microsoft .NET
http://www.microsoft.com/net/

[J2EE] Java 2 Platform, Enterprise Edition
http://java.sun.com/j2ee/

[SOAP] Simple Object Access Protocol (SOAP) 1.1
http://www.w3.org/TR/SOAP/

[UDDI] Universal Description, Discovery and Integration
http://www.uddi.org/

[GOOGLE] GoogleAPI
http://api.google.com/search/beta2

[MONO] Project MONO
http://www.go-mono.org

[NB] NetBeans
http://www.netbeans.org