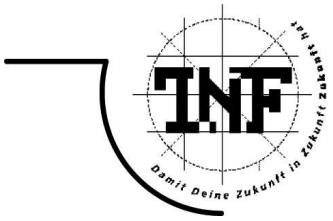
JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis


# Anforderungsanalyse, Entwurf und Realisierung eines AJAX-Web-Systems für dialektische Ausdrücke

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

in der Studienrichtung

INFORMATIK

Eingereicht von:
*Simon Kohlberger Bakk.techn., 0155806*

Angefertigt am:
*Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM)*

Betreuung:
*o. Univ.-Prof. Dr. Jörg R. Mühlbacher*

Mitbetreuung:
*Dipl.-Ing. Andreas Putzinger*

*Linz, Oktober 2007*

# Kurzfassung

Diese Magisterarbeit beschäftigt sich grundsätzlich mit einem System zur webbasierten Verwaltung von Wörtern und Übersetzungen/Bedeutungen in beliebigen Sprachen.

Das Ziel des ersten Teils dieser Arbeit besteht darin, die Anforderungen an das System zu spezifizieren. Dies umfasst einerseits eine Anforderungsanalyse für das Produkt selbst, wie auch eine Evaluierung bestehender Wörterbuch-Systeme sowie möglicher Komponenten. Außerdem werden ausgewählte Java-Webframeworks untersucht. Da im Endprodukt auch eine phonetische Suche zur Verfügung steht, wird im ersten Teil auf die verschiedenen Techniken, sowohl zur Suche als auch zur Reihung der Ergebnisse eingegangen.

Der zweite Teil der Magisterarbeit beschäftigt sich mit der Implementierung des Systems. Es werden sowohl die Architektur des Systems, als auch die Systemanforderungen geklärt. Des weiteren werden die Möglichkeiten des Systems vorgestellt. Features des Systems sind beispielsweise, dass Wörter direkt im Webbrowser vertont werden können oder dass die Datenbasis in ein PDF-Dokument exportiert werden und somit jederzeit ein aktuelles „Wörterbuch“ als Hardcopy erstellt werden kann.

# Abstract

This Master's thesis generally deals with a web-based management system for words, their translations and meanings.

The first part of this work elaborates on requirements for the management system. This includes a requirements analysis as well as an evaluation of already existing online dictionary systems and Java components. In addition selected Java web frameworks are investigated. As the final product supports phonetic searching, various search techniques and algorithms for ordering the results are discussed.

The second part of the thesis focuses on the implementation of the system itself. This covers the architecture as well as the requirements of the system. Furthermore the main features of the system are presented, like the possibility to directly record words through the web browser or to export the whole data into a PDF-document which can be used to print an up-to-date hard copy of the dictionary.

# Danksagung

An dieser Stelle möchte ich einigen Personen besonders danken, die mich während meines Studiums und bei der Anfertigung dieser Magisterarbeit unterstützt haben.

In erster Linie bedanke ich mich bei meiner Familie, ohne deren Hilfe und Unterstützung dieses Studium nicht möglich gewesen wäre. Des Weiteren bedanke ich mich bei meiner Freundin Astrid Waldhör für ihre Unterstützung, insbesondere für das Korrekturlesen dieser Arbeit.

Außerdem danke ich den Mitarbeitern des Instituts für Informationsverarbeitung und Mikroprozessortechnik (FIM). Allen voran o. Univ.-Prof. Dr. Jörg R. Mühlbacher für die Betreuung dieser Arbeit und die mir gebotene Möglichkeit eines Auslandssemesters in Reading (GB). Darüber hinaus bedanke ich mich bei Dipl.-Ing. Andreas Putzinger für die wertvollen Tipps, die wesentlich zum Gelingen dieser Arbeit beigetragen haben.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Listings

# Kapitel 1

# Einleitung

## 1.1 Ziel der Magisterarbeit

Diese Magisterarbeit beschäftigt sich mit einer Analyse der Anforderungen an ein webbasiertes System zur Verwaltung von Wörtern und deren Übersetzungen in beliebigen Sprachen, sowie mit der Implementierung eines solchen Systems. Daraus folgt sowohl eine umfassende Anforderungsanalyse an das System, als auch eine Evaluierung und Auswahl möglicher Komponenten. Da die Implementierung vorwiegend als Online-Wörterbuch für die Sprachen Oberösterreichisch-Deutsch-Englisch eingesetzt wird, muss bei der Entwicklung speziell auf den oberösterreichischen Dialekt Rücksicht genommen werden.

## 1.2 Motivation

Die Idee zu diesem Projekt ist am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM) entstanden. Ursprünglich gab es nur eine statische HTML-Seite (siehe Abbildung 1.1) die eine Übersetzung zwischen Oberösterreichisch, Deutsch und Englisch anbietet. Da die Seite stetig wuchs und somit die Wartung und das Suchen erschwert wurden, war es nur ein Frage der Zeit, bis eine Web-Anwendung mit einer umfassenden Management-Umgebung und verbesserter Suchmöglichkeit notwendig wurde.

Die eigentliche Motivation für diese Arbeit ist es, speziell internationalen Studenten eine Hilfestellung beim Erlernen des oberösterreichischen Dialekts und der deutschen

**(Ober)österreichisch-Deutsch-Englisches Wörterbuch**

Zur besseren Verständigung zwischen (ober)österreichischen Studenten und Erasmus-Studenten

Wenn Sie Ideen und Vorschläge zur Erweiterung des Wörterbuches haben, schicken Sie bitte eine E-Mail an: **wundawuzi@fim.uni-linz.ac.at**.

A B D E F G H I J K L M N O P Q R S T U V W Z

Wir haften nicht für einen unangemessenen Gebrauch der angeführten Wörter!!

Legende:

- Grundsätzlich **NICHT verwenden**!!

L ländlicher Raum

S städtischer Raum

W wienerisch/städtisch

Verwendung: im positiven Sinn: J
im negativen Sinn: L
neutral: K
kontextabhängig: k

| | | (Ober)österreichisch | Deutsch | Englisch | | |
|---|---|---|---|---|---|---|
| **A** | M | | | | | |
| | | a... | Aussprache ähnlich "oir" (fr) oder in "more" oder "was" (engl.), bzw. "au" oder a (de) | pronounciation like "oir" (fr.) or in "more" or "was" (engl.), or like "au" or "a" (german) | | |
| | | a | auch | also, too | K | |
| | | a, an, aner, ane, ans | ein, einen, einer, eine, eines; hängt vom Casus/Fall UND vom grammatikalischen Geschlecht (männl./weibl./sächl.) ab; wird nur im | a, an , one (form depends on case and gender) The "a" is pronounced as in Southern English "but" | K | |

**Abbildung 1.1:** Oberösterreichisch-Deutsch-Englisches Wörterbuch [N.N07d]

Sprache zu bieten. Da unter den Studenten an der JKU hauptsächlich im oberösterreichischen Dialekt gesprochen wird, sind wahrscheinlich auch für internationale Studenten mit guten Deutsch-Kenntnissen Wörter dabei, die sie nicht kennen. Über dieses Online-Wörterbuch können Studenten nun unbekannte Wörter nachschlagen. Somit soll eine einfachere und raschere Integration der internationalen Studenten an der JKU ermöglicht werden.

## 1.3 Struktur der Magisterarbeit

Dies Arbeit ist in drei Hauptkapitel unterteilt. Kapitel 2 beschäftigt sich primär mit den Anforderungen an das System. Es werden die Akteure im System identifiziert und die wichtigsten Anforderungen erklärt und mit Anwendungsfalldiagrammen (engl. use case diagram) dargestellt. Außerdem werden ausgewählte bestehende Online-Wörterbücher evaluiert.

In Kapitel 3 werden ausgewählte Java-Komponenten und Algorithmen vorgestellt, die für die Implementierung in Betracht gezogen wurden. Zuerst werden die Basis-Web-Technologien für Java behandelt. Des weiteren werden drei Webframeworks vorgestellt, die für die Implementierung in die engere Auswahl gekommen sind. Neben Algorithmen für eine phonetische Suche werden auch welche zum Sortieren der Suchergebnisse erklärt. Da Hibernate als Persistenzschicht und die Bibliothek iText für die Erzeugung von PDF-Dokumenten eingesetzt wurde, werden diese detailliert beschrieben. Abschließend werden weitere verwendete Komponenten kurz vorgestellt.

Kapitel 4 beschäftigt sich mit der Implementierung des Systems. Zu Beginn werden die Systemanforderungen geklärt. Des weiteren wird auf die Architektur des Systems eingegangen. Dazu werden u. a. die wichtigsten Klassen und deren Funktionen kurz beschrieben. In einem weiteren Abschnitt werden die Implementierungen ausgewählter Anwendungsfälle vorgestellt.

# Kapitel 2

# Funktionale Anforderungen an das System

Dieses Kapitel beschäftigt sich mit den Anforderungen an das System. Der Implementierung ist eine umfassende Anforderungsanalyse vorangegangen. Diese bestand sowohl aus Gesprächen mit Mitarbeitern des Instituts, welche die zukünftigen Betreiber des Systems sind, als auch einer Evaluierung ausgewählter, bestehender Online-Wörterbücher. Das Ergebnis der Evaluierung ist im letzten Abschnitt dieses Kapitels zu finden. In den folgenden Abschnitten wird nun auf die verschiedenen Akteure des Systems und die Anforderungen eingegangen.

## 2.1 Grundlegende Anforderungen

Dieser Abschnitt beschäftigt sich mit den grundlegenden Anforderungen an das System. Die Anforderungsanalyse ergab folgende wichtige Punkte, die vom System erfüllt werden müssen:

**Oberösterreichischer Dialekt:** Das Online-Wörterbuch soll speziell im Hinblick auf den oberösterreichischen Dialekt entwickelt werden.

**Phonetische Suche:** Da die Schreibweise im Dialekt meist nicht eindeutig ist, muss das System auch ähnlich ausgesprochene Wörter als Suchergebnis liefern. Ähnliche Ergebnisse müssen aber auch als solche gekennzeichnet werden. Die phonetische Suche soll für möglichst viele Sprachen zur Verfügung stehen.

**Auto-Vervollständigung:** Während der Eingabe des Suchbegriffs soll der Benutzer durch Vorschläge unterstützt werden. Es sollen sowohl exakt übereinstimmende Wörter, als auch phonetisch ähnliche Wörter vorgeschlagen werden.

**Auditive Wiedergabe:** Vertonte Wörter und Beispielsätze sollen vom System abgespielt werden können. Dies soll dem Benutzer die richtige Aussprache vermitteln.

**Anmerkungen:** Wörter müssen mit beliebigen Anmerkungen versehen werden können. Beispielsweise, dass ein Wort normalerweise im negativen oder positiven Sinne verwendet wird. Diese Anmerkungen müssen in den Suchergebnissen angezeigt werden. Die Anmerkungen sollen im System zentral gespeichert und vom Administrator verwaltet werden können.

**Weiterführende Links:** Um den Benutzer bei der weiterführenden Suche zu unterstützen, sollen Links zu externen Webseiten angeboten werden. Durch Betätigen eines Links soll, wenn möglich, eine Suche auf der externen Seite durchgeführt werden. Die angebotenen Links müssen sich natürlich in Abhängigkeit von der gerade ausgewählten Sprache unterscheiden. Bei einem englischen Wort müssen also andere Links angeboten werden, als bei einem Wort im oberösterreichischen Dialekt.

**Lokalisierung:** Die Benutzerschnittstelle des Online-Wörterbuchs soll in deutscher und englischer Sprache verfügbar sein.

## 2.2 Akteure und das Rollenkonzept

Im Zuge der Anforderungsanalyse wurden im System folgende vier Akteure identifiziert, die jeweils unterschiedliche Rollen einnehmen:

**Benutzer:** Der nicht-angemeldete Benutzer kann nur die Basisfunktionen des Systems verwenden. Dazu zählt u. a. die Suche im Online-Wörterbuch.

**Angemeldeter Benutzer:** Dem angemeldeten Benutzer steht neben der Suche auch eine persönliche Vokabelliste zur Verfügung.

**Sprecher:** Der Sprecher ist für das Vertonen von Wörtern zuständig. Ihm stehen zusätzliche Funktionen zur Verfügung um Wörter aufnehmen zu können.

**Administrator:** Der Administrator ist für die Verwaltung des Systems verantwortlich. Ihm steht der gesamte Funktionsumfang des Systems zur Verfügung. Der Administrator kann beispielsweise den Sprechern Wörter zum Vertonen zuteilen.

Die eben genannten Rollen bauen aufeinander auf, d. h. ein Administrator hat automatisch auch die Rolle eines Sprechers, ein Sprecher hat auch die Rolle eines angemeldeten Benutzers und ein angemeldeter Benutzer nimmt auch die Rolle eines nichtangemeldeten Benutzers ein. Mit den verschiedenen Rollen und ihren unterschiedlichen Aufgaben geht auch die Forderung nach einer Benutzer-Authentifizierung einher. Das System muss also einen Benutzer durch Eingabe von Benutzername und Passwort authentifizieren können. Außerdem muss sich ein Benutzer registrieren können, um die Rolle eines angemeldeten Benutzers einnehmen zu können.

## 2.3 Detaillierte Darstellung ausgewählter Anforderungen

Dieser Abschnitt behandelt ausgewählte Anforderungen. Sie wurden so gewählt, dass sie den wichtigsten Funktionsumfang des Systems abdecken. Neben der textuellen Beschreibung werden auch Anwendungsfalldiagramme zur Darstellung herangezogen.

### 2.3.1 Nach Wörtern suchen

Nach Wörtern zu suchen ist der zentrale Anwendungsfall im Online-Wörterbuch. Wie in Abbildung 2.1 zu erkennen ist, wird zwischen zwei Rollen unterschieden: dem nicht-angemeldeten Benutzer und dem angemeldeten Benutzer. Der Unterschied zwischen den beiden Rollen ist der, dass sich der angemeldete Benutzer zusätzlich alle Wörter des Wörterbuchs anzeigen lassen kann. Ein nicht-angemeldete Benutzer hingegen kann sich lediglich alle Wörter eines Anfangsbuchstaben anzeigen lassen.

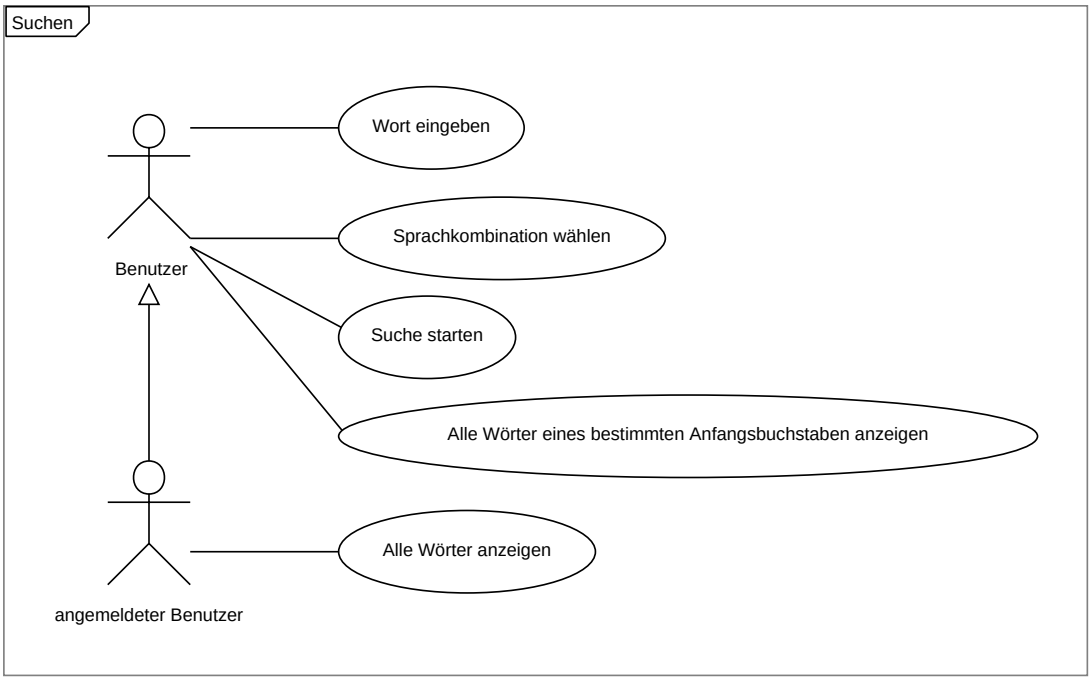



**Abbildung 2.1:** Wörter suchen

### 2.3.2 Suchergebnisse anzeigen

Die Anzeige der Suchergebnisse soll in zwei Kategorien unterteilt werden: eine Ergebnistabelle für exakte Übereinstimmungen und eine Ergebnistabelle für phonetische Übereinstimmungen. Bei mehr als acht Ergebnissen soll die Ergebnistabelle in mehrere

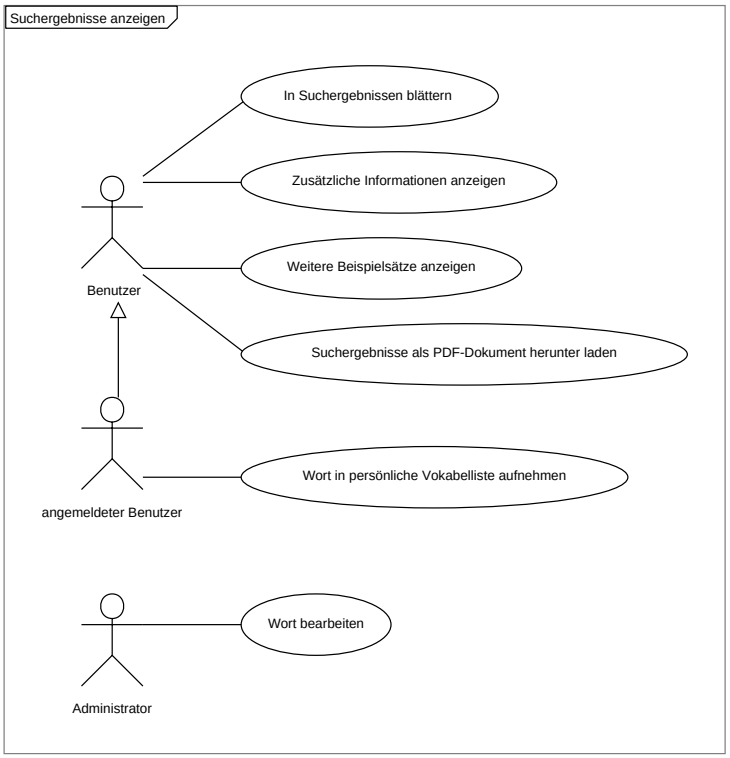



**Abbildung 2.2:** Suchergebnisse anzeigen

Seiten unterteilt werden. Wie im Anwendungsfall (Abbildung 2.2) eingezeichnet ist, soll der Benutzer durch die Seiten blättern können.

Über einen Link soll es möglich sein, zusätzliche Informationen zu einem Wort abzurufen, z. B. um zu sehen, mit welchen Anmerkungen das Wort versehen wurde und welche weiterführenden Links auf externe Seiten vorhanden sind.

Grundsätzlich soll maximal ein Beispielsatz pro Wort angezeigt werden. Dies soll ein Überfüllen der Ergebnistabelle verhindern. Falls ein Benutzer mehrere Beispiele sehen möchte, dann müssen diese über einen Link extra angefordert werden.

Damit ein Benutzer die Ergebnistabelle auch offline nutzen und ausdrucken kann, soll die Tabelle als PDF-Dokument gespeichert werden können.

Dieser Anwendungsfall bietet je nach Rolle verschiedene Möglichkeiten. Ein angemeldeter Benutzer soll zusätzlich Wörter in eine persönliche Vokabelliste aufnehmen können. Außerdem soll ein Administrator die Möglichkeit haben, einen Eintrag im Online-Wörterbuch zu bearbeiten.

### 2.3.3 Wort vorschlagen

Dieser Anwendungsfall (siehe Abbildung 2.3) bildet den Vorgang ab, wenn ein Benutzer ein neues Wort für das Online-Wörterbuch vorschlägt. Damit dem Benutzer geantwortet werden kann, muss er seine Email-Adresse bekannt geben. Vom System soll die E-Mail-Adresse auf ihre Gültigkeit überprüft werden. Nachdem der Benutzer seine Nachricht bzw. das Wort eingegeben hat, soll ein Captcha[1] feststellen, dass die Eingabe von einem Menschen und nicht von einer Maschine (Bot) stammt. Nachdem alle Daten eingegeben wurden, soll der Benutzer die Daten abschicken können.

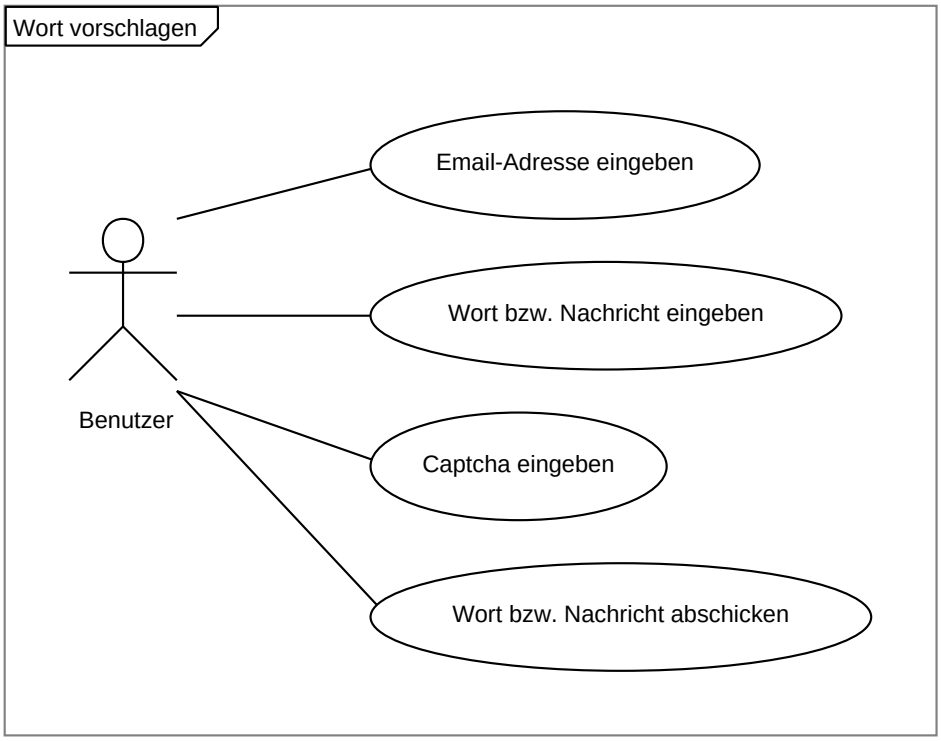



**Abbildung 2.3:** Wort vorschlagen

### 2.3.4 Wörter hinzufügen und ändern

Dieser Anwendungsfall steht nur einem Administrator zur Verfügung. Wie in Abbildung 2.4 zu sehen ist, soll ein Wort eingegeben, die dazugehörige Sprache gewählt und

[1] Akronym für Completely Automated Public Turing test to tell Computers and Humans Apart.

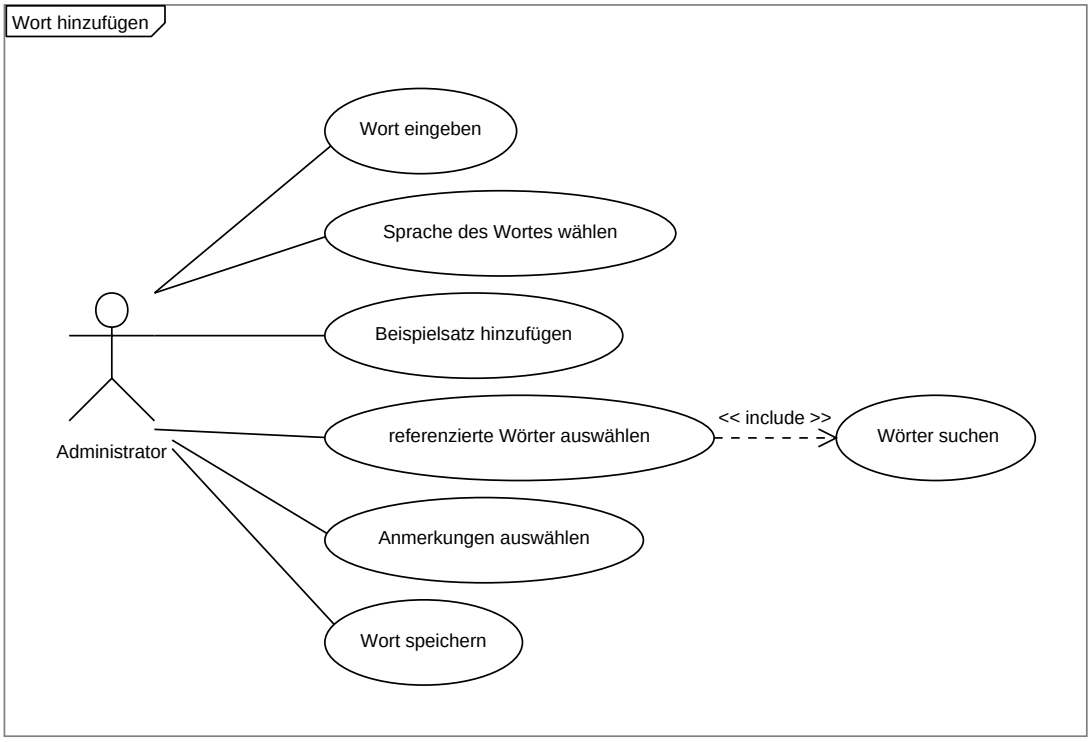



**Abbildung 2.4:** Wörter hinzufügen

beliebig viele Beispielsätze hinzugefügt werden können.

Außerdem soll der Administrator bei der Suche und Auswahl der referenzierten Wörter unterstützt werden. Die Suche soll daher auch phonetisch ähnliche Wörter finden können. Zusätzlich zu den gefundenen Wörtern sollen deren referenzierte Wörter angezeigt werden. Wenn möglich sollen diese hervorgehoben werden. Im Idealfall muss dadurch nur nach einem Wort gesucht werden, da alle bereits bestehenden Referenzen mit angezeigt werden. Der Administrator soll dann mehrere Wörter auf einmal auswählen können.

Des weiteren sollen beliebige Anmerkungen ausgewählt werden können. Wenn alle Daten eingegeben wurden, soll der gesamte Datensatz gespeichert werden können.

Der Anwendungsfall *Wörter ändern* läuft im Grunde ähnlich ab und wird daher nicht weiter behandelt. Anstatt einen Datensatz hinzuzufügen, wird er lediglich verändert.

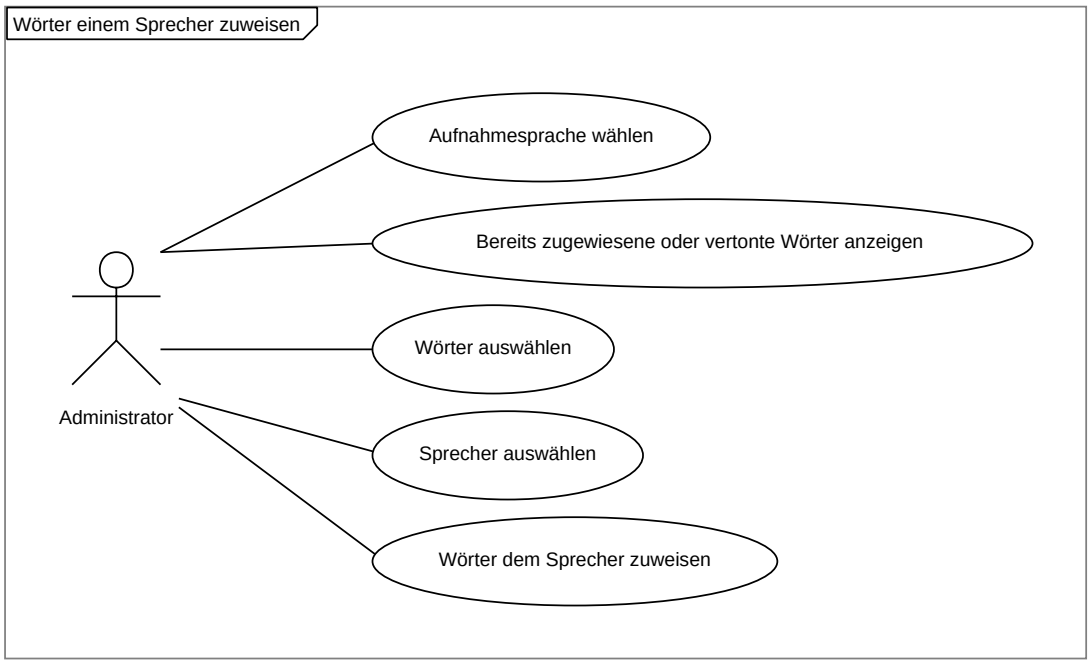



**Abbildung 2.5:** Wörter einem Sprecher zuordnen

### 2.3.5 Wörter einem Sprecher zuordnen

Das Online-Wörterbuch soll es einem Administrator ermöglichen, Wörter auszuwählen und diese dann einem Sprecher zuzuweisen. In Abbildung 2.5 ist dieser Anwendungsfall dargestellt.

Zuerst soll ein Administrator die Aufnahmesprache festlegen können. Dadurch sollen dann nur mehr Wörter der ausgewählten Sprache angezeigt werden, die noch nicht vertont wurden. Auf Wunsch sollen dennoch bereits vertonte Wörter angezeigt werden, um diese gegebenenfalls erneut vertonen zu können.

Nachdem eine Liste mit Wörtern ausgewählt wurde, soll ein Sprecher gewählt werden können. Dabei sollen nur jene Sprecher aufgelistet werden, die auch die vorher gewählte Sprache sprechen.

Zum Schluss soll der Administrator die Auswahl speichern könnnen.

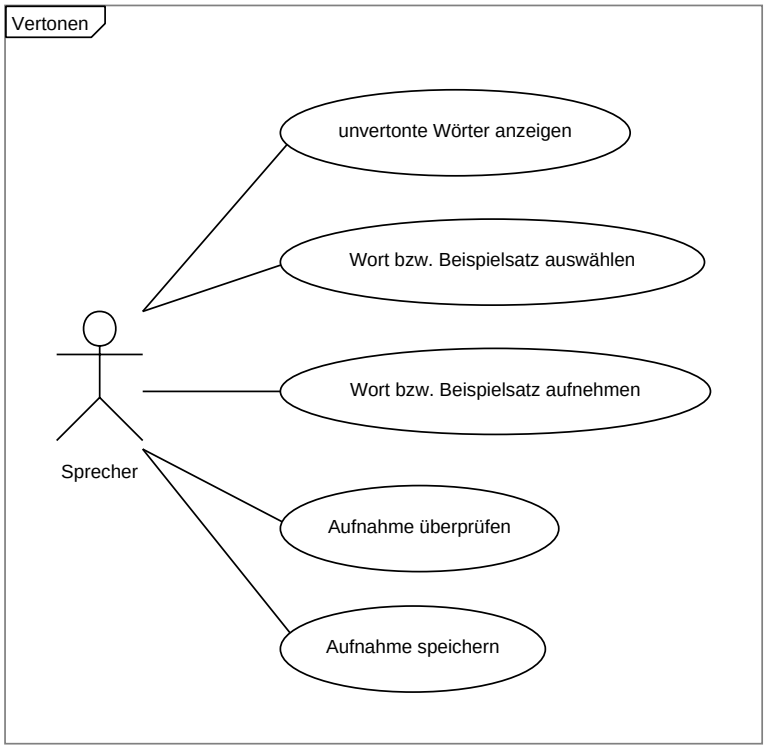



**Abbildung 2.6:** Wörter vertonen

### 2.3.6 Wörter vertonen

Dieser Anwendungsfall zeigt, wie ein Sprecher ein Wort bzw. einen Beispielsatz vertonen kann. Wie man in Abbildung 2.6 sehen kann, soll der Sprecher eine Liste mit den für ihn zugeteilten Wörtern angezeigt bekommen. Nachdem ein Wort ausgewählt wurde, soll die Aufnahme starten. Des weiteren soll der Sprecher die Möglichkeit haben, das aufgenommene Wort anhören zu können. Damit soll gewährleistet werden, dass nur qualitativ hochwertige Aufnahmen im Online-Wörterbuch verwendet werden. Zum Schluss soll die Aufnahme permanent gespeichert werden können.

### 2.3.7 Importieren von CSV-Dateien

Um das Online-Wörterbuch rasch mit den bereits existierenden Daten füllen zu können, soll es möglich sein, CSV-Dateien hochzuladen. Neben dem eigentlichen Wort und dessen Sprache soll die CSV-Datei auch beliebig viele Beispielsätze enthalten können.

Außerdem sollen auch Anmerkungen mit importiert werden können, welche nur ein Administrator durchführen kann.

### 2.3.8 Benutzer verwalten

Im Online-Wörterbuch sollen sich Benutzer selbst registrieren können. Um die entstehende Benutzerliste zu warten, bedarf es einer Verwaltung. Wie in Abbildung 2.7 zu sehen ist, soll ein Administrator daher die Möglichkeit haben, wichtige Daten wie z. B. Benutzername, Passwort, Sprache und E-Mail-Adresse zu ändern.

Des weiteren soll er die Rolle der einzelnen Benutzer ändern und so jemandem die Rolle eines Sprechers oder Administrators zuteilen können.

Für Benutzerkonten, die nicht mehr benötigt werden, soll es auch eine Möglichkeit geben, diese zu entfernen.

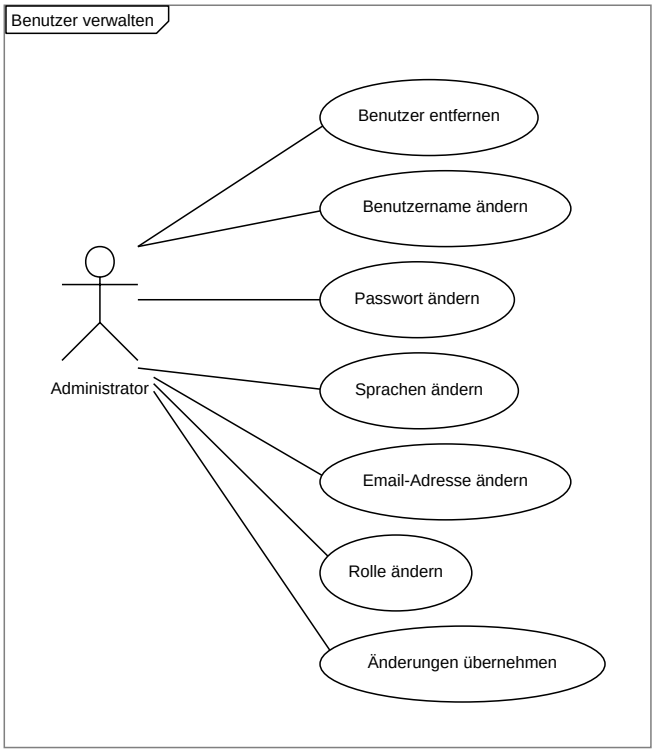



**Abbildung 2.7:** Benutzer verwalten

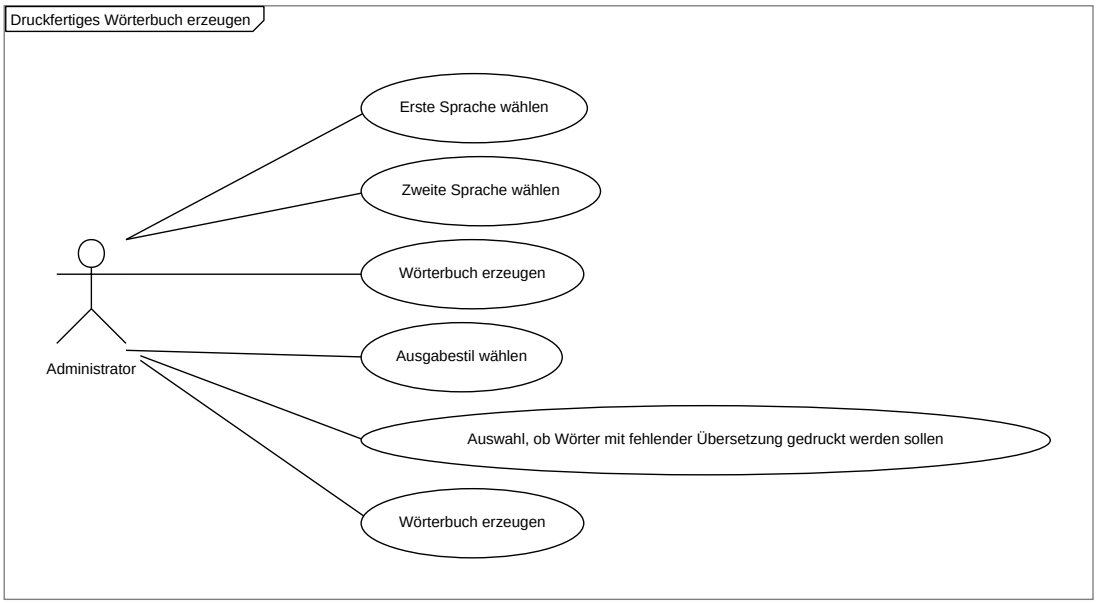



**Abbildung 2.8:** Druckfertiges Wörterbuch erzeugen

### 2.3.9 Druckfertiges Wörterbuch erzeugen

Ein druckfertiges Wörterbuch zu erzeugen ist ein primärer Anwendungsfall (siehe Abbildung 2.8) eines Administrators. Das System soll es daher ermöglichen, aus einer beliebigen Sprachkombination ein Wörterbuch im PDF-Format zu erzeugen. Das PDF-Dokument soll dann direkt im Web-Browser angezeigt und von dort gespeichert werden können. Bevor ein PDF-Dokument erzeugt wird, soll ein Ausgabestil ausgewählt werden können, z. B. um ein PDF-Dokument in verschiedenen Seitenformaten und Schriften erstellen zu können.

Des weiteren soll ausgewählt werden können, ob Wörter ohne Übersetzung in das PDF-Dokument übernommen werden sollen.

## 2.4 Evaluierung ausgewählter Online-Wörterbücher

Dieser Abschnitt beschäftigt sich mit bereits bestehenden Online-Wörterbüchern. Zum Einen, um einen Einblick in diese Wörterbücher und deren Funktionsumfang zu erhalten und zum Anderen, um die Unterschiede bzw. Gemeinsamkeiten zum implementierten Online-Wörterbuch aufzuzeigen.

Hauptaugenmerk wird dabei nicht auf die grafische Präsentation, sondern auf den Funktionsumfang und die einfache Verwendung der Webseite gelegt. Außerdem erfolgt die Evaluierung nicht nach einem strengen, formalen Kriterienkatalog und hat daher eher einen informellen Charakter.

Die im Folgenden beschriebenen Online-Wörterbücher wurden nicht zufällig gewählt. LEO wurde gewählt, weil es eines der meistgenutzten Online-Wörterbücher ist. Dict.cc hingegen wird nicht ganz so häufig verwendet, hat dafür aber einen größeren Wortschatz. Das Online-Wörterbuch Ostarrichi wurde ausgewählt, weil es auf die Übersetzung zwischen österreichischem Dialekt und Deutsch spezialisiert ist. Das letzte Wörterbuch, BeoLingus, wurde ausgewählt, weil es vom technischen Standpunkt her eines der ausgereiftesten Online-Wörterbücher darstellt. Die Auswahl bildet damit einen Querschnitt über die meistgenutzten Online-Wörterbücher und den derzeitigen Stand der Technik.

### 2.4.1 LEO

Das LEO-Online-Wörterbuch ist wohl eines der bekanntesten Wörterbücher im deutschsprachigen Raum und wird von der LEO GmbH mit Sitz in Sauerlach, Deutschland, angeboten.

Leo.org [Rie07] bietet Wörterbücher für die Sprachen Deutsch-Englisch, Deutsch-Französisch und Deutsch-Spanisch an. Das Deutsch-Englisch-Wörterbuch ist derzeit das umfangreichste und enthält ca. 450.000 Wörter und Redewendungen. Außerdem enthält die Webseite eine große Link-Sammlung zu den verschiedensten Themen, Informationen zu München und eine Sammlung von Cocktail-Rezepten. Zum weiteren Angebot zählen auch Diskussionsforen und ein Vokabeltrainer. An Wochentagen werden zwischen sechs- und neun Millionen Anfragen pro Tag verbucht.

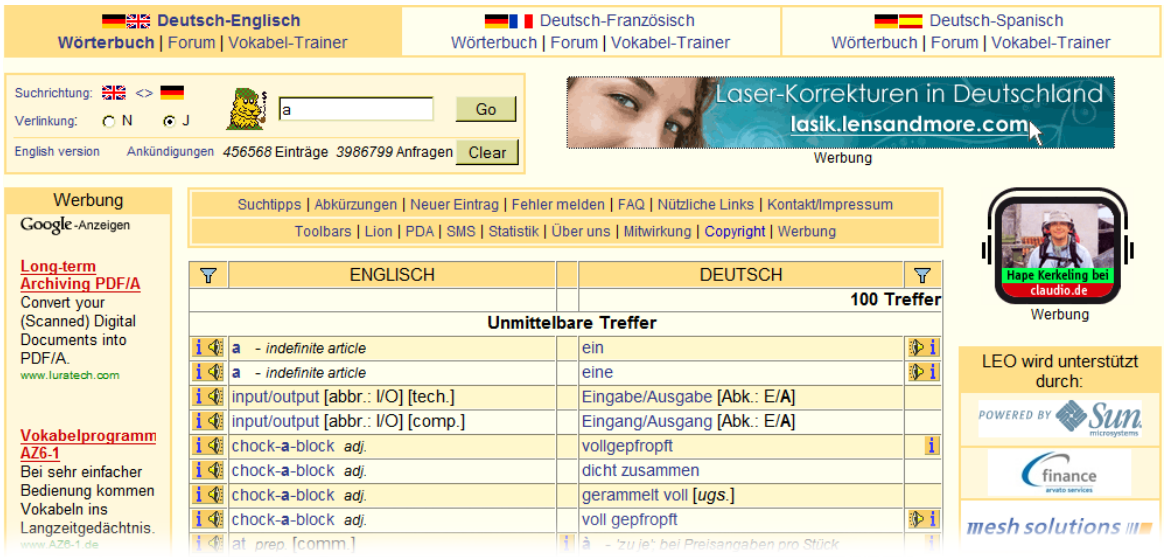



**Abbildung 2.9:** Screenshot www.leo.org

Wie in Abbildung 2.9 zu sehen ist, kann ganz oben im Bild die Sprache ausgewählt werden. Links darunter befindet sich das Suchfenster, wo der Suchbegriff eingeben werden kann. Außerdem kann die Suchrichtung verändert werden. Klickt man auf ein Flaggen-Symbol, dann wird versucht, den Suchbegriff nur in die ausgewählte Sprache zu übersetzen.

In der Ergebnistabelle werden maximal einhundert Treffer angezeigt. Wenn zusätzliche Informationen zu einem Treffer vorhanden sind, dann wird das durch ein i-Symbol angedeutet. Ist das Wort auch vertont, so wird ein Lautsprecher-Symbol angezeigt. Über das i-Symbol kann ein Fenster (Info-Fenster) geöffnet werden, das Verknüpfungen zu externen Webseiten enthält. Die verlinkten Seiten enthalten z. B. Informationen zur Konjugation, Deklination oder Definition des Wortes. Beim Klicken auf das Lautsprecher-Symbol wird das Wort nicht sofort abgespielt, sondern es erscheint das Info-Fenster. In diesem wird dann zu einer Seite verlinkt, welche das Wort abspielt. Die Wörter in der Ergebnistabelle werden grundsätzlich als Link dargestellt. Beim Klicken auf den Link wird das ausgewählte Wort als Suchbegriff übernommen und die Suche wird gestartet. So ist eine einfache weiterführende Suche möglich.

Unter der Ergebnistabelle werden auch Forumsdiskussionen angezeigt, die den Suchbegriff enthalten.

### 2.4.2 dict.cc

Das Deutsch-Englisch-Wörterbuch dict.cc wurde von Paul Hemetsberger entwickelt und enthält derzeit ungefähr 540.000 Wörter [Hem07]. Im Durchschnitt besuchen täglich etwa eine Million Benutzer die Seite.


**Abbildung 2.10:** Screenshot dict.cc

Wie in Abbildung 2.10 zu sehen ist, verfügt das Suchfenster über keine Auto-Vervollständigung. Bei falsch geschriebenen Wörtern werden dem Benutzer jedoch Vorschläge unterbreitet.

In der Ergebnistabelle können Wörter durch das Klicken auf das Lautsprecher-Symbol angehört werden. Allerdings wird die Stimme vom Computer generiert. Für jedes Wort kann ein eigenes kleines Fenster geöffnet werden, indem sich Verknüpfungen zu weiteren externen Online-Wörterbüchern befinden.

Dict.cc bietet neben der Suchfunktion viele weitere Besonderheiten. So gibt es einen Vokabeltrainer, eine persönliche Vokabelliste, ein Forum sowie die Möglichkeit, Wörter und Vokabeln als Textdatei zu exportieren.

Zur Erweiterung des Wörterbuchs können auch angemeldete Benutzer beitragen, indem sie in einer Eingabemaske neue Wörter vorschlagen. Wurde das Wort von mindestens zehn weiteren Benutzern verifiziert, dann wird es ins Wörterbuch übernommen.

### 2.4.3 Ostarrichi

Das Online-Wörterbuch Ostarrichi.org ist spezialisiert auf umgangssprachlich verwendete Wörter im österreichischen Sprachraum und kommt daher der Intention des FIM-Wörterbuchs sehr nahe. Die Webseite wird von ca. 1500 bis 3000 Benutzern pro Tag besucht [Rus07].


**Abbildung 2.11:** Screenshot www.ostarrichi.org

In Abbildung 2.11 im Fenster „Wortsuche“ kann der Suchbegriff eingegeben werden. Außerdem sieht man Suchbegriffe, die andere Benutzer kürzlich eingegeben haben. Eine Auto-Vervollständigung, eine phonetische Suche oder eine Tippfehlerkorrektur stehen bei der Eingabe nicht zur Verfügung.

Bei der Suche werden nur exakt geschriebene Wörter bzw. Teilwörter gefunden. Ähnlich klingende oder geschriebene Wörter werden nicht vorgeschlagen. Die Ergebnistabelle zeigt neben der Übersetzung auch den Bekanntheitsgrad und die Qualität des Wortes an. Diese Kriterien können von jedem Benutzer durch eine Bewertung beeinflusst werden. Des weiteren kann die Ergebnistabelle durch Auswählen verschiedener Kategorien gefiltert werden.

Das Vertonen von Wörtern und somit das Anhören ist ebenfalls nicht möglich. Ein interessantes Feature ist jedoch, dass beim Anklicken eines Wortes der Bekanntheitsgrad

ortsabhängig auf einer Österreichkarte dargestellt wird. Außerdem kann das Wort über weiterführende Verknüpfungen im Duden und der Presse gesucht werden.

Von jedem registrierten Benutzer können Wörter hinzugefügt werden, was ein rasches Füllen des Wörterbuches ermöglicht. Aufgrund der Bewertung anderer Benutzer wird dann entschieden, ob das Wort im Datenbestand bleiben darf oder gelöscht werden muss.

Zusätzlich zum Wörterbuch gibt es ein Forum, wo sich Benutzer über Themen wie z. B. Grammatik und Wortbedeutungen austauschen können.

### 2.4.4 BeoLingus

BeoLingus heißt das Online-Wörterbuch der Technischen Universität Chemnitz, welches schon seit 1995 angeboten wird. Zur Zeit enthält es ca. 350.000 Übersetzungen für Deutsch-Englisch und ca. 105.000 Übersetzungen für Deutsch-Spanisch [Gle07].

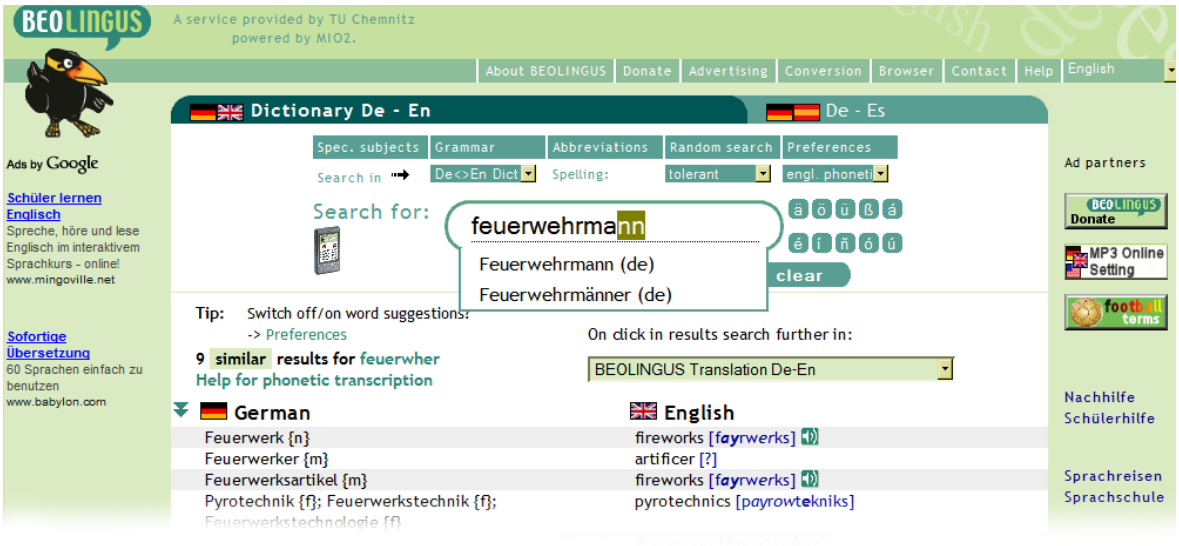



**Abbildung 2.12:** Screenshot BeoLingus

Wie in Abbildung 2.12 zu sehen ist, unterstützt BeoLingus den Benutzer mit einer Auto-Vervollständigung. Diese kann bei Bedarf ein- und ausgeschaltet werden. Wenn kein exaktes Ergebnis gefunden wird, dann schlägt das Online-Wörterbuch ähnliche Wörter mittels phonetischer Suche vor.

Über das grüne Lautsprecher-Symbol in der Ergebnistabelle können vertonte Wörter direkt im Browser angehört werden. Im Gegensatz zu den anderen vorgestellten Online-Wörterbüchern sind bei vielen Wörtern Beispielsätze vorhanden.

### 2.4.5 Fazit

In der Tabelle 2.1 finden sich zusammengefasst die wichtigsten Kriterien der Evaluierung wieder, die sich größtenteils mit den Anforderungen aus Abschnitt 2.1 decken. Ein „+“ wurde vergeben, wenn das Kriterium vollständig erfüllt wurde. Ein „o“, wenn es teilweise erfüllt wurde und ein „-“ wurde vergeben, wenn das Kriterium nicht erfüllt wurde.

| **Kriterien** | **LEO** | **Ostarrichi** | **dict.cc** | **BeoLingus** |
|---|---|---|---|---|
| phonetische Suche | o | - | + | + |
| Auto-Vervollständigung | - | - | - | + |
| Aussprache | + | - | o | + |
| Anmerkungen/Kategorien | + | + | + | + |
| weiterführende Links | + | - | + | + |
| zusätzliche Angebote | + | - | + | - |
| oberösterreichischer Dialekt | - | + | - | - |

**Tabelle 2.1:** Evaluierung von Online-Wörterbüchern

Eine phonetische Suche wird von fast allen Wörterbüchern in irgendeiner Weise angeboten. Teilweise unterschieden sich die Vorschläge jedoch sehr. Beim falsch geschriebenen Wort „feurwehr“ beispielsweise liefern nur dict.cc und BeoLingus Vorschläge.

Verwunderlich ist, dass lediglich BeoLingus eine Auto-Vervollständigung anbietet.

Bei der Aussprache gibt es ebenfalls Unterschiede. Auf Dict.cc werden die Wörter von einer computergenerierten Stimme vorgelesen, während bei den anderen Wörterbüchern Aufnahmen abgespielt werden.

Kategorien bzw. Anmerkungen werden bei allen Online-Wörterbüchern angezeigt. Teilweise kann auch nach Kategorien gefiltert werden.

Die meisten Wörterbücher bieten auch weiterführende Verknüpfungen an, um auf anderen Webseiten mehr Information zu einem Wort zu erhalten.

Zusätzliche Angebote wie z. B. Foren, Vokabellisten und Vokabeltrainer bieten nur LEO und dict.cc an.

Weitere, bekannte Online-Wörterbücher sind u. a.:

- Linguadict (`http://www.linguatec.net/onlineservices/linguadict`)
- OneLook, ein Meta-Online-Wörterbuch (`http://www.onelook.com/`)
- e-Woerterbuch, vor allem technische Begriffe (`http://www.e-woerterbuch.de/`)
- Pons (`http://www.ponsline.de/cgi-bin/wb/w.pl`)

# Kapitel 3

# Auswahl der Komponenten

Die meisten IT-Projekte, wie auch diese Magisterarbeit, werden nicht von Grund auf neu implementiert, normalerweise werden verschiedenste Frameworks und Komponenten verwendet. Oft stehen auch mehrere Alternativen zur Auswahl. In diesem Kapitel werden nun ausgewählte Komponenten vorgestellt, die für diese Arbeit in Frage kommen.

## 3.1 Basis Web-Technologien für Java

Da für das zu implementierende System die Programmiersprache Java verwendet wird, wird nun in diesem Abschnitt nur auf Java-basierte Technologien eingegangen.

### 3.1.1 Servlets

Java Servlets und JavaServer Pages (JSP) sind die grundlegenden Technologien in der Entwicklung von Web-Anwendungen. Da alle Java Webframeworks, die in dieser Arbeit vorgestellt werden, mehr oder weniger auf diese Technologien aufbauen, werden die Funktionsweisen detailliert erklärt.

Wenn ein Client einen *Request* (Anfrage) an den HTTP-Server schickt, wird dieser an das Servlet weitergeleitet. Das Servlet verarbeitet den Request und erzeugt einen *Response* (Antwort), der dann an den Client zurückgeschickt wird.

### Servlet Request

Ein Request stellt dem Server die Informationen darüber zur Verfügung, was der Client genau haben möchte. Im Folgenden sehen Sie ein Beispiel eines gültigen HTTP-Requests:

```
1 GET /index.html HTTP/1.1
2 Host: www.fim.uni-linz.ac.at
3 User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.4)
4 Accept: image/gif, image/jpeg, image/pjpeg, image/png, */*
5 Accept-Language: en-us
6 Accept-Charset: iso-8859-1,*,utf-8
```

**Listing 3.1:** Servlet Request

In der ersten Zeile werden der Typ, die zu ladende Ressource und die Protokollversion angegeben. In diesem Fall ist der Request vom Typ `GET`. In der zweiten Zeile wird die URL zum Webserver angegeben. Die übrigen Zeilen enthalten Informationen über den Browser, das verwendete Betriebssystem und welche Sprache im Browser eingestellt ist. Diese Daten ermöglichen dem Server auf die verschiedenen Browser und Sprachen Rücksicht zu nehmen [Ber02].

### Servlet Response

Wenn ein Webserver einen Request empfängt, dann kann er auf Grund der URL und der Informationen im Header entscheiden, wie er verarbeitet werden muss. Solange der Browser einen Response erhält, ist es ihm egal, wie bzw. wer den Response erzeugt hat. Im folgenden Beispiel beginnt der Header mit dem Protokollnamen und der Protokollversion, gefolgt von einem Statuscode und einer Statusbeschreibung. Der Statuscode 200 bedeutet, dass der Request erfolgreich verarbeitet wurde. Der `Content-Type` Header informiert den Browser über den Typ der Daten im Response und durch den `Content-Length` Header weiß der Browser, wie groß der Datenbereich ist. Die übrigen Header-Informationen erklären sich von selbst [Ber02].

```
1 HTTP/1.1 200 OK
2 Date: Sat, 14 Jul 2007 12:59:01 GMT
3 Server: Microsoft-IIS/6.0
4 Content-Length: 59
5 Content-Type text/html
```

```
6
7 <html>
8   <body>
9     <h1>Hello World!</h1>
10  </body>
11 </html>
```

**Listing 3.2:** Servlet Response

Bei einem Request des Clients wird im Servlet eine Methode aufgerufen, die ein Request-Objekt und ein Response-Objekt als Parameter hat. Je nachdem, ob der Request vom Typ `GET` oder `POST` ist, wird im Servlet die Methode `doGet` bzw. `doPost` abgearbeitet [Mah03]. Im Listing 3.3 ist ein Servlet zu sehen, das den Text „Hello World“ in den Response schreibt. Um HTML-konform zu sein, müsste man eigentlich auch `html`-, `head`- und `body`-Tags an den entsprechenden Stellen ausgeben. Nur um die Struktur eines Servlets zu sehen, ist dies jedoch nicht notwendig.

```
1 public class HelloWorldServlet extends HttpServlet{
2    public void doGet(HttpServletRequest req, HttpServletResponse res)
3       throws ServletException, IOException{
4
5    //  Pass all GET request to the the doPost method
6    doPost(req,res);
7    }
8
9    public void doPost(HttpServletRequest req, HttpServletResponse res)
10      throws ServletException, IOException{
11
12   //  Set the content type of the response
13   res.setContentType("text/html");
14   //  PrintWriter to write text to the response
15   PrintWriter out=res.getWriter();
16   //  Write Hello World
17   out.println("<h1>"+"Hello World!"+"</h1>");
18   out.close();
19   }
20 }
```

**Listing 3.3:** HelloWorldServlet.java (aus [Wil03])

In Abbildung 3.1 ist die Architektur einer Servlet-Anwendung zu sehen. Bei einem Request entscheidet der HTTP-Server, ob er den Request an den Servlet-Container weiter-

leiten muss, oder ob nur ein statischer Inhalt angefordert wird. Ist der Servlet-Container für die Verarbeitung zuständig, dann lädt er das angeforderte Servlet, welches dann den Response generiert. Der Response wird dann wieder über den Servlet-Container, den HTTP-Server und den Client übertragen. In der Literatur wird manchmal auch von einer Servlet-Engine anstatt eines Servlet-Containers gesprochen [Kur02].

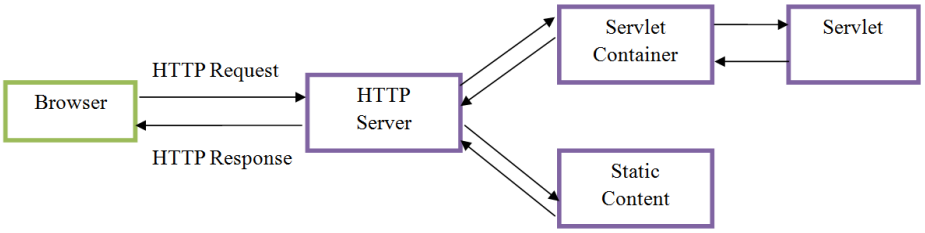



**Abbildung 3.1:** Servlet Anwendungsarchitektur (nach [Kur02])

### 3.1.2 JavaServer Pages

Für kleine Webseiten lassen sich mit der Servlet-Technologie relativ rasch dynamische Seiten programmieren. Wird die Seite jedoch umfangreicher, dann können Servlets schnell unübersichtlich werden, da typischerweise die gesamte Ansicht und Logik in einem Servlet steckt. Soll also eine kleine Änderung am Design durchgeführt werden, dann benötigt man einen Programmierer, der die Änderungen durchführt und anschließend den Code neu kompiliert. Ein weiteres Manko bei Servlets ist, dass statische Inhalte einer Seite über die `out.println()`-Methode ausgegeben werden müssen. Bei umfangreichen Seiten kann das sehr aufwendig werden. Aus demselben Grund können Servlets auch nicht direkt mit Webeditoren bearbeitet werden. Es besteht nur die Möglichkeit, HTML-Code in einem Webeditor zu erzeugen und diesen dann händisch in das Servlet zu kopieren. Um diese Probleme zu umgehen, entwickelte Sun die JavaServer Pages (JSP) - Technologie [Ber02].

#### Allgemeiner Aufbau

Grundsätzlich ist eine JSP-Seite eine normale HTML-Webseite mit JSP-Elementen, die oft auch als JSP-Tags bezeichnet werden. In Listing 3.4 ist eine „Hello World“ JSP-Seite zu sehen. Zeile 4 und 7 sind in diesem Beispiel JSP-Elemente. Genauer gesagt wird in

Zeile 4 ein String mit dem Text „Hello World“ initialisiert und in Zeile 7 wird dieser Text ausgegeben. Der restliche Quellcode wird als *Template Text* bezeichnet. Obwohl in den meisten Fällen HTML-Code als Template verwendet wird, kann im Grunde jeder beliebige Text verwendet werden [Ber02].

```
1 <html>
2    <head>
3       <title>Hello World</title>
4       <%! String message="Hello World!"; %>
5    </head>
6    <body>
7       <h1><%= message %></h1>
8    </body>
9 </html>
```

**Listing 3.4:** HelloWorld.jsp

### Verarbeitung von JSP’s

Ebenso wie ein Servlet einen Servlet-Container benötigt, braucht eine JSP-Seite einen *JSP-Container*. Wie in Abbildung 3.2 zu sehen ist, wird die JSP-Seite zuerst in ein entsprechendes Servlet konvertiert. Dies geschieht, indem der Template Text durch die `out.println()`-Methode ergänzt wird. JSP-Tags werden zum entsprechenden Java-Code konvertiert. Das Ergebnis ist dann der Servlet-Quellcode, was gleichzeitig das Ende der „Translation phase“ darstellt. Nach dem Kompilieren des Quellcodes beginnt die zweite Phase, nämlich die „Request processing phase“. In dieser Phase wird das Servlet letztendlich ausgeführt.

Solange eine JSP-Seite nicht verändert wird, muss die „Translation phase“ nicht erneut durchlaufen werden, es kann also direkt in die „Request processing phase“ gesprungen werden.

Bei den meisten Webservern sind der Servlet-Container und der JSP-Container bereits in einem Modul zusammengefasst, welches dann als Web-Container bezeichnet wird [Ber02].

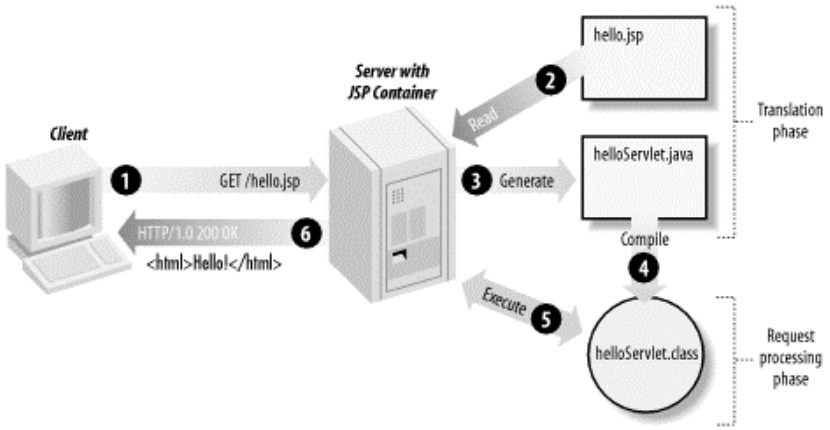



**Abbildung 3.2:** Verarbeitung von JSP's (aus [Ber02])

### JSP-Elemente

Wie vorhin bereits beschrieben, besteht eine JSP-Seite aus JSP-Elementen und dem Template Text. In diesem Abschnitt wird auf die unterschiedlichen Elementtypen eingegangen. JSP-Elemente können in zwei unterschiedlichen Notationen eingebunden werden. Zum Einen gibt es die „<% ... %>"-Notation und zum Anderen gibt es eine XML-Notation. Bei umfangreicheren Projekten ist die XML-Notation sicherlich sinnvoller, da viele Entwicklungsumgebungen XML von Haus aus bearbeiten können und mit der anderen Notation aber nicht umgehen können [CLJ+05].

Es gibt drei Typen von Elementen:

**Directive-Elemente** dienen im Unterschied zu den anderen Elementen, als Steuerungsinstrument einer JSP-Seite und erzeugen daher keine Ausgabe. Diese Elemente geben dem Web-Container bei der Übersetzung der Seite z. B. Hinweise darüber, ob weitere Seiten eingebunden werden müssen oder ob bestimmte Tag-Bibliotheken (tag libraries) verwendet werden [CLJ+05].

**Scripting-Elemente** ermöglichen das Einbinden von Sourcecode in einer anderen Programmiersprache. Typischerweise ist dies die Sprache Java. Es können aber auch andere Sprachen verwendet werden, sofern diese der Web-Container beherrscht [CLJ+05].

**Action-Elemente** sind normale Tags, die während dem Kompiliervorgang durch entsprechenden Java Code ersetzt werden. Action-Elemente können dabei standard-

oder benutzerdefiniert sein. Standard-Action-Elemente können verwendet werden, wenn der Web-Container dem JSP 2.0 Standard entspricht [CLJ+05,Kur02].

**Weitere Elemente**

Die JSP-Technologie wurde im Lauf der Zeit immer weiterentwickelt, so entstand neben der *Expression Language* (EL) auch eine umfangreiche Bibliothek, die sogenannte *JSP Standard Tag Library* (JSTL). Durch die EL ist es z. B. möglich, komplexe Ausdrücke mit Java-Objekten auszuwerten. Die JSTL ermöglicht es, dass z. B. die Logik einer Seite in die Tags ausgelagert werden kann [CLJ+05].

## 3.2 Untersuchte Java-Webframeworks

Frameworks im Allgemeinen bieten die Möglichkeit, bereits existierendes Wissen einer Architektur wieder zu verwenden. Dem Anwender stehen also bereits grundlegende Architekturen und je nach Framework auch mehr oder weniger Funktionen zur Verfügung. Dabei unterscheiden sich die diversen Frameworks in ihrem Spezialisierungsgrad. Einige Frameworks sind bereits lauffähige Anwendungen, andere wiederum stellen lediglich ein Grundgerüst für Anwendungen dar. Obwohl Webframeworks viele Vorteile, wie die Wiederverwendbarkeit der Architektur oder der Funktionalität hat, gibt es auch Nachteile, z. B. ein hoher Einarbeitungsaufwand und fehlende Standards für die Integration bereits bestehender Frameworks [KPRR06].

In diesem Abschnitt werden nun einige Frameworks anhand ihrer Spezialisierung eingeteilt. Auf drei Web-Frameworks wird anschließend genauer eingegangen, nämlich Struts[2], JavaServer Faces und Apache Wicket. Abschließend wird geklärt, welches Framework für die Entwicklung des Systems verwendet wurde.

### 3.2.1 Taxonomie

Beinahe alle Web-Frameworks basieren auf dem MVC-Pattern, daher kann dieses Kriterium nicht für die Klassifizierung der Web-Frameworks herangezogen werden. Es gibt jedoch verschiedene Design-Prinzipien, die die meisten Web-Frameworks versuchen, so gut wie möglich zu erfüllen. [SH06] identifiziert folgende Design-Prinzipien:

**Einfachheit:** So wenig und so einfacher Code wie möglich. Möglichst wenige Konfigurationsdateien.

**Konsistenz:** Die Verwendung der Komponenten soll möglichst einheitlich sein.

**Effizienz:** Die Anwendung soll möglichst performant sein.

**Integration:** Bereits existierende Frameworks sollen einfach zu integrieren sein.

**Wiederverwendbarkeit:** Komponenten eines Frameworks sollen einfach wieder zu verwenden sein.

**Non-intrusive:** HTML-Dokumente sollen nicht durch Code belastet werden. Außerdem sollen die Webseiten durch normale HTML-Editoren zu bearbeiten sein.

**Diagnose:** Das Framework soll den Entwickler bei der Fehlersuche unterstützen.

**Entwicklungsumgebungen:** Maximale Unterstützung bei der Entwicklung durch geeignete Werkzeuge.

Natürlich können die Frameworks nicht alle Prinzipien gleich gut erfüllen und setzen daher ihre Schwerpunkte unterschiedlich.

Nach [SH06] können Webframeworks in fünf Kategorien unterteilt werden, nämlich Request-basierte, Komponenten-basierte, RIA-basierte, Hybride- und Meta-Frameworks. In folgenden Absätzen werden die wichtigsten Merkmale dieser Kategorien vorgestellt.

**Request-basierte Frameworks** verwenden Controller und Actions, um einen hereinkommenden Request zu verarbeiten. Auf Actions wird im nächsten Abschnitt zurückgekommen. Grundsätzlich ist ein Request zustandslos. Durch das Konzept der Server-seitigen Sessions können jedoch Zustände gespeichert werden. Die Frameworks dieser Kategorie unterscheiden sich hauptsächlich darin, wie die Logik auf die URLs abgebildet wird und auf welche Art und Weise die Daten der Geschäftslogik zur Verfügung gestellt werden. Typische Vertreter dieser Kategorie sind z. B. Struts[1], Beehive[2] und Stripes[3].

[1] `http://struts.apache.org/`
[2] `http://beehive.apache.org/`
[3] `http://stripes.mc4j.org/confluence/display/stripes/Home`

**Komponenten-basierte Frameworks** bestehen, wie der Name schon sagt, aus vielen eigenständigen Komponenten. So ist die Logik beispielsweise in der Komponente gekapselt. Auch der Zustand wird von der Komponenten-Instanz selbst verwaltet. Der Aufbau einer Web-Anwendung erfolgt ähnlich der Entwicklung einer GUI im Desktop-Bereich. Wichtige Vertreter dieser Art sind unter anderem JavaServer Faces, Tapestry[4] und Wicket.

**Hybride-Frameworks** kombinieren die beiden oben genannten Kategorien, so wird z. B. der gesamte Datenfluss und der Ablauf der Logik Request-basiert gesteuert. Dadurch hat der Entwickler die Kontrolle über URLs, Form-Parameter und Cookies. Dennoch steht dem Entwickler ein Komponenten-Modell zur Verfügung. Damit ist es möglich, Komponenten zu gruppieren und diese als eigenständige Komponente wieder zu verwenden. Zusammengefasst kann man sagen, dass trotz der guten Steuerungsmöglichkeit ein hoher Grad an Wiederverwendbarkeit gegeben ist. RIFE[5] ist z. B. ein typischer Vertreter dieser Kategorie.

**Meta-Frameworks** stellen nur grundlegende Dienste zur Verfügung, sind aber sehr einfach und flexibel erweiterbar. Sie stellen also nur ein Grundgerüst dar, in dem weitere bestehende Frameworks eingebettet werden können. Meist liegt diesem Framework das Inversion of Control Pattern zugrunde. Meta-Frameworks werden oft auch als Framework über Frameworks bezeichnet. Aktuelle Vertreter dieser Art sind z. B. Spring[6] und Keel[7].

**RIA-basierte Frameworks** verwenden ein Client-seitiges Modell, um mit dem Server möglichst wenig kommunizieren zu müssen. Bei einer Benutzerinteraktion kann das Framework auf zwei Arten reagieren. Zum Einen kann die Anfrage lokal bearbeitet werden, d. h. ohne jegliche Kommunikation mit dem Server. Zum Anderen können jedoch Daten vom Server benötigt werden, was eine Kommunikation erfordert. Anstatt die gesamte Webseite neu zu laden, wird bei diesen Frameworks nur jener Teil neu geladen, der auch wirklich aktualisiert werden muss. Es besteht also eine richtige Client-seitige Anwendung, die ein Zustandsmodell und ein Interaktionsmodell zur Verfügung stellt. Vertreter dieser Kategorie sind

[4] `http://tapestry.apache.org/`
[5] `http://rifers.org/`
[6] `http://www.springframework.org/`
[7] `http://www.keelframework.org/`

z. B. DWR[8], Echo2[9] und JSON-RPC-Java[10].

In den nächsten Abschnitten werden nun drei Web-Frameworks detaillierter beschrieben. Da bei dem Online-Wörterbuch die Wahl des Frameworks eher zu einem Komponenten-basierten Framework tendierte, wurden zwei Vertreter dieser Kategorie ausgewählt, nämlich JavaServer Faces und Wicket. Dennoch wurde auch ein Request-basiertes Framework in Betracht gezogen, welches in der Web-Entwicklung sehr verbreitet ist: Struts$^2$.

### 3.2.2 Struts$^2$

Struts$^2$ ist eine Weiterentwicklung aus der Kombination von den Frameworks Struts und Webwork und basiert auf vielen Standard-Technologien wie z. B. Java Beans, ResourceBundles und XML. Zu den wesentlichen Erweiterungen gegenüber den früheren Versionen zählen unter anderem eine AJAX-Unterstützung, eine einfache Integration von Java-Spring, Annotations und viele erweiterte Tags.

In diesem Abschnitt wird nun auf die grundlegende Architektur, sowie auf den Ablauf bei einem Request näher eingegangen.

#### Architektur

Struts$^2$ basiert auf der MVC-Architektur, die oben bereits detailliert beschrieben wurde. Abbildung 3.3 zeigt, wie die Struts$^2$-Komponenten in das MVC-Konzept passen. Der Controller agiert dabei als Vermittler zwischen der View und den verschiedenen Model-Komponenten. Realisiert ist der Controller aus einem *Servlet-dispatch-filter* und mehreren *Interceptoren*. Durch Interceptoren kann ein Request vorverarbeitet bzw. nachbearbeitet werden. Typische Aufgaben, die Interceptoren durchführen, sind z. B. das Loggen von Informationen oder das Ver- bzw. Entschlüsseln von Daten. Im Model befinden sich sowohl die Anwendungsdaten, als auch sogenannte *Actions*. Der Servlet-dispatch-filter stellt sicher, dass ein hereinkommender Request an die zuständige Action-Klasse weitergeleitet wird. In der Action-Klasse wird dann die Geschäftslogik ausgeführt und der

[8] `http://getahead.org/dwr`
[9] `http://nextapp.com/platform/echo2/echo/`
[10] `http://oss.metaparadigm.com/jsonrpc/`

Zustand der Web-Anwendung geändert. Nachdem die Geschäftslogik ausgeführt wurde, wählt der Controller die View aus, die im Client angezeigt werden soll. Obwohl in Abbildung 3.3 nur JSP als View angegeben ist, können viele verschiedene Technologien zum Einsatz kommen, wie z. B. JSP, Velocity, Freemaker, JSF und XSLT. Zur Konfiguration benötigt Struts$^2$ mindestens drei Dateien, nämlich *web.xml*, *struts.xml* und *struts.properties*, die während der Entwicklung stets angepasst werden müssen [Raj07b].

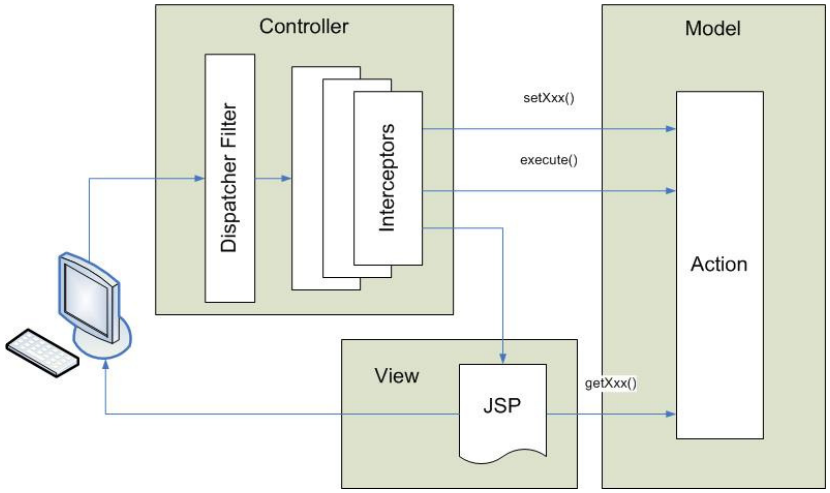



**Abbildung 3.3:** Struts$^2$ Architektur [Rou06]

### Detaillierter Ablauf bei einem Request

Die folgenden Schritte zeigen, wie sich eine Struts$^2$- Anwendung verhält, wenn eine Seite aufgerufen wird [Raj07b].

1. Der Web-Browser sendet einen Request an die Web-Anwendung.
2. Der Servlet-dispatch-filter wird geladen und sucht aus der Konfiguration die Action-Klasse, welche für diesen Request zuständig ist.
3. Bevor der Request jedoch an die Action-Klasse weitergeleitet wird, kann der Request durch eine Reihe von Interceptoren laufen. Danach wird der Request an die Action-Klasse weitergeleitet.
4. Die Action-Klasse führt schließlich, unter Zuhilfenahme der Request-Parameter, die Geschäftslogik aus. Danach gibt die Action-Klasse einen Ergebnisstring zurück. Der Ergebnisstring kann dabei entweder *success* oder *error* sein.

5. Aufgrund des Ergebnisstrings kann der Controller nun entscheiden, welche View dem Benutzer angezeigt werden soll.

### 3.2.3 JavaServer Faces

JavaServer Faces (JSF) ist ein Framework-Standard, der mit Hilfe des Java Community Process (JCP) - Programms entwickelt wurde. Im JCP-Programm sind viele namhafte Firmen wie z. B. Sun und IBM beteiligt. Sie haben sich zum Ziel gesetzt, Java-Technologien weiter zu entwickeln und Standards zu schaffen [N.N07b].

Ziel bei der Entwicklung von JSF war es, eine schnelle Entwicklung von Web-Anwendungen zu ermöglichen. D. h., dass das Framework dem Entwickler bei immer wiederkehrenden Problemstellungen unterstützen soll. Die wichtigsten Funktionen von JSF sind unter anderem die Verwaltung von Web-Anwendungs-Zuständen, Validierung und Konvertierung von Daten und eigene User-Interface (UI) - Komponenten. UI-Komponenten können relativ einfach an Anwendungsdaten gekoppelt werden. Außerdem stehen Tag-Libraries zur Verfügung, um JSF-Komponenten in JSP-Seiten einbinden zu können. Ein weiteres wichtiges Konzept in JSF sind Events. Ähnlich wie bei Java Swing können verschiedene Events ausgelöst und an registrierte Listener weitergegeben werden.

In diesem Abschnitt wird zuerst auf die Architektur des Frameworks und anschließend auf die Navigationssteurung, eingegangen.

#### Architektur

Wie Struts[2] ist auch JSF nach dem MVC-Prinzip aufgebaut. Wie man in Abbildung 3.4 sieht, besteht der Controller nur aus einem *FacesServlet*. Dieses Servlet leitet die Requests an die zuständige JSF-Seite weiter.

Im Model werden bei JSF *Managed Beans* verwendet, welche den Ursprung in den JavaBeans haben. Der Vorteil von Managed Beans besteht darin, dass UI-Komponenten einfach mit der Geschäftslogik verbunden werden können, z. B. um Formulardaten auf ein Modell abzubilden. Um Managed Beans verwenden zu können, müssen sie jedoch in der `faces-config.xml` definiert werden. In dieser Konfigurationsdatei kann auch der

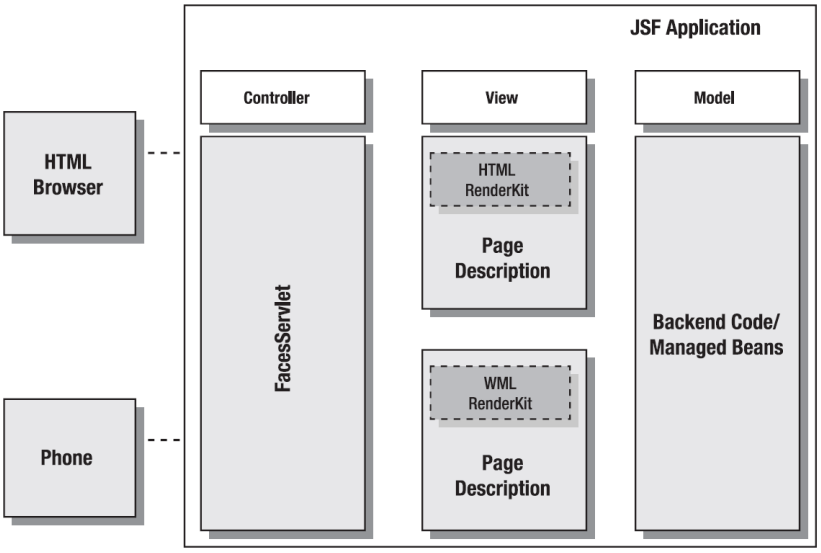



**Abbildung 3.4:** JSF MVC-Architektur [JF06]

Gültigkeitsbereich der Managed Bean bestimmt werden, z. B. ob das Managed Bean nur für die Dauer eines Requests oder über die ganze Session gültig sein soll [JF06,Sch05b].

Die View definiert das Layout und das Verhalten der Anwendung und wird durch die *Page Description* und einen *Render Kit* festgelegt. Während die Page Description die Hierarchie der UI-Komponenten beschreibt, hat der Render Kit die Information, welcher Code für eine bestimmte UI-Komponente erzeugt werden muss. Standardmäßig ist JSF so eingestellt, dass HTML-Code erzeugt wird. Es kann aber genauso gut ein Render-Kit definiert werden, der WML-, XML- oder XUL-Code ausgibt [JF06].

### Navigationssteuerung und Konfiguration

Neben der Datei `web.xml`, in der das JavaServer-Faces-Servlet konfiguriert werden kann, muss auch noch eine Datei mit dem Namen `faces-config.xml` vorhanden sein. In dieser Datei befinden sich u. a. Informationen über Managed Beans.
`faces-config.xml` hat jedoch noch viele weitere wichtige Funktion. Eine davon ist die Navigationssteuerung. Für jede Seite können Navigationsregeln erstellt werden. In Abhängigkeit von der Eingabe des Benutzers wird dann aufgrund dieser Regeln die nächste Seite bestimmt [Sch05b].

### 3.2.4 Wicket

Wicket ist ein relativ junges Web-Framework. Version 1.0 wurde 2005 veröffentlicht. Im Gegensatz zu den anderen beschriebenen Frameworks versucht Wicket die Geschäftslogik und die HTML-Seiten strikt voneinander zu trennen. Folgender Satz kann als eine kompakte Beschreibung von Wicket gesehen werden:

> Wickte is „a Java software framework to enable component oriented, programmatic manipulation of markup.“ [DH07]

Dieser Satz beschreibt die wichtigsten Ziele von Wicket. „programmatic manipulation“ bezieht sich z. B. darauf, dass die HTML-Tags und deren Inhalte nur durch Java verändert werden. Mit „component oriented“ ist gemeint, dass man eine Wicket-Anwendung in viele kleine UI-Komponenten mit eingekapselter Logik aufteilen kann. Diese Komponenten sind dann eigenständig und wiederverwendbar. Erwähnenswert ist auch, dass Wicket keine Konfigurationsdateien für die Verknüpfung der Seiten benötigt, wie das z. B. bei Struts der Fall ist. Nach [DH07] hat Wicket mehr Ähnlichkeit zum Programmieren einer Desktop-Anwendung als zu den anderen Web-Frameworks. In den folgenden Abschnitten werden nun die wichtigsten Konzepte von Wicket erklärt.

#### Basiskonzepte

Jede Wicket-Seite muss von der Klasse `WebPage` abgeleitet werden. Eine Wicket-Seite und die entsprechende HTML-Seite haben immer eine Eins-zu-Eins-Beziehung. In Wicket ist es wichtig, dass alle Komponenten in der Klasse und in der HTML-Seite die gleiche Hierarchie aufweisen. D. h. wenn eine Klasse ein Formular enthält, das wiederum ein Eingabefeld enthält, dann muss die HTML-Seite ebenfalls diese Komponenten in derselben Ordnung enthalten. Die HTML-Seite kann in Wicket also als „View“ im MVC-Konzept betrachtet werden. Damit die Wicket-Seite der HTML-Seite zugeordnet werden kann, müssen diese bis auf die Dateiendung den gleichen Namen tragen [Gur06]. Die folgenden zwei Listings sollen die Zusammengehörigkeit der beiden Dateien verdeutlichen.

```
1 public class HelloWorld extends WebPage {
2    public HelloWorld() {
3       add( new Label("messageId", "Hello World!") );
```

```
4 }
```

**Listing 3.5:** helloworld.java

```
1 <html>
2     <body>
3         <div wicket:id="messageId">Message goes here</div>
4     </body>
5 </html>
```

**Listing 3.6:** helloworld.html

Im Listing 3.5 kann man sehen, wie die Komponenten zu einer Seite hinzugefügt werden. In diesem Fall wird lediglich ein `Label`-Objekt hinzugefügt. Das `Label`-Objekt hat zwei Parameter. Einerseits einen eindeutigen Identifikations-String (ID) und andererseits ein Modell, welches den Text des Labels enthält. Damit Wicket die Java-Komponente in der HMTL-Datei zuordnen kann, muss auch in der HTML-Datei die ID aufscheinen. Dies geschieht mit dem Attribut „wicket:id“. Die ID muss dabei eindeutig sein und darf in der HTML-Datei nicht mehrfach vorkommen [Gur06].

Das Hinzufügen geschieht über die Methode `add()`. Da jede Komponente eine `add()`-Methode hat, und somit weitere Komponenten aufnehmen kann, können komplexe Hierarchien aufgebaut werden. In Wicket haben die Komponenten selbst die Information darüber, wie sie gerendert werden. Der folgende Ablauf zeigt, wie das Rendern einer Webseite erfolgt.

**Render-Phase:**

1. Die Render-Phase wird durch den Aufruf der Methode `Page.render()` eingeleitet.
2. Wicket sucht die dazugehörige HTML-Datei, iteriert über die HTML-Tags und baut eine interne Java-Darstellung der Seite auf.
3. Wird ein Tag ohne dem Attribut „wicket:id“ gefunden, dann wird dieser so ausgegeben wie er ist.
4. Wird ein Tag mit dem Attribut „wicket:id“ gefunden, dann wird die dazugehörige Komponente gesucht und das Rendern an die Komponente delegiert.

5. Das fertige Exemplar der Seite wird letztendlich angezeigt und intern in der sogenannten `PageMap` gespeichert. Wicket legt jeweils eine `PageMap` pro Session an.

### Modellkonzept

Ein weiteres wichtiges Konzept in Wicket ist das Modellkonzept. Ein Modell enthält dabei die Daten, die für eine Komponente oder auch mehrere Komponenten benötigt werden. Wurde einer Komponente ein Modell-Objekt zugewiesen, dann kann diese Daten lesen aber auch neue Daten darauf schreiben. Die Daten werden erst dann gelesen, wenn diese auch tatsächlich benötigt werden und zwar in der „Render Phase“. Bei einer `DropDownChoice` beispielsweise müssen zwei verschiedene Modelle angegeben werden. Eines für die Elemente, die zur Auswahl stehen und ein Modell für das ausgewählte Element. Verändert der Benutzer im Browser seine Auswahl, dann wird je nach Sende-Strategie das Modell auf der Server-Seite sofort geändert oder erst nach einem „submit“ [Gur06].

Da jedes Modell-Objekt das Interface `IModel` implementiert, können diese auch zur Laufzeit gebunden und geändert werden. Alle Wicket-Komponenten arbeiten nur mit Instanzen von `IModel`. D. h. welchen Wert bzw. welchen Typ das eigentliche Modell haben muss, hängt immer von der Komponente selbst ab. Ein `Label` beispielsweise muss an ein Modell gebunden werden, das in ein String-Objekt umgewandelt werden kann. Eine `ListView` wiederum muss an ein Modell gebunden werden, welches zu einem `java.util.List`-Objekt umgewandelt werden kann [N.N06c].

Im folgenden Abschnitt werden einige typische Modell-Varianten näher behandelt.

**Model:** Die Klasse `Model` ist die einfachste Implementierung des `IModel`-Interfaces. Sie kann ein beliebiges serialisierbares Java-Objekt kapseln. Der Nachteil dieses Modells ist, dass die gesamte Information in der Session abgelegt wird. Außerdem wird in einer Cluster-Umgebung das serialisierte Objekt repliziert und führt somit zu einem hohen Speicherbedarf. Es sollte daher nur für kleine Objekte verwendet werden [LTH06].

**Property Model:** Mit der schon etwas komplexeren Klasse `PropertyModel` ist es möglich, auf Java Beans oder „POJO’s“ (Plain Old Java Objects) direkt zuzugreifen. Im Konstruktor müssen eine Referenz auf ein Java Bean und ein String

übergeben werden. Der String identifiziert dabei das Feld, auf das zugegriffen werden soll. Will man z. B. in einer Klasse `Person` auf das Feld `name` zugreifen, dann muss im Konstruktor auch der String „name“ übergeben werden. Über diesen String kann das Modell nun die dazu passenden Get- und Set-Methoden herleiten [LTH06].

Normalerweise macht Wicket die Typ-Konvertierung für ein Model-Objekt automatisch. Da in speziellen Fällen eine Konvertierung in eine eigene Klasse sinnvoll sein kann, ist es möglich, durch ein weiteres Argument im Konstruktor den Typ festzulegen [N.N06c].

Für große Webseiten kann es sehr mühsam sein, jeder Komponete sein eigenes Modell zuzuweisen. Für solche Fälle gibt es die Klasse `CompoundPropertyModel`. Mit dieser ist es möglich, ein POJO an eine Webseite zu binden. Details zur Verwendung können aus [LTH06] entnommen werden.

**Detachable Model:** Wie vorhin bereits angedeutet, soll die Klasse `Model` nicht für große Modelle verwendet werden. Für diese Fälle eignet sich eine Ableitung der Klasse `AbstractDetachableModel` besser. Ist ein Detachable-Model im Zustand „detached“, dann benötigt es nur sehr wenig Speicherplatz, da auch nur wenig Information über ein Objekt, wie z. B. die Objekt-ID, vorhanden ist. Wird das gesamte Objekt benötigt, dann kann es über die Objekt-ID nachgeladen werden. Typischerweise werden die Daten aus einer Datenbank ausgelesen, dabei werden die Daten erst geladen, wenn diese auch tatsächlich benötigt werden. In einer Cluster-Umgebung müssen somit nur wenig Daten repliziert werden [Gur06, N.N06c].

**String Resource Model:** Ein spezielles Modell ist die Klasse `StringResourceModel`. Sie bietet die Möglichkeit, lokalisierte Texte aus einer Datei auszulesen. Außerdem können Texte sprachabhängig formatiert werden. Um ein `StringResourceModel` erzeugen zu können, benötigt man zumindest einen „Resource Key“ und eine Komponente. Der „Resource Key“ ist ein String, der den Text in der Datei referenziert. Die Komponente im Konstruktor wird benötigt, um den Namen der Datei aufzulösen [LTH06, N.N06c]. Weitere Information zum Thema Lokalisierung ist im Abschnitt 3.2.4 zu finden.

### Layout-Gestaltung

Wicket legt großen Wert auf die Wiederverwendbarkeit von Komponenten, daher gibt es auch mehrere Möglichkeiten, Komponenten zu gruppieren. In folgenden Absätzen werden nun vier verschiedene Möglichkeiten vorgestellt.

`Panel` Diese Komponente ist eine der wichtigsten in Wicket. Zu ihr können beliebige Komponenten hinzugefügt werden. Damit lässt sich sehr einfach eine wiederverwendbare Komponente erzeugen [N.N07f]. In Listing 3.7 ist z. B. eine Klasse `MyPanel` zu sehen, die von `Panel` abgeleitet wurde. In Zeile 5 und 6 wird jeweils eine `Label`-Komponente hinzugefügt, die einen Text anzeigen kann.

```
1 class MyPanel extends Panel{
2    public MyPanel(String id) {
3       super(id);
4
5       add(new Label("label1",new Model("My Text")));
6       add(new Label("label2",new Model("Another Text")));
7    }
8 }
```

**Listing 3.7:** MyPanel.java

Listing 3.8 zeigt das entsprechende HTML-Template. Wie in den Zeilen 3 und 4 zu sehen ist, können die Komponenten über einen Span-Tag eingebunden werden. Durch die Tags `<wicket:panel> ... </wicket:panel>` weiß Wicket, wo der HTML-Code des Panels beginnt bzw. endet. Weiterer HTML-Code außerhalb dieser Tags wird ignoriert und nicht angezeigt.

```
1 <html xmlns:wicket>
2    <wicket:panel>
3       <span wicket:id="label1">text goes here</span>
4       <span wicket:id="label2">text goes here</span>
5    </wicket:panel>
6 </html>
```

**Listing 3.8:** MyPanel.html

Durch die `MyPanel`-Komponente können nun beliebig viele `MyPanel`-Objekte erzeugt und in eine Webseite eingebunden werden. Werden mehrere solcher Panels in einer Seite eingebunden, dann müssen aber auch unterschiedliche IDs vergeben werden.

`Border` Die `Border`-Komponente kann andere Komponenten „dekorieren“. Sie erlaubt es, lose Komponenten zusammenzufassen und z. B. einen Rahmen um die Komponenten zu zeichnen. Zusammengefasste Komponenten werden also immer von der `Border`-Komponente umgeben. Anwendung findet dieses Konzept beispielsweise bei der Validierung eines Eingabefeldes. Wenn die Eingabe nicht korrekt ist, wird z. B. ein Rahmen um das Eingabefeld gelegt, um die Aufmerksamkeit des Anwenders zu erlangen.

`Include` Mit dieser Komponente kann ein beliebiger HTML-Code in eine Wicket-Seite eingebunden werden. Grundsätzlich sollte sie jedoch eher in Ausnahmefällen verwendet werden.

`Fragment` Eine `Fragment`-Komponente ist der `Panel`-Komponente sehr ähnlich. Sie kann ebenfalls beliebige Komponenten in sich aufnehmen, aber es muss keine eigenständige HTML-Datei angelegt werden. Der HTML-Code eines Fragments kann also in eine andere HTML-Datei ausgelagert werden, z. B. in die Datei, in der das Fragment eingebunden werden soll.

### AJAX-Unterstützung und -Komponenten

In Wicket können zu Komponenten verschiedene Verhaltensmuster hinzugefügt werden. Diese Verhaltensmuster werden durch Implementierungen des Interfaces `IBehavior` erreicht. So gibt es z. B. ein Verhalten `AttributeBehavior`, das zu einer Komponente ein HTML-Attribut hinzufügen kann. Für Ajax wurde ebenfalls ein solches Verhaltensmuster implementiert, das `AbstractDefaultAjaxBehavior` heißt. Folgender Ablauf zeigt exemplarisch, was das Webframework macht, wenn ein Ajax-Verhalten zu einer Komponente hinzugefügt wird:

1. Das nötige JavaScript wird automatisch erzeugt und mit dem angegebenen Clientseitigen Event (z. B. *onClick*, *onBlur*, etc.) verbunden.
2. Wenn das JavaScript durch eine Benutzerinteraktion ausgelöst wurde, wird die Komponente Server-seitig aktualisiert. Falls Daten mitgeschickt werden, können diese validiert werden.

3. Als nächstes wird die `onEvent`-Methode aufgerufen. In dieser Methode kann der Entwickler entscheiden, welche weiteren Schritte vorgenommen werden. Komponenten können beispielsweise über das Objekt `AjaxRequestTarget` auf der Client-Seite aktualisiert werden.

**Beispiel:** In Listing 3.9 wird der Ablauf noch einmal verdeutlicht. In Zeile 1 und 2 wird eine Label-Komponente erzeugt, der ein Text zugewiesen wird. Danach wird in Zeile 4 ein neues Ajax-Verhalten erzeugt und zum Label hinzugefügt. Außerdem wird das JavaScript-Event *onClick* festgelegt. Damit wird nun die `onEvent`-Methode aufgerufen, wenn auf das Label geklickt wird. In der `onEvent`-Methode wird in Zeile 8 der Text der Label-Komponente aktualisiert. Außerdem steht in der Methode ein `AjaxRequestTarget`-Objekt zur Verfügung. Komponenten, die zu diesem Objekt hinzugefügt werden, können per Ajax an den Webbrowser zurückgeschickt und dort angezeigt werden. In diesem Beispiel wird in Zeile 10 die Label-Komponente hinzugefügt.

```
1  final Label label = new Label("ajaxlabel");
2  label.setModel(new Model("Component not clicked!"));
3
4  label.add(new AjaxEventBehavior("onClick"){
5
6      @Override protected void onEvent(AjaxRequestTarget target) {
7
8          label.setModel(new Model("Component clicked!"));
9
10         target.addComponent(label);
11     }
12 });
13
14 label.setOutputMarkupId(true);
15 add(label);
```

**Listing 3.9:** Ajax-Event

D. h. wird im Browser auf den Text *Component not clicked!* geklickt, dann wird am Server der Text geändert, die Label-Komponente am Client ausgetauscht und somit der Text *Component clicked!* angezeigt.

Damit Wicket die Komponenten beim Austauschen eindeutig identifizieren kann, müssen die Komponenten im HTML-Code ein ID-Attribut haben. Dieses ID-Attribut kann

Wicket automatisch erzeugen. In Zeile 14 ist zu sehen, wie das ID-Attribut gesetzt werden kann.

Wicket hat bereits eine breite Palette an Ajax-Komponenten, u. a. ein Auto-Complete Textfield, ein editierbares Label, Ajax-Links, ein Modal-Window und Tabbed Panels.

### Benutzer-Authentifizierung

Es gibt mehrere Möglichkeiten, in Wicket einen Autorisierungsmechanismus hinzuzufügen. Eine davon ist, die Klassen `AuthenticatedWebApplication` und `AuthenticatedWebSession` zu verwenden. Sie stellen ein Grundgerüst für die Autorisierung von Benutzern zur Verfügung. Da sich die Klassen nicht im Kernpaket von Wicket befindet, muss es extra installiert werden. Das Paket heißt `wicket-auth` und kann auf der Webseite von Wicket heruntergeladen werden.

**AuthenticatedWebSession:** Wenn die Session-Klasse von `AuthenticatedWebSession` abgeleitet wird, dann müssen zwei abstrakte Methoden implementiert werden. In Listing 3.10 sind diese Methoden zu sehen. In der Methode `authenticate()` in Zeile 2 wird überprüft, ob der Benutzer überhaupt existiert und ob er berechtigt ist, auf die Web-Anwendung zuzugreifen. Normalerweise findet in dieser Methode eine Datenbankabfrage statt, die auch gleich die Benutzerdaten lädt, falls diese vorhanden sind. In Zeile 4 wird einem angemeldeten Benutzer der Status eines Administrators gegeben. Es können jedoch beliebig viele Rollen durch die Klasse `Roles` angelegt werden. Wenn der Benutzer autorisiert ist, dann wird in Zeile 5 der Wert `true` zurückgegeben. Damit erhält die Variable `signedIn` ebenfalls den Wert `true`.

```
1 @Override
2 public boolean authenticate(String username, String password) {
3     if(isUserAuthenticated(username, password)){
4         currentRole=new Roles("Admin");
5         return true;
6     }
7
8     return false;
9 }
10
11 @Override
12 public Roles getRoles() {
```

```
13     if (isSignedIn())
14     {
15         // If the user is signed in, they have these roles
16         return currentRole;
17     }
18     return null;
19 }
```

**Listing 3.10:** AuthenticatedWebSession

Die Methode `getRoles()` in Zeile 12 wird aufgerufen, wenn überprüft wird, ob ein Benutzer eine bestimmte Rolle hat. In Zeile 13 wird sicherheitshalber überprüft, ob der Benutzer angemeldet ist. Bei positiver Überprüfung wird die Rolle returniert.

**AuthenticatedWebApplication:** Neben den oben genannten Methoden werden noch zwei weitere benötigt, die sich in der Klasse `AuthenticatedWebApplication` befinden. Wie in Listing 3.11 zu sehen ist, liefert die Methode `getWebSessionClass()` die Klasse der `AuthenticatedWebSession` zurück. Diese Methode wird aufgerufen, wenn der Benutzer das erste Mal die Webseite aufruft. Mit Hilfe dieser Klasse wird dann das Session-Objekt erzeugt.

Die Methode `getSignInPageClass()` in Zeile 7 liefert die Klasse der Login-Webseite. Diese Methode wird z. B. aufgerufen, wenn ein unautorisierter Benutzer versucht, eine geschützte Seite aufzurufen. Anstatt die geschützte Seite anzuzeigen, lädt Wicket dann die Login-Webseite. Falls der Benutzer ein gültiges Passwort und einen Benutzernamen eingibt, dann wird er automatisch auf die ursprünglich angeforderte Seite weitergeleitet.

```
1 @Override
2 protected Class<? extends AuthenticatedWebSession> getWebSessionClass() {
3     return MyAuthenticatedWebSession.class;
4 }
5
6 @Override
7 protected Class<? extends WebPage> getSignInPageClass() {
8     return MySignInPage.class;
9 }
```

**Listing 3.11:** AuthenticatedWebApplication

Bis jetzt wurde in diesem Abschnitt gezeigt, welche Klassen und Methoden nötig sind, um eine Authentifizierung zu ermöglichen. In den folgenden Absätzen wird nun erklärt,

wie einzelne Komponenten und Klassen für bestimmte Rollen erlaubt bzw. gesperrt werden können.

**Webseiten mit Annotations sichern:** Eine Webseite nur einer bestimmten Rolle zu erlauben, kann mit der Annotation `AuthorizeInstantiation` erzwungen werden. Diese Annotation kann sowohl auf Klassen, als auch auf ganze Packages angewendet werden. Generell muss bei Annotations in Verbindung mit Packages eine Hilfsdatei (`package-info.java`) [Fri06] verwendet werden. Webseiten können z. B. abhängig von der Rolle in unterschiedliche Packages gegeben werden. Damit können mit einer einzigen Annotation gleich mehrere Webseiten geschützt werden [LTH06].

In Listing 3.12 ist ein Beispiel zu sehen, wie eine Webseite gesichert werden kann. In Zeile 3 wird definiert, dass nur ein Benutzer mit der Rolle *Admin* die Seite öffnen darf. Außer dem Hinzufügen der Annotation muss nichts an der Klasse geändert werden. Versucht ein unautorisierter Benutzer die Webseite zu öffnen, dann wird er nun automatisch zur Login-Seite weitergeleitet.

```
1 import wicket.authorization.strategies.role.annotations.
    AuthorizeInstantiation;
2
3 @AuthorizeInstantiation("Admin")
4 public class SecuredPage extends WebPage {
5
6    public SecuredPage(final PageParameters parameters) {
7       ...
8    }
9 }
```

**Listing 3.12:** Gesicherte Wicket-Webseite

**Komponenten mit Annotations sichern:** Neben ganzen Webseiten können auch einzelne Wicket-Komponenten gesichert werden. Mit der Annotation `AuthorizeAction` kann festgelegt werden, welche Rollen die Komponente verwenden dürfen. Außerdem wird definiert, wie sich die Komponente verhalten soll, wenn der Benutzer autorisiert bzw. nicht autorisiert ist. In Listing 3.13 ist eine `AjaxLink`-Komponente zu sehen, die nur für Benutzer, welche die Rolle *Admin* oder *User* haben, angezeigt wird. Das Verhalten der Komponenten wird durch die Variable `action` in Zeile 3 festgelegt. In diesem

Beispiel wird der Variable der Wert `Action.RENDER` zugewiesen. Damit weiß Wicket, dass die Komponente nur angezeigt werden soll, wenn der Benutzer die entsprechende Berechtigung besitzt. Die Variable kann aber auch den Wert `Action.ENABLE` annehmen. Dadurch wird die Komponente zwar immer angezeigt, sie kann aber nicht verwendet werden, wenn keine ausreichende Berechtigung vorhanden ist.

```
1 import wicket.authorization.strategies.role.annotations.AuthorizeAction;
2
3 @AuthorizeAction(action = Action.RENDER, roles = {"Admin", "User"})
4 public class MyAjaxLink extends AjaxLink {
5
6     public MyAjaxLink(String id) {
7         ...
8     }
9 }
```

**Listing 3.13:** Gesicherter Wicket-Link

### Lokalisierung

Wicket bietet mehrere Möglichkeiten, Webseiten zu lokalisieren. In den folgenden Absätzen werden nun einige Ansätze vorgestellt.

**Property-Dateien:** In den Property-Dateien stehen die lokalisierten Strings einer Webseite. In den Listings 3.14 und 3.15 sind z. B. zwei Property-Dateien für eine Login-Seite zu sehen. Dabei ist zu beachten, dass der Dateiname der Property-Dateien eine wichtige Rolle spielt. Er muss nämlich immer mit dem Dateinamen der Webseite beginnen. Für die Seite `LoginPage.java` muss die dazugehörige Property-Datei `LoginPage.properties` wie im Listing 3.14 heißen. Im Listing 3.15 ist die Property-Datei für die Sprache Deutsch zu sehen. Damit Wicket die Sprachen unterscheiden kann, werden die Dateien mit dem Kürzel der jeweiligen Sprachen versehen, z. B. *_de* für Deutsch oder *_es* für Spanisch. Wenn im Browser eine Sprache eingestellt ist, für die es keine Lokalisierung gibt, dann wird jene Property-Datei herangezogen, die kein Sprachkürzel im Namen hat. In diesem Beispiele würde die Datei `LoginPage.properties` verwendet werden.

```
1 name=Username
```

```
2 pwd=Password
```

**Listing 3.14:** LoginPage.properties

```
1 name=Benutzername
2 pwd=Passwort
```

**Listing 3.15:** LoginPage_de.properties

Wie in obigen Listings zu sehen ist, werden die lokalisierten Strings immer über einen Schlüssel referenziert. In diesem Beispiel werden die Schlüssel *name* und *pwd* verwendet.

**Lokalisierung über den `<wicket:message>` Tag:** Diesen Tag zu verwenden, ist die einfachste Möglichkeit, einen String sprachabhängig anzuzeigen. Der `<wicket:message>` Tag kann dabei an einer beliebigen Stelle in der HTML-Datei stehen. Damit Wicket weiß, welcher String angezeigt werden soll, muss im Attribut `key` der Schlüssel eingegeben werden, der den String eindeutig identifiziert. Zum Beispiel <wicket:message key ="name"/>. Wicket würde den Tag nun entweder durch *Username* oder *Benutzername* ersetzen.

**Lokalisierung über HTML-Templates:** Wenn Webseiten, die vorwiegend Text enthalten, lokalisiert werden sollen, dann ist obige Methode meist zu aufwendig. Daher kann man in Wicket verschiedensprachige HTML-Templates anlegen. Ebenso wie die Property-Dateien können auch die HTML-Templates mit Sprach-Kürzeln versehen werden. Für eine Login-Seite `LoginPage.java` könnten z. B. folgende Templates angelegt werden: `LoginPage.html`, `LoginPage_de.html`, `LoginPage_es.html` usw..

Die Möglichkeit, mit der Klasse `StringResourceModel` lokalisierte Strings programmatisch einzulesen, wurde in Abschnitt 3.2.4 bereits kurz angeschnitten.

### Integration mit anderen Frameworks

Obwohl Wicket schon eine beachtliche Anzahl an Komponenten und Funktionen zur Verfügung stellt, gibt es die Möglichkeit, weitere Frameworks und Bibliotheken einzubinden. Folgende Liste zeigt nur eine Auswahl an Frameworks und Komponenten.

**Velocity** ist eine Java-basierte Templating-engine. Sie bietet ähnliche Funktionen wie JSP-Seiten. Neben dem HTML-Code kann ein Velocity-Template auch Code enthalten, der zur Laufzeit ausgewertet wird. Um Velocity in Wicket verwenden zu können, wird ein eigenes Paket benötigt, welches in Wicket 1.3 bereits enthalten ist. Für frühere Versionen kann das Paket auf SourceForge[11] heruntergeladen werden.

**Rome** ist eine Bibliothek zum Erzeugen von Atom- und RSS-Feeds. Das Projekt *wicketstuff-rome* enthält eine Klasse `FeedPage`, die anstelle der Klasse `WebPage` verwendet werden kann. `FeedPage` stellt Methoden bereit, um News-Feeds zu erzeugen und auf einer Webseite anzuzeigen. Das nötige Jar-Paket kann auf der Wicket-Stuff-Seite[12] heruntergeladen werden.

**Groovy** [13] ist eine dynamische, Java-ähnliche Programmiersprache, die direkt in der Java-Virtuellen-Maschine ausgeführt werden kann. Das Projekt *wicket-groovy* bietet u. a. die Möglichkeit, Groovy-Dateien in Wicket zu laden. Weitere Informationen sind auf der Wicket-Stuff-Seite zu finden.

**Portlets** sind auch bekannt als JSR-168[14]. Ursprünglich war die Portlet-Funktionalität in der Kern-Bibliothek von Wicket enthalten, wurde aber dann als eigenständiges Unterprojekt ausgelagert. Weitere Infos sind auf Wicket-Stuff zu finden.

**Scriptaculous** ist eine JavaScript-Bibliothek. In Verbindung mit Wicket können u. a. relativ einfach Seiten mit Drag&Drop-Funktionalität implementiert werden. So steht z. B. eine `SortableListView`-Komponente zur Verfügung, die per Drag&Drop sortiert werden kann. Eine Projektbeschreibung befindet sich auf der Wicket-Stuff-Seite.

**Dojo** ist ebenfalls eine JavaScript-Bibliothek und steht derzeit in Version 1.3 auf der Wicket-Stuff-Seite zum Download zur Verfügung. Sie bietet eine Vielzahl an Wicket-Komponenten wie z. B. Drag&Drop-Komponenten, Date/Time-Picker, Tooltips, Sprechblasen, u. v. m.

[11] `http://sourceforge.net/project/showfiles.php?group_id=134391`
[12] `http://wicketstuff.org/confluence/display/STUFFWIKI/Wiki`
[13] `http://groovy.codehaus.org/`
[14] `http://www.jcp.org/en/jsr/detail?id=168`

**Spring** [15] ist ein Java-Application-Framework und zählt zur Kategorie der Meta-Frameworks und baut auf die Prinzipien *inversion of control* (IoC) und *dependency injection* (DI) auf. In Wicket 1.3 ist bereits ein Paket enthalten, um die Funktionalität von Spring zu integrieren.

**JMX** [16] Mit der Java Management Extensions (JMX) lassen sich Java Anwendungen überwachen und verwalten. Wicket stellt bereits ein Paket zur Verfügung, um verschiedene Einstellungen wie z. B. ApplicationSettings, PageSettings, usw. verwalten zu können.

### 3.2.5 Fazit

Wie in diesem Abschnitt gezeigt wurde, ist es bei der Auswahl eines Webframeworks wichtig, dass man sich zunächst für eine Kategorie entscheidet. Erst dann ist es sinnvoll, ausgewählte Webframeworks zu evaluieren. Für die Implementierung des Online-Wörterbuchs wurde das Webframework Wicket ausgewählt. Folgende Gründe waren ausschlaggebend: Im Gegensatz zu den anderen oben genannten Webframeworks benötigt Wicket keine zusätzliche XML-Datei, die z. B. Actions oder Navigationsregeln abbildet. Somit muss der Entwickler nur mehr die HTML-Datei und die Java-Klassen synchron halten. Ein weiterer Punkt ist, dass HTML-Dateien nur durch eine Hand voll Wicket-spezifischer HTML-Tags manipuliert werden. Dadurch können die HTML-Dateien durch jeden beliebigen HTML-Editor bearbeitet werden. Außerdem können Entwickler und Designer beinahe unabhängig voneinander arbeiten. Ein wichtiger Vorteil gegenüber den anderen Frameworks ist, dass Komponenten in Wicket richtige Java-Objekte sind und somit Kapselung, Vererbung und Events unterstützen. Des weiteren lassen sich sehr einfach wiederverwendbare Komponenten erzeugen, die sowohl den Java-Code als auch den HTML-Code kapseln.

## 3.3 Phonetische Suche

Dieser Abschnitt beschäftigt sich mit Algorithmen, die zur phonetischen Suche, also zur Suche ähnlich klingender Wörter, eingesetzt werden können.

[15] `http://www.springframework.org/`
[16] `http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/`

### 3.3.1 Soundex

Der Name *Soundex* ist eine Kombination aus den Wörtern *Sound* und *Index*. Entwickelt wurde der Soundex-Algorithmus bereits 1918 von Robert C. Russell. Ursprünglich wurde der Algorithmus von der US-Army und für die demografische Forschung verwendet. Bei der Entwicklung machte sich Russell die sechs phonetischen Klassen der menschlichen Sprache zu Nutze, die sich anhand der Position der Zunge und der Lippen beim Sprechen bestimmen lassen [Rep02, CPS05].

#### Der Algorithmus

Der Soundex-Algorithmus erzeugt aus einer beliebigen Eingabe einen vierstelligen Code. Die erste Stelle ist immer ein Buchstabe, die restlichen Stellen sind Zahlen. Der Algorithmus arbeitet dabei nach folgendem Schema [Rep02]:

1. Alle Buchstaben werden in Großbuchstaben umgewandelt.
2. Der erste Buchstabe wird nicht verändert.
3. Die Buchstaben A, E, I, O, U, H, W, Y werden durch die Zahl 0 ersetzt.
4. Folgende Buchstaben werden durch die Nummer der Klasse ersetzt:

   1 = B, F, P, V

   2 = C, G, J, K, Q, S, X, Z

   3 = D, T

   4 = L

   5 = M, N

   6 = R
5. Entferne doppelte Zahlen, die direkt nebeneinander vorkommen (aus Schritt 4).
6. Entferne alle Nullen, die in Schritt 3 gesetzt wurden.
7. Wenn der String aus Schritt 5 kleiner als vier Zeichen ist, dann füge genügend Nullen an.
8. Gib die ersten vier Zeichen des Strings aus.

**Beispiel:** Der Name „Jackson“ wird als „J250“ codiert.

Jackson $\xrightarrow{1}$ JACKSON $\xrightarrow{2,3}$ J0CKS0N $\xrightarrow{4}$ J022205 $\xrightarrow{5}$ J0205 $\xrightarrow{6}$ J25 $\xrightarrow{7}$ J250

Source-Codes für die Sprachen C, Perl, JavaScript, Visual Basic und Java können auf den Seiten von [Rep02] und [N.N06a] heruntergeladen werden.

#### Probleme mit Homofonen

Bei bestimmten Wörtern kann es vorkommen, dass ähnlich klingende Wörter, sogenannte Homofone, unterschiedliche Codes haben. Ein Grund dafür sind Wörter mit stummen Buchstaben, z. B. das *p* in *Thompson* (T512) im Gegensatz zu *Thomson* (T525). Des weiteren können gleich gesprochene Wörter auch unterschiedliche Anfangsbuchstaben haben, wie z. B. *circa* (C620) und *zirka*(Z620) [Whi04].

#### Soundex und Dialekt

Der Soundex-Algorithmus ist für den oberösterreichischen Dialekt nur bedingt geeignet, da die oben genannten Probleme ebenfalls zum Tragen kommen. Bei den meisten Wörtern funktioniert er jedoch ziemlich gut. So werden die Wörter *heißt*, *hoast*, *hoasd* und *hast* jeweils in „H320“ codiert. Das resultiert daraus, dass zum Einen die Selbstlaute entfernt werden und zum Anderen die Buchstaben `t` und `d` in der selben Klasse sind und `ß` in `s` umgewandelt wird.

Bei Wörtern mit unterschiedlichen Anfangsbuchstaben wie z. B. *acht* und *ocht*, errechnet der Algorithmus die Codes „A230“ bzw. „O230“. Eine Suche nach einem der beiden Wörter würde also nicht erfolgreich verlaufen.

### 3.3.2 Double-Metaphone

Aufgrund der Probleme mit dem Soundex-Algorithmus entwickelte Lawrence Philips den *Metaphone*-Algorithmus, den er 1990 publizierte. Der Metaphone-Algorithmus wurde so gestaltet, dass nicht Buchstabe für Buchstabe in einen Code umgerechnet wird, sondern anhand von Silben. So erzeugt der Algorithmus den Code „STFN“ für die Wörter *Steven*, *Stephan* und *Stefan*. Da über die Jahre viele Fehler und Ausnahmen speziell für nicht englische Wörter bekannt wurden, musste der Algorithmus angepasst werden [Phi03].

#### Einblick in den Algorithmus

Im Jahr 2000 wurde der Double-Metaphone-Algorithmus vorgestellt. Double-Metaphone erzeugt aus einer Eingabe einen meist vier Zeichen langen Code. Der Code wird aus nur zwölf Konsonanten gebildet. Den Namen Double-Metaphone hat der Algorithmus, weil er zwei Codes pro Eingabe berechnen kann. Zum Einen kann er die amerikanische Aussprache berechnen und zum Anderen eine native Aussprache. Für das spanische Wort *Cabrillo* würde der Algorithmus „Cabrillo“ im Amerikanischen und „Cabrijo“im Spanischen als Basis für die Berechnung heranziehen. Der aktuelle Double-Metaphone-Algorithmus kann mit Eigenheiten der Sprachen Englisch, Italienisch, Spanisch und Französisch sowie mit einigen slawischen und germanischen Sprachen umgehen. Die Java-Implementierung in [N.N06a] hat ca. eintausend Zeilen Quellcode. Alleine für den Buchstabe *C* werden über einhundert verschiedene Kontexte berücksichtigt [Phi03].

#### Double-Metaphone und Dialekt

Der Double-Metaphone-Algorithmus löst vor allem das Problem mit unterschiedlichen Anfangsbuchstaben. So werden z. B. *acht* und *ocht* zu „AKT“ codiert.

Ein weiterer Vorteil gegenüber dem Soundex-Algorithmus besteht darin, dass er nicht Zeichen für Zeichen codiert, sondern auch Zeichenketten erfassen kann. So werden z. B. *Scherz*, *Sherz* und *Cherz* zu „XRS“ codiert.

### 3.3.3 Fazit

Generell sind alle phonetischen Algorithmen ganz annehmbar, weil bei Wörtern im Dialekt oft nur Selbstlaute anders ausgesprochen werden und diese bei den oben genannten Algorithmen herausgefiltert werden, z. B. *heißt* und *hoast*.

Der Soundex-Algorithmus ist zwar der kürzeste Algorithmus im Hinblick auf die Codelänge, da stumme Buchstaben und gleich klingende Anfangsbuchstaben jedoch nicht berücksichtigt werden, ist der Algorithmus nur bedingt für das Online-Wörterbuch geeignet. Besonders bei umgangssprachlicher Aussprache ergeben sich oft andere Anfangsbuchstaben. Das deutsche Wort *Test* beispielsweise wird im Oberösterreichischen häufig wie *Dest* ausgesprochen. Solche Wörter würden bei einer Suche kein Ergebnis liefern.

Wie im letzten Abschnitt gezeigt wurde, verhält sich der Double-Metaphone-Algorithmus wesentlich besser im oberösterreichischen Dialekt. Aus diesem Grund wurde der Double-Metaphone-Algorithmus für die Implementierung des Systems herangezogen.

## 3.4 Sortieren ähnlicher Wörter

Um bei Suchergebnissen eine Reihung vornehmen zu können, braucht man ein Maß, das angibt, inwieweit der Suchbegriff mit den Ergebnissen übereinstimmt. Die folgenden Algorithmen können aus zwei Strings eine Distanz errechnen. Grundsätzlich gilt: Je kleiner die Distanz ist, desto ähnlicher sind sich die Strings. Eine Distanz von 0 würde z. B. bedeuten, dass die Strings exakt gleich sind.

### 3.4.1 Hamming-Distanz

Bei der Hamming-Distanz, benannt nach Richard Hamming, werden Strings nicht anhand der phonetischen Kodierung, sondern der Schreibweise verglichen. Die Hemming-Distanz $hd(x, y)$ gibt an, an wie vielen Stellen sich zwei Wörter $x$ und $y$ der Länge $m$ unterscheiden [Röp05]. Anders ausgedrückt, wie viele Buchstaben in $x$ müssen ersetzt werden, um Wort $y$ zu erhalten. Mathematisch kann der Algorithmus folgendermaßen formuliert werden:

$$hd(x, y) = \big|x[i] \neq y[i]\big| \qquad wobei \quad 1 \leq i \leq m \quad und \quad |x| = |y| = m$$

**Beispiel:** $hd(x, y) = 1$ mit $x = Hallo$ und $y = Hello$.

#### Anwendungsbereiche

Hauptsächlich wird der Algorithmus in der Nachrichtentechnik für die Fehlererkennung bzw. Fehlerkorrektur verwendet [Bha95]. Häufig findet der Algorithmus auch in der Bioinformatik Anwendung, z. B. um gleiche DNA-Sequenzen zu finden [Col07].

Für das Online-Wörterbuch ist dieser Algorithmus jedoch nicht zu gebrauchen, da nur gleich lange Wörter zur Berechnung der Distanz herangezogen werden können.

### 3.4.2 Levenshtein-Distanz

Im Gegensatz zur Hamming-Distanz werden bei der Levenshtein-Distanz zusätzlich zu Ersetzungsoperationen auch Einfüge- und Löschoperationen berücksichtigt. Das heißt, die Levenshtein-Distanz $lev(x, y)$ ergibt sich aus der kleinsten Anzahl dieser Operationen, die nötig sind, um Wort $x$ in $y$ zu überführen.

#### Der Algorithmus

Zur Berechnung der Levenshtein-Distanz wird eine Matrix aufgebaut. Durch den kürzesten Weg durch die Matrix ergeben sich dann die Operationen, die nötig sind, um aus Wort $x$, Wort $y$ zu erzeugen. Im Beispiel weiter unten wird der Weg durch die Matrix genauer beschrieben.

Definition der Levenshtein-Matrix nach [Bec07]:

$$LEV[i,j] := lev(x[1 \ldots i], y[1 \ldots j]) \quad mit \quad 0 \leq i \leq m, 0 \leq j \leq n$$

Die Werte der Matrix $LEV[i,j]$ können mit folgenden rekursiven Formeln berechnet werden:

- $LEV[0,j] = j \quad für \quad 0 \leq j \leq n, \quad LEV[i,0] = i \quad für \quad 0 \leq i \leq m$
- $LEV[i,j] = min \begin{cases} LEV[i-1,j]+1 \\ LEV[i,j-1]+1 \\ LEV[i-1,j-1]+\delta(x[i],y[j]) \end{cases}$
- $\delta(a,b) = \begin{cases} 0 & falls \quad a = b \\ 1 & sonst \end{cases}$

**Beispiel 1:** In Abbildung 3.5 ist die berechnete Matrix für $lev(x, y) = 4$ zu sehen, wobei $x = Fiction$ und $y = Dictionary$ ist. Die Levenshtein-Distanz kann man in der Matrix an der Stelle $LEV[i,j]$, das ist die Zahl in der rechten unteren Ecke, ablesen. In diesem Fall ist die Distanz 4, d. h. es sind maximal vier Einfüge-, Lösch- oder Ersetzungsoperationen nötig, um aus $x$ $y$ zu erhalten. Des weiteren ist in Abbildung 3.5 der kürzeste Weg durch die Matrix eingezeichnet. Der Weg beginnt immer im Punkt

$LEV[0,0]$ und endet in $LEV[i,j]$. Diagonale Kanten stellen in der Matrix eine Ersetzungsoperation oder das Auslassen einer Operation dar. Horizontale Kanten stellen Einfügeoperationen dar, vertikale Kanten hingegen Löschoperationen.

Eine Transformation von „Fiction“ in „Dictionary“ sieht folgendermaßen aus, wobei $S$ andeutet, dass eine Substitution erforderlich ist, $G$, dass der Buchstabe gleich bleibt und $E$, dass ein Buchstabe eingefügt werden muss. Eine Löschoperation ist in diesem Beispiel nicht erforderlich.

$$Fiction \rightarrow \overset{S}{D}iction \rightarrow D\overset{G}{i}ction \ \ldots\ Dictio\overset{G}{n} \rightarrow Diction\overset{E}{a} \ \ldots\ Dictionar\overset{E}{y}$$

**Beispiel 2:** In Abbildung 3.6 ist die Levenshtein-Matrix für die Transformation von „Dictionary“ in „Fiction“ zu sehen. Grundsätzlich verläuft die Transformation sehr ähnlich. Es wird lediglich anstatt der Einfügeoperation ($E$) eine Löschoperation ($L$) durchgeführt.

$$\begin{aligned} Dictionary \rightarrow \overset{S}{F}ictionary \rightarrow F\overset{G}{i}ctionary \ \ldots\ Fictio\overset{G}{n}ary \\ \rightarrow Fiction\overset{L}{a}ry \ \rightarrow Fiction\overset{L}{r}y \ \rightarrow Fiction\overset{L}{y} \rightarrow Fiction \end{aligned}$$

Die Abbildungen 3.5 und 3.6 wurden auf der Webseite von [Gee07] erstellt. Auf dieser Seite ist es möglich, für beliebige Wörter die Levenshtein-Matrix berechnen zu lassen.

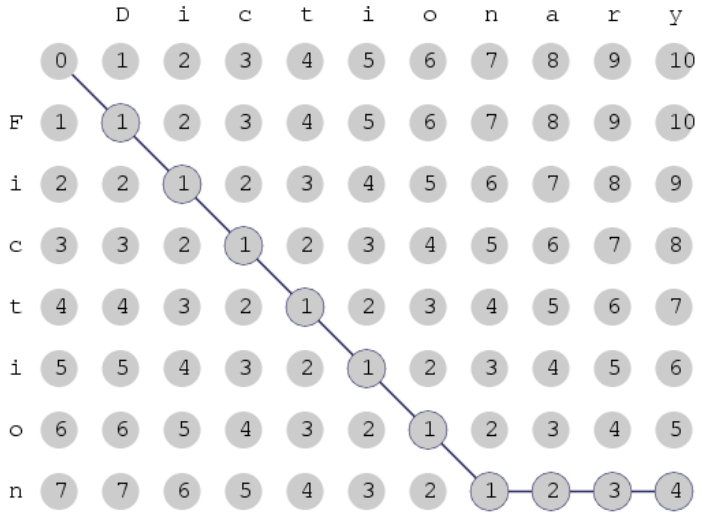



**Abbildung 3.5:** Levenshtein-Distanz für Fiction→Dictionary

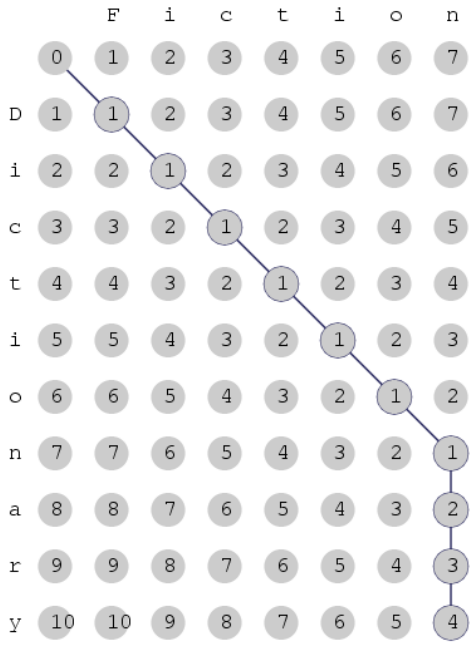



**Abbildung 3.6:** Levenshtein-Distanz für Dictionary→Fiction

**Komplexität:** Die naive Implementierung des Levenshtein-Distanz-Algorithmus hat eine Laufzeitkomplexität von $O(mn)$ und eine Speicherkomplexität von ebenfalls $O(mn)$, wobei $m$ und $n$ die Wortlängen darstellen. Dan Hirschberg [Hir97] überarbeitete den Algorithmus, sodass eine lineare Speicherkomplexität von $O(min\{m, n\})$ möglich ist.

### 3.4.3 Damerau-Levenshtein-Distanz

Die Damerau-Levenshtein-Distanz $dlev(x, y)$ ist eine Erweiterung der Levenshtein-Distanz um eine weitere Operation. Neben der Einfüge-, Lösch- und Ersetzungsoperation gibt es auch eine Vertauschungsoperation. Dies soll das einfachere Finden von Tippfehlern ermöglichen [Hyy03].

**Beispiel:** Für die Wörter $x = Test$ und $y = Tset$ ist $dlev(x, y) = 1$, während $lev(x, y) = 2$ sein würde. Da der Damerau-Levenshtein-Algorithmus die Vertauschungsoperation als eine einzelne Operation betrachtet, der Levenshtein-Algorithmus jedoch als zwei Operationen, gilt immer: $dlev(x, y) \leq lev(x, y)$ [Bec07].

**Der Algorithmus**

Definition der Damerau-Levenshtein-Matrix nach [CjK03]:

- $DLEV[i,j] := dlev(x[1 \ldots i], y[1 \ldots j]) \quad mit \quad 0 \leq i \leq m, 0 \leq j \leq n$
- $DLEV[0,j] = j \quad für \quad 0 \leq j \leq n, \quad DLEV[i,0] = i \quad für \quad 0 \leq i \leq m$
- $DLEV[i,j] = min \begin{cases} DLEV[i-1,j]+1 \\ DLEV[i,j-1]+1 \\ DLEV[i-1,j-1]+\delta(x[i],y[j]) \\ DLEV[i-2,j-2]+\delta(x[i-1],y[j])+\delta(x[i],y[j-1])+1 \end{cases}$
- $\delta(a,b) = \begin{cases} 0 & falls \quad a=b \\ 1 & sonst \end{cases}$

### 3.4.4 Fazit

Die Hamming Distanz scheidet für das System schon von vorne herein aus, da die beiden zu vergleichenden Strings immer gleich lang sein müssen. Bei einer Wörterbuch-Anwendung ist dies eher ein Einzelfall. Natürlich könnte man den Algorithmus so implementieren, dass die Anzahl der zu vergleichenden Zeichen der Länge des kürzesten Wortes entspricht. Mehr als eine Notlösung wäre das jedoch nicht.

Die Levenshtein-Distanz ist für die Reihung von Wörtern bestens geeignet. Da die Damerau-Levenshtein-Distanz jedoch zusätzlich Tippfehler in Betracht zieht, wurde dieser Algorithmus im Online-Wörterbuch verwendet.

Beim Online-Wörterbuch werden die Vorschläge nicht nur durch die String-Distanz ermittelt. Zuerst wird per phonetischem Algorithmus eine grobe Vorauswahl getroffen, danach werden die Ergebnisse per Damerau-Levenshtein-Distanz sortiert. Die besten Ergebnisse werden schließlich angezeigt. Im nächsten Kapitel wird der Suchvorgang detailliert behandelt.

## 3.5 Persistenzschicht Hibernate

Dieser Abschnitt beschäftigt sich mit Hibernate und den grundlegenden Begriffen, die bei Object-Relational-Mapping-Frameworks vorkommen. Neben Hibernate sind auch

TopLink von Oracle und das Java Persistence API (JPA) häufig verwendete Technologien. Hibernate hat sich in letzter Zeit jedoch als der De-facto-Standard in der Java Welt etabliert [Raj07a]. Da das System den gesamten Datenbankzugriff über Hibernate abwickelt, werden die verschiedenen Funktionen ausführlich behandelt.

### 3.5.1 Begriff Object-Relational-Mapping

Object-Relational-Mapping, manchmal auch ORM oder O/R-Mapping genannt, hat seine Wurzeln in den 90er Jahren und versucht das sogenannte „object-relational paradigm mismatch"-Problem zu lösen. Dieses Problem besteht, weil das Objekt-orientierte Paradigma auf den Prinzipien der Softwareentwicklung beruht und das relationale Paradigma hingegen rein auf mathematischen Prinzipien. Da sich diese Paradigmen unterscheiden, können die Technologien nicht ohne weiteres zusammenarbeiten. Beim Datenzugriff kann man die Unterschiede sehr gut sehen. Bei Objekten traversiert man über Beziehungen zu den Daten, die man benötigt. Bei Datenbanken hingegen muss man Tabellen verknüpfen und Datenreihen selektieren [Amb07]. ORM versucht nun Java-Objekte automatisiert auf einer Datenbank abzubilden, sodass diese möglichst einfach geladen bzw. gespeichert werden können.

### 3.5.2 Metadaten

Um ORM bzw. Hibernate verwenden zu können, werden Metadaten benötigt, die eine Abbildung (Mapping) erst ermöglichen. Hibernate muss also genau wissen, wie Klassen geladen und gespeichert werden.

Im folgenden Beispiel wird gezeigt, wie eine Klasse `Message.java` auf eine Tabelle abgebildet werden kann. Dabei soll dieses Beispiel nur einen Einblick in die grundlegende Struktur einer Mapping-Datei geben. Detaillierte Beschreibungen sind in [BK07] und [PH06] zu finden.

Wie im Listing 3.16 zu sehen ist, besitzt die Klasse drei Attribute. Damit eine Klasse überhaupt abgebildet werden kann, muss sie ein `id`-Attribut besitzen. Dieses Attribut wird auf den Primärschlüssel der Tabelle abgebildet. Das Attribut `text` kann einen Nachrichtentext als String enthalten. `nextMessage` hingegen referenziert auf ein

weiteres `Message`-Objekt. Aus Platzgründen sind in dem Listing keine Get- und Set-Methoden zu finden, diese sind jedoch unbedingt nötig, damit Hibernate auf die Attribute zugreifen kann [BK07].

```
1 public class Message {
2    private Long id;      //Identifier attribute
3    private String text; //Message text
4    private Message nextMessage;  //Reference to another Message instance
5
6    Message() {}
7    public Message(String text) {
8       this.text = text;
9    }
10
11   /* get- and set-methods */
12 }
```

**Listing 3.16:** Message.java (aus [BK07])

Da die Klasse `Message` keine Hibernate-spezifischen Methoden enthält, kann sie unabhängig von Hibernate in jedem Kontext verwendet werden. Um diese Klasse mit Hibernate benutzen zu können, benötigt man noch ein *XML mapping document*, welches die Abbildung erst ermöglicht.

Listing 3.17 zeigt die dazugehörige Mapping-Datei zur Klasse `Message`. Durch diese Datei weiß Hibernate, dass die Klasse auf die Tabelle `MESSAGE`, dass das `id`-Attribut auf die Spalte `MESSAGE_ID` und dass das Attribut `text` auf die Spalte `MESSAGE_TEXT` abgebildet wird. Außerdem wird das Attribut `nextMessage` über eine n:1-Beziehung abgebildet, wobei Hibernate automatisch einen Fremdschlüssel `FK_NEXT_MESSAGE` zur Datenbank hinzufügt. Mit diesem Dokument hat Hibernate nun genug Informationen, um automatisch SQL-Anweisungen zu erzeugen, die ein Message-Objekt einfügen, updaten, löschen und laden können [BK07].

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3    "-//Hibernate/Hibernate Mapping DTD//EN"
4    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5 <hibernate-mapping>
6    <class
7          name="Message"
8          table="MESSAGES">
```

```
9         <id
10            name="id"
11            column="MESSAGE_ID">
12            <generator class="increment"/>
13        </id>
14        <property
15            name="text"
16            column="MESSAGE_TEXT"/>
17        <many-to-one
18            name="nextMessage"
19            cascade="all"
20            column="NEXT_MESSAGE_ID"
21            foreign-key="FK_NEXT_MESSAGE"/>
22    </class>
23 </hibernate-mapping>
```

**Listing 3.17:** Message.hbm.xml (aus [BK07])

### 3.5.3 Konfiguration

Im vorhergehenden Abschnitt wurde durch die Mapping-Datei eine Verbindung zwischen einer Java-Klasse und der dazugehörigen Tabelle geschaffen. Hibernate hat dadurch jedoch noch keine Information darüber, welche Datenbank tatsächlich benutzt wird und wie diese angebunden ist. Dieser Abschnitt beschäftigt sich daher mit der Konfiguration von Hibernate.

Hibernate kann auf zwei verschiedene Arten konfiguriert werden. Einerseits über eine Java-Properties-Datei und andererseits über eine XML-Datei (`hibernate.cfg.xml`). Da die Implementierung die XML-Variante verwendet, wird hier auch nur auf diese eingegangen.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <property name="connection.username">myUsername</property>
      <property name="connection.password">myPassword</property>
      <property name="connection.url">
         jdbc:mysql://localhost/db
      </property>
      <property name="connection.driver_class">
         com.mysql.jdbc.Driver
      </property>
      <property name="dialect">
         org.hibernate.dialect.MySQLDialect
      </property>

      <mapping resource="Messsage.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

**Listing 3.18:** hibernate.cfg.xml (nach [PH06])

Wie in Listing 3.18 zu sehen ist, befinden sich alle wichtigen Konfigurationsdaten im Tag `session-factory`. Über die ersten `property`-Tags werden zunächst Benutzername, Passwort und die URL zur Datenbank festgelegt. Da in diesem Beispiel eine JDBC-Verbindung aufgebaut wird, muss auch eine JDBC-konforme URL und die JDBC-Treiberklasse angegeben werden. Besonders wichtig ist das `dialect`-Property. Durch diese Angabe weiß Hibernate, welche Datenbank verwendet wird. Außerdem wird diese Information dazu verwendet, um HQL-Anweisungen in die Datenbank-spezifische Abfragesprachen zu konvertieren. Die HQL wird weiter unten in Abschnitt 3.5.5 noch genauer behandelt. Weitere Beispiele und Infos zur Konfiguration sind in [N.N07a] zu finden.

### 3.5.4 Laden und Speichern von Objekten

Wurde Hibernate richtig konfiguriert, dann können Java-Objekte sehr einfach geladen und gespeichert werden. Um auf die Datenbank zugreifen zu können, muss zuerst eine

`Session` angelegt werden. Eine Hibernate-`Session` überwacht die Abfragen und Transaktionen. Außerdem enthält sie grundlegende SQL-Anweisungen, die für die Synchronisation der Objekte nötig sind [BK07]. In Listing 3.19 wird das `Session`-Objekt von der Klasse `HibernateUtil` zur Verfügung gestellt. Diese Klasse wird in der Hibernate-Dokumentation [N.N07a] näher beschrieben.

```
1 private void createAndStoreMessage(String text, Message nextMessage) {
2
3    Session session = HibernateUtil.getSessionFactory()
4        .getCurrentSession();
5   session.beginTransaction();
6
7    Message msg = new Message();
8    msg.setText(text);
9    msg.setNextMessage(nextMessage);
10
11  session.save(msg);
12  session.getTransaction().commit();
13 }
```

**Listing 3.19:** Erzeugen und Speichern eines Objekts

Wenn Zeile 11 in Listing 3.19 aufgerufen wird, dann erzeugt Hibernate eine eindeutige ID für das `Message`-Objekt, führt selbständig die nötigen Insert-Anweisungen aus und speichert das Objekt in der Datenbank. Mit der Methode `commit()` wird die Transaktion letztendlich abgeschlossen. Analog funktioniert das Laden, Aktualisieren und Löschen.

### 3.5.5 Hibernate Query Language (HQL)

Im vorhergehenden Abschnitt wurde gezeigt, wie man Objekte über Methoden von Hibernate laden und speichern kann. Eine weitere Möglichkeit, auf Objekte zuzugreifen, besteht darin, die Hibernate Query Language (HQL) zu verwenden. Da die HQL sehr ähnlich zu SQL ist, wird hier exemplarisch nur ein Beispiel angeführt, jedoch nicht näher auf die Syntax eingegangen. Außerdem werden die Vorteile der HQL erläutert. Für detailliertere Informationen wird auf [N.N07a] und [PH06] verwiesen.

**Beispiel:** Im Listing 3.20 wird ein `Query`-Objekt mit einer HQL-Abfrage als Parameter erzeugt. Wenn die Abfrage in Zeile 3 ausgeführt wird, dann liefert das `Query`-Objekt eine Liste mit jenen `Message`-Objekten deren `text`-Attribut das Wort *Hello* enthält.

```
1 Query q = session.createQuery("from Message where text = ? ");
2 q.setParameter(0, "Hello");
3 List results = q.list();
```

**Listing 3.20:** HQL-Query

**Vorteile:** HQL ist sehr ähnlich zur standardisierten Abfragesprache SQL. Nun stellt man sich vielleicht die Frage, wofür eine weitere Sprache nötig ist. Ein Hauptargument für HQL ist die Portabilität. Da die verschiedenen Datenbank-Hersteller oft nicht-standardisierte Funktionen bereitstellen, kann die Portierung von SQL-Anweisungen auf eine andere Datenbank sehr mühsam sein. Wird jedoch HQL verwendet, dann kann Hibernate die Anweisungen in viele verschiedene Datenbankdialekte übersetzen. Ein weiterer Grund für HQL ist, dass SQL für relationale Tabellen entwickelt wurde, HQL hingegen wurde speziell für die Abfrage von Objekt-Graphen entwickelt [PH06].

### 3.5.6 Hibernates Performanz

Ein wesentliches Kriterium für Hibernate ist die Performanz, welche hauptsächlich durch drei Konzepte erhöht wird, nämlich spezifische Datenbank-Adapter, First-Level-Cache und Lazy-Loading. Hibernate hat ein Detailwissen über die gängigsten Datenbanken, so ist es möglich, die Zugriffe Datenbank-spezifisch zu optimieren. Der Anwender muss lediglich in der Konfiguration den Typ(Dialekt) der Datenbank angeben.

Hibernate benutzt von Haus aus einen First-Level-Cache, der Daten pro Session zwischenspeichert. Erst wenn eine Transaktion abgeschlossen ist oder der Cache geleert wird, werden die Objekte in die Datenbank geschrieben.

Durch Lazy-Loading steht Hibernate ein Mechanismus zur Verfügung, um Daten erst dann zu laden, wenn diese tatsächlich benötigt werden. Werden weitere Daten benötigt, dann können diese zu einem späteren Zeitpunkt nachgeladen werden [Roo06]. Ab Hibernate 3 ist Lazy-Loading von vornherein eingeschaltet.

Außerdem bietet Hibernate vier verschiedene Second-Level-Cache-Implementierungen

an. Im Unterschied zum First-Level-Cache, der Daten nur für eine Session zwischenspeichert, kann der Second-Level-Cache Daten anwendungsweit zwischenspeichern [PH06].

Laut Rook [Roo06] lassen sich pro Anwendung ca. 30% Programmcode einsparen. Somit erhöht sich auch die Produktivität und die Fehlerwahrscheinlichkeit wird geringer.

## 3.6 Java-Applets

Dieser Abschnitt gibt einen kurzen Überblick über Java-Applets, da ein solches im Online-Wörterbuch zur Aufnahme von Wörtern verwendet wird. Anstatt eines Applets hätte aber auch eine andere Technologie wie z. B. Adobe Flash verwendet werden können. Da die gesamte Web-Anwendung in Java implementiert wurde und in Applets die vollständige Java-API zur Verfügung steht, wurde auf ein Java-Applet zurückgegriffen.

Java-Applets oder kurz Applets sind kleine Programme, die über das Internet übertragen werden und in einem Browser angezeigt werden können. Im Gegensatz zu Servlets, die auf der Server-Seite ausgeführt werden, müssen Applets von der Java-Virtuellen-Maschine (JVM) auf dem Client ausgeführt werden.

### 3.6.1 Struktur

Jedes Applet muss von der Klasse `java.Applet` abgeleitet werden. Die Klasse Applet stellt vier Methoden, nämlich `init()`, `start()`, `stop()` und `destroy()`, zur Verfügung, die für die Ausführung notwendig sind. Eine weitere Methode `paint()` wird von der Klasse `Component` vererbt [Sch05a].

```
1 public class AppletSkeleton extends Applet {
2    // Called first.
3    public void init() {
4       // initialization
5    }
6    /* Called second, after init(). Also called whenever
7    the applet is restarted. */
8    public void start() {
9       // start or resume execution
10   }
11   // Called when the applet is stopped.
```

```
12    public void stop() {
13       // suspends execution
14    }
15    /* Called when applet is terminated. This is the last
16    method executed. */
17    public void destroy() {
18       // perform shutdown activities
19    }
20    // Called when an applet's window must be restored.
21    public void paint(Graphics g) {
22       // redisplay contents of window
23    }
24 }
```

**Listing 3.21:** AppletSkeleton.java (aus [Sch05a])

Im Listing 3.21 ist die Struktur eines Applets dargestellt. Außerdem sind die oben genannten Methoden zu sehen. Da die Basisklassen bereits diese Methoden implementieren, müssen nur diejenigen überladen werden, die auch tatsächlich verwendet werden. Bei der Ausführung wird zuerst die Methode `init()` aufgerufen. In dieser werden z. B. Variablen initialisiert. Als zweites wird die Methode start() aufgerufen. Sie kann auch mehrmals im Lebenszyklus aufgerufen werden, und zwar dann, wenn ein Benutzer zu einer vorher besuchten Webseite zurückkehrt. Die Methode `stop()` wird aufgerufen, wenn der Benutzer die Webseite verlässt. Sie ist notwendig, um alle gestarteten Threads zu beenden und das Applet in einen sicheren Zustand zu versetzen. Nach Beendigung des Applets kann es jedoch jederzeit wieder neu gestartet werden. Wenn das Applet definitiv nicht mehr benötigt wird, dann wird die Methode `destroy()` aufgerufen [Sch05a]. Die Methode paint() wird nur dann aufgerufen, wenn die Benutzerschnittstelle verändert wurde und somit neu gezeichnet werden muss.

Damit ein Applet im Browser verwendet werden kann, muss es in den HTML-Code eingebettet werden. Dies geschieht mit einem speziellen Tag. Listing 3.22 zeigt ein Beispiel, wie ein Applet eingebunden werden kann. Im Applet-Tag können verschiedene Parameter gesetzt werden. Der Parameter `archive` gibt z. B. an, wie der Name der JAR-Datei lautet, in der sich die Applet-Klassen befinden. Der Parameter `code` gibt hingegen an, wie die Klasse heißt. Außerdem können noch Höhe und Breite des Applets eingestellt werden.

```
<applet archive="TestApplet.jar" code="TestApplet.class"
    width="400" height="200">
</applet>
```

**Listing 3.22:** Applet-Tag (aus [Sur07])

### 3.6.2 Applets signieren

Applets werden oft von unbekannten Webseiten heruntergeladen und im Client ausgeführt. Dies birgt Gefahren in sich, da eventuell böswilliger Code ausgeführt werden könnte. Um dies zu verhindern, haben Applets nur eingeschränkte Rechte, so können sie z. B. nicht auf die lokale Festplatte oder auf ein Mikrofon zugreifen.

Für das Online-Wörterbuch muss aber ein Zugriffsrecht auf das Mikrofon bestehen. Durch ein signiertes Applet wird das gewährleistet.

Für das Signieren von Applets sind drei Programme nötig: `keytool`, `jar` und `jarsigner`. Alle drei Programme sind im JDK (Java Development Kit) inkludiert. Das unten beschriebene Verfahren basiert auf dem von [Sur07].

1. Als erstes wird mit folgendem Befehl das Zertifikat `certificate.store` angelegt. Es enthält Daten wie z. B. Name und Organisation des Ausstellers.

   ```
   keytool -genkey -keystore certificate.store -alias certificate
   ```

2. Um ein Applet signieren zu können, muss es in ein JAR-Archiv gepackt werden. Wichtig dabei ist, dass alle Class-Dateien angegeben werden, die das Applet benötigt. In diesem Beispiel trägt das Applet den Namen `TestApplet`:

   ```
   jar cvf TestApplet.jar TestApplet.class
   ```

3. Im nächsten Schritt wird das Archiv signiert. Das kann mit folgendem Befehl erreicht werden:

   ```
   jarsigner -keystore certificate.store TestApplet.jar certificate
   ```

   Beim Signieren wird man nach einem Passwort gefragt. Hier ist jenes einzugeben, das auch beim Anlegen des Zertifikats verwendet wurde.

## 3.7 PDF-Erzeugen mit iText

iText wurde von Bruno Lowagie und Paulo Soares entwickelt und ist eine Java-Bibliothek zum programmatischen Erzeugen von PDF-Dokumenten. Derzeit ist sie in Version 2.0.4 verfügbar. Als Lizenzen stehen MPL und LGPL zur Auswahl. Neben PDF kann iText auch RTF-, HTML- und XML-Dokumente erzeugen. Eine .Net-Portierung, genannt iTextSharp, steht ebenfalls zum Download bereit [LS07].

Neben den bereits genannten Features bietet iText folgende Möglichkeiten:

- PDF-Dokument an einen Webbrowser schicken
- Dynamisches Erzeugen von PDF-Dokumente aus XML-Dateien oder Datenbanken
- PDF mit interaktiven Elementen versehen (z. B. durch Java Script)
- Hinzufügen von Lesezeichen, Seitennummern, Wasserzeichen, etc.
- Teilen, Zusammenfügen und Ändern von PDF-Seiten
- Automatisches Ausfüllen von PDF-Formularen
- Digitale Signaturen zu einem PDF-Dokument hinzufügen

### 3.7.1 Allgemeine Verwendung

Jedes neue Dokument in iText wird in fünf Schritten erzeugt:

1. Ein Exemplar der Klasse `Document` anlegen.
2. Ein Exemplar der Klasse `Writer` (z. B. `PdfWriter`) erzeugen.
3. Das `Document`-Objekt öffnen.
4. Inhalt zum Dokument hinzufügen
5. Das `Document`-Objekt schließen.

Wie man in den fünf Schritten erkennen kann, spielen die `Document`-Klasse und die `Writer`-Klassen eine spezielle Rolle. In den folgenden beiden Absätzen werden sie daher näher behandelt.

**Das `Document`-Objekt** ist das Wurzelelement in iText. Alle weiteren iText-Objekte werden über die Methode `add()` zu diesem Objekt hinzugefügt.

Durch das `Document`-Objekt kann auch das Seitenformat festgelegt werden. Da iText eine europäische Entwicklung ist, wurde DIN A4 als Standardformat gewählt. Es steht aber eine große Auswahl an Standardformaten zur Verfügung. Diese Formate sind in der Klasse `PageSize` zu finden. Mit `document.setPageSize(PageSize.A3)` wird z. B. auf das DIN A3 Format umgestellt. Man ist aber nicht unbedingt auf diese Klasse angewiesen. Es ist möglich, ein Dokument mit einem Format von bis zu 200 mal 200 Zoll zu erstellen.

Normalerweise ist das Dokument im Hochformat. Um ein Dokument im Querformat zu erhalten, kann die `rotate()`-Methode verwendet werden,
z. B. `document.setPageSize(PageSize.A4.rotate())`.

Des Weiteren können über das Document-Objekt die Seitenränder eingestellt werden. Es besteht auch die Möglichkeit, für gerade bzw. ungerade Seiten die Seitenränder alternierend zu spiegeln.

**Die `Writer`-Objekte** können nur über eine statische Methode (z. B. `PdfWriter.getInstance()`) erzeugt werden. Diese Methode benötigt zwei Parameter. Zum Einen das `Document`-Objekt und zum Anderen einen `OutputStream`. Um eine Dokument in eine Datei zu schreiben, empfiehlt es sich, ein `FileOutputStream` zu verwenden. Soll das Dokument z. B. über einen Webserver verschickt werden, dann eignet sich ein `ByteArrayOutputStream` am besten.

Die `Writer`-Objekte sind als Listener implementiert, d. h. es können mehrere verschiedene `Writer` an ein und dasselbe `Document`-Objekt gebunden werden. In Listing 3.23 wird z. B. in Zeile 9 ein Pdf-Writer und in Zeile 11 ein Rtf-Writer erzeugt. Während nun Text zum Dokument hinzugefügt wird (Zeile 16), wird auf beiden `Writern` gleichzeitig geschrieben. Als Ergebnis erhält man ein PDF- und ein RTF-Dokument mit demselben Text.

```
1 public class iTextHelloWorld {
2    public static void main(String[] args) {
3
4       // step 1: creation of a document-object
5       Document document = new Document();
```

```
6       try {
7           // step 2:
8           // we create writers that listens to the document
9           PdfWriter.getInstance(document,
10                  new FileOutputStream("HelloWorld.pdf"));
11          RtfWriter2.getInstance(document,
12                  new FileOutputStream("HelloWorld.rtf"));
13          // step 3: we open the document
14          document.open();
15          // step 4: we add a paragraph to the document
16          document.add(new Paragraph("Hello World"));
17
18      } catch (DocumentException de) {
19          System.err.println(de.getMessage());
20      } catch (IOException ioe) {
21          System.err.println(ioe.getMessage());
22      }
23      // step 5: we close the document
24      document.close();
25  }
26 }
```

**Listing 3.23:** iTextHelloWorld.java (nach [LS07])

**Anmerkung:** Obwohl iText verschiedene Dateiformate erzeugen kann, wird in den nächsten Abschnitten nur mehr auf PDF eingegangen, da bei der Implementierung des Online-Wörterbuches nur dieses Format verwendet wurde. Außerdem gibt es für einige Formate spezielle Komponenten, die in einem anderen Format nicht dargestellt werden können. Ein Exemplar der Klasse `PdfPTable` kann z. B. nicht von einem `RtfWriter` geschrieben werden.

### 3.7.2 Komponenten

Da nun der grundlegende Ablauf beim Erzeugen eines PDF-Dokuments geklärt ist, werden in diesem Abschnitt ausgewählte Komponenten und deren Verwendung vorgestellt.

### Basisobjekte

Jedes PDF-Dokument in iText besteht aus einer Komposition der unten vorgestellten Basisobjekte. Durch sie ist auch die Bildung von wiederverwendbaren Textstücken und die Erzeugung einer bestimmten Hierarchie möglich.

**Chunk:** Ein `Chunk` ist die kleinste Text-Einheit in iText. Hauptsächlich werden sie dazu verwendet, kleine Text-Stücke in ein Objekt zu kapseln. Dieses Textstück kann so z. B. speziell formatiert werden. Der gesamte Inhalt hat dabei die gleiche Formatierung.

**Phrase:** Einem `Phrase`-Objekt werden meist ein oder mehrere `Chunk`-Objekte hinzugefügt. Im Gegensatz zum `Chunk`-Objekt kann beim `Phrase`-Objekt ein Zeilenabstand festgelegt werden. Normalerweise ist ein 1,5-facher Zeilenabstand eingestellt. Im PDF-Dokument wird nach einem `Phrase`-Objekt keine neue Zeile angefangen.

**Paragraph:** Das `Paragraph`-Objekt ist im Grunde eine Liste von einem oder mehreren Elementen wie z. B. `Chunk`, `Phrase`, `List` und `Image`. Im Gegensatz zum `Chunk`-Objekt kann ein `Paragraph` Texte mit unterschiedlichen Formatierungen enthalten. Außerdem beginnt jeder `Paragraph` in einer neuen Zeile. In einem `Paragraph`-Objekt können viele verschiedene Parameter gesetzt werden, z. B. die Text-Ausrichtung, der Einzug oder der Abstand zwischen den Zeichen.

Neben den oben genannten Objekte stehen auch noch weitere zur Verfügung, die zur Gliederung des Textes beitragen, wie z. B. das `Chapter`- und das `Section`-Objekt.

Im Listing 3.24 wird die Verwendung der gerade vorgestellten Objekte demonstriert. Über die `add()`-Methode können weitere Basisobjekte hinzugefügt werden. In Zeile 5 wird z. B. ein String hinzugefügt, während in Zeile 6 ein `Chunk`-Objekt hinzugefügt wird. In Zeile 8 ist zu sehen, wie eingestellt werden kann, dass der Text zentriert wird. Zum Schluss, in Zeile 10 wird das `Paragraph`-Objekt zum `Document`-Objekt hinzugefügt.

```
1 Paragraph p1 = new Paragraph(new Chunk(
2       "This is my first paragraph. ",
3       FontFactory.getFont(FontFactory.HELVETICA, 10)));
4
5 p1.add("The leading of this paragraph is calculated automagically. ");
```

```
6 p1.add(new Chunk("You can add chunks "));
7 p1.add(new Phrase("or you can add phrases. "));
8 p1.setAlignment(Element.ALIGN_CENTER);
9
10 document.add(p1);
```

**Listing 3.24:** Basiselemente von iText (nach [LS07])

### Tabellen

Im Online-Wörterbuch können die Suchergebnisse als PDF-Dokument heruntergeladen werden, dabei werden die Daten in einer Tabelle dargestellt. Für Tabellen werden die Klassen `PdfPTable` und `PdfPCell` benötigt. Wie vorhin bereits erwähnt, können diese Klassen nur in Verbindung mit dem `PdfWriter` verwendet werden. Für andere Dateiformate muss die Klasse `Table` verwendet werden, die jedoch nicht mehr weiter gewartet wird.

Im Listing 3.25 ist ein Beispiel für eine Tabelle zu sehen. Über einen Parameter im `PdfPTable`-Konstruktor wird die Anzahl der Spalten festgelegt. In diesem Fall sind es drei Spalten. In den Zeilen 5 bis 11 ist zu sehen, wie die einzelnen Zellen hinzugefügt werden. Die erste Zelle überspannt drei Zellen, als eine ganze Reihe. Das wird mit der Methode `cell.setColspan(3)` erreicht. Interessant ist, dass es kein spezielles Objekt für eine Zeile gibt. Die Zellen werden einfach nacheinander befüllt. Wenn die Zeile voll ist, wird automatisch eine neue begonnen und weiter befüllt. Durch Listing 3.25 erhält man eine Tabelle mit drei Zeilen und drei Spalten, wobei die erste Zeile die Kopfzeile darstellt.

```
1 PdfPTable table = new PdfPTable(3);
2 PdfPCell cell =
3   new PdfPCell(new Paragraph("header with colspan 3"));
4 cell.setColspan(3);
5 table.addCell(cell);
6 table.addCell("1.1");
7 table.addCell("2.1");
8 table.addCell("3.1");
9 table.addCell("1.2");
10 table.addCell("2.2");
11 table.addCell("3.2");
```

```
12 document.add(table);
```

**Listing 3.25:** PdfPTable und PdfPCell (nach [LS07])

**Geschachtelte Tabellen:** Aufgrund der Architektur von `PdfPTabel` ist es nicht möglich, dass eine Zelle mehrere Zeilen überspannt. Aus diesem Grund können in Zellen wieder PdfPTables eingefügt werden. Dazu kann die Methode `addCell(PdfPTable table)` verwendet werden.

**Formatieren der Zellen:** iText bietet eine Vielzahl an Formatierungsmöglichkeiten. Es können alle gängigen Eigenschaften wie z. B. die Zellenfarbe, die Zellenhöhe, die Ausrichtung u. v. m. beeinflusst werden. Für weitere Informationen sei auf [LS07] verwiesen.

#### Bilder

iText kann auch diverse Bildformate in PDF-Dokumente integrieren. Mögliche Formate sind JPEG, GIF, PNG, TIFF, BMP, WMF und EPS. Außerdem können `java.awt.Image` - Objekte und verschiedene Barcodes direkt als Bilder eingebunden werden.

Geöffnet werden Bilder über die statische Methode `Image.getInstance()`. Als Parameter kann dabei entweder eine URL oder ein String, der auf eine Datei zeigt, angegeben werden. Das erzeugte Exemplar kann dann mit `add()` bzw. `addCell()` zum Dokument hinzugefügt werden.

Image-Objekte können über eine Transformationsmatrix verschoben, skaliert und rotiert werden. Eine einfachere Variante, um die Bildgröße anzupassen, ist die Methode `scaleToFit(float fitWidth, float fitHeight)`. Über die Parameter können Breite und Höhe des Bildes eingestellt werden. Das Seitenverhältnis bleibt bei der Transformation erhalten.

### 3.7.3 Bestehende PDF-Dateien einbinden

Manchmal kann es nützlich sein, Seiten eines bereits bestehenden PDF-Dokuments einzubinden. Im Online-Wörterbuch besteht z. B. die Möglichkeit, ein Deckblatt zum PDF-Wörterbuch hinzuzufügen.

Um eine Seite einbinden zu können, werden die Klassen `PdfReader`, `PdfImportedPage` und `PdfContentByte` benötigt. Durch das `PdfContentByte`-Objekt kann der Entwickler Texte oder Grafiken exakt im PDF-Dokument positionieren. Der `PdfReader` kann ein bestehendes PDF-Dokument öffnen und lesen. Das `PdfImportedPage`-Objekt kann genau eine PDF-Seite enthalten.

Im Listing 3.26 ist ein Code-Fragment zu sehen, das eine Seite einer bestehenden PDF-Datei einbindet. Durch die Methode getImportedPage() in Zeile 3 wird die erste Seite geladen. In Zeile 4 wird die Seite zum PDF-Dokument an den Koordinaten (0,0) hinzugefügt.

```
1 PdfContentByte directcontent = writer.getDirectContent();
2 PdfReader reader = new PdfReader("existing.pdf");
3 PdfImportedPage page1 = writer.getImportedPage(reader, 1);
4 directcontent.addTemplate(page1, 0, 0);
```

**Listing 3.26:** Einbinden einer PDF-Seite (nach [LS07])

## 3.8 Weitere Komponenten und Bibliotheken

Neben den bereits vorgestellten Technologien und Komponenten werden noch einige weitere vom Online-Wörterbuch verwendet. In diesem Abschnitt werden die Komponenten kurz vorgestellt, ohne jedoch genauer auf die Verwendung einzugehen.

### 3.8.1 JavaScript-Bibliothek: Scriptaculous

Die JavaScript-Bibliothek Scriptaculous bietet ein Vielzahl an visuellen Effekten und UI-Komponenten für interaktive Webseiten. Die Bibliothek steht unter der MIT-Lizenz und kann somit auch für kommerzielle Produkte verwendet werden [N.N07e]. Das Online-Wörterbuch verwendet lediglich einige visuelle Effekte, etwa um Beispielsätze in der Ergebnistabelle ein- und auszublenden oder um dem Benutzer anzuzeigen, dass ein Link angeklickt wurde.

### 3.8.2 JavaScript-Bibliothek: Sweet Titles

Sweet Titles ist eine kleine JavaScript-Bibliothek, die es erlaubt, Tooltips mit eigenen Cascading Style Sheets (CSS) zu versehen und diese anzuzeigen. Die Bibliothek ist in Version 1.0 und unter der Creative Commons Lizenz frei verfügbar [Dia05].

### 3.8.3 LAME MP3 Encoder

Mit dem LAME Encoder können WAV-Dateien in MP3-Dateien konvertiert werden. Das Online-Wörterbuch verwendet den Encoder, um die vertonten Wörter und Beispielsätze zu konvertieren. Die Entwicklung des Encoders begann um 1998 und wurde zuerst von Mike Cheng vorangetrieben. Momentan steht die Version 3.97 als Quellcode zum Download bereit und kann auf sehr vielen Plattformen kompiliert werden. Da der Encoder unter LGPL steht, kann er auch in kommerziellen Projekten eingesetzt werden [N.N06b].

### 3.8.4 MP3 Player: Wimpy Button

Der Wimpy Button ist eine Adobe Flash - Komponente, die es erlaubt, MP3-Dateien auf einer Webseite abzuspielen. Die Komponente kann sehr einfach, nur durch HTML-Code konfiguriert und in die Seite integriert werden. Außerdem steht auf der Wimpy-Webseite ein Programm zur Verfügung, welches den HTML-Code erzeugen kann. Wimpy Button ist kostenpflichtig. Es gibt eine gratis Testversion, die jedoch lediglich 10 Sekunden einer Datei abspielen kann [N.N06d]. Der MP3-Player wird im Online-Wörterbuch für das Abspielen der vertonten Wörter und Beispielsätze eingesetzt.

### 3.8.5 CSV Parser: OpenCSV

OpenCSV ist eine kleine Open-Source-Bibliothek zum Lesen und Schreiben von CSV-Dateien. Ursprünglich wurde sie von Glen Smith implementiert. OpenCSV steht unter der Apache-2.0-Lizenz und ist somit auch für kommerzielle Zwecke nutzbar [Smi07]. Diese Bibliothek wird im Online-Wörterbuch verwendet, um Wörter und Beispielsätze einfach in das System zu importieren.

# Kapitel 4

# Implementierung des Systems

In diesem Kapitel wird nun auf die Systemanforderungen und die Konfiguration eingegangen. Außerdem werden wichtige Klassen und die Package-Struktur des Systems beschrieben. Abschließend werden ausgewählte Anwendungsfälle und deren Umsetzungen vorgestellt.

## 4.1 Systemanforderungen

Für den erfolgreichen Betrieb des Online-Wörterbuchs werden nachfolgende Komponenten benötigt:

**Java:** Das System wurde mit Java 5 kompiliert, daher muss die Java Laufzeitumgebung Version 1.5 oder höher haben.

**Java-Anwendungsserver:** Entwickelt wurde das System auf einem Mortbay Jetty Server Version 4.2.24. Trotzdem sollte es auf den meisten Java-Anwendungsservern funktionieren. Auf einem Apache Tomcat Server 6.0.10 wurde eine Installation erfolgreich getestet.

**Datenbanksystem:** Das System wurde für eine MySql-Datenbank entwickelt, dabei wurde Version 5.0.27 verwendet.

**MP3-Encoder:** Zum Komprimieren der Audiodaten wird der der Lame-Encoder in Version 3.97 verwendet. Der Quellcode kann auf der Projektseite [N.N06b] her-

untergeladen werden. Um das System verwenden zu können, muss der Quellcode des MP3-Encoders kompiliert und in das Verzeichnis `wavefiles` kopiert werden.

Die restlichen Komponenten des Systems sind in der komprimierten WAR-Datei bereits enthalten. Nach der Installation der oben genannten Komponenten sind noch einige Einstellungen nötig, die im nächsten Abschnitt beschrieben werden.

## 4.2 Konfiguration

Dieser Abschnitt beschäftigt sich mit der Konfiguration der Systems. Zunächst wird auf die Konfiguration der Datenbank eingegangen. Anschließend werden die Einstellungsmöglichkeiten der Property-Dateien behandelt.

**Datenbank:** Um das Online-Wörterbuch betreiben zu können, muss zunächst das Datenbank-Schema aus Anhang B in der MySQL-Datenbank angelegt werden. Des Weiteren müssen die Verbindungsdaten, wie z. B. Benutzername und Passwort des Datenbankservers in die Datei `hibernate.cfg.xml` eingetragen werden.

**DictionaryApplication:** Die meisten Einstellungen sind in der Datei `DictionaryApplication.properties` nötig. Hauptsächlich beschränkt sich diese Datei auf die Konfiguration von URL- und Pfad-Angaben. Oft müssen beide Angaben für nur eine Ressource angegeben werden, um eben über HTML(URLs) und Java(Pfad-Angaben) die Ressource verwenden zu können. Die verschiedenen Einstellungsmöglichkeiten werden nun erklärt.

`externalLink` Dieser Variable muss eine URL zugewiesen werden. Sie enthält die URL für einen externen Link, der im System dynamisch mit einem Wort und einer Sprache verknüpft werden kann. Der resultierende Link kann dann z. B. eine Suche auf einer externen Webseite auslösen. Da mehrere Links für verschiedene Sprachen definiert werden können, müssen die Variablen durchnummeriert werden z. B. `externalLink1`, `externalLink2`. Die Nummerierung bestimmt auch die spätere Anzeigereihenfolge im Info-Fenster. Neben der URL muss der Variable auch eine

Sprache und eine Beschreibung, jeweils durch Semikolon getrennt, zugewiesen werden.

Beispiel: `http://www.google.com/search?hl=at&q=_WORDtOsEARCHfor_`;AT;Google
Diese URL würde nur bei einem Wort im oberösterreichischen Dialekt angezeigt werden, da als Sprache *AT* definiert wurde. Außerdem würde im Info-Fenster als Beschreibung *Google* angezeigt werden. Der Platzhalter *_WORDtOsEARCHfor_* wird in der URL beim Anzeigen dynamisch durch das Wort ersetzt.

`replacementString` Diese Variabel definiert den Platzhalter für das Wort in der URL eines externen Links. Normalerweise hat der Platzhalter den Text *_WORDtOsEARCHfor_*.

`langReplacementString` Bei manchen externen Links ist es nötig, dass dynamisch eine Sprache mit angegeben wird. Wenn dieser Platzhalter in einer URL vorkommt, dann wird er durch die Sprache (z. B. AT für Österreich) ersetzt. Normalerweise hat der Platzhalter den Text *_LANGtAG_*.

`urlToHomePage` Dieser Variable muss die URL der Hauptseite zugewiesen werden. Sie wird vor allem vom Aufnahme-Applet verwendet, um zu wissen, wohin die aufgenommenen Daten geschick werden müssen.

`pathToWaveFilesApplet` Damit das System weiß, wo die aufgenommenen Wörter gespeichert werden sollen, muss der Ordner in dieser Variable angegeben werden.

`pathToWaveFiles` Da der MP3-Player über eine URL auf die aufgenommenen Wörter zugreift, muss in dieser Variable die URL zu den MP3-Dateien festgelegt werden.

`lameCmd` Diese Variable definiert den Dateinamen des MP3-Encoders. In einer Windows-Umgebung ist das normalerweise *lame.exe*. Für Linux muss der Name entsprechend angepasst werden.

`lamePath` Über diese Variable kann der Ordner angegeben werden, der den MP3-Encoder enthält.

`pathToWimpy` Diese Variable enthält die URL zum MP3-Player Wimpy-Button.

`osDependantPathSeparator` In dieser Variable muss in Abhängigkeit vom Betriebssystem das Trennzeichen für Pfadangaben eingeben werden. Für Windows ist das \\ und / für Linux-Systeme.

`annotationImages` Diese Variable definiert eine URL, die angibt, wo die Icons für die Anmerkungen zu finden sind.

`pathToAnnotationImages` In dieser Variable wird der Ordner zu den Icons angegeben. Hauptsächlich wird diese Angabe für das Laden der Icons beim Erzeugen der PDF-Dateien verwendet.

`smtpServer` Damit Wortvorschläge von den Benutzern verschickt werden können, muss in dieser Variable ein SMTP-Server eingetragen werden.

`mailFrom` Hier wird die E-Mail-Adresse angegeben, die als Absender eines Wortvorschlags angezeigt werden soll.

`mailTo` In dieser Variable kann der Empfänger der Wortvorschläge eingetragen werden.

`subject` Mit dieser Variable wird der Betreff der E-Mail festgelegt.

`pathToClassFiles` Damit beim Erzeugen der PDF-Dokumente verschiedenen Ausgabestile verwendet werden können, muss der Order angegeben werden, indem sich die `ExportPage`-Klassen befinden.

**PDF Deckblatt:** Eine weitere Einstellungsmöglichkeit bietet die Datei `ExportPage.properties`. Sie enthält die Variable `pathToDictionaryCoverPDF`. Diese Variable kann auf eine PDF-Datei zeigen, welche ein Deckblatt enthält. Wenn die Variable definiert ist, dann wird beim Erzeugen eines druckfertigen Wörterbuchs das angegebene Deckblatt verwendet. Falls ein Deckblatt in unterschiedlichen Sprachen gewünscht wird, dann müssen sprachabhängige Property-Dateien hinzugefügt werden. Das kann man erreichen, indem man beim Namen der Datei das Sprachkürzel anhängt. Eine Datei für die Sprache Deutsch würde z. B. den Namen `ExportPage_de.properties` haben.

## 4.3 Architekur des Systems

### 4.3.1 Package Struktur

Der Quellcode wurde in drei Packages aufgeteilt. Die Webseiten sowie alle erstellten Wicket-Komponenten befinden sich im Package `at.jku.fim.dictionary`. Data-access-

objects (DAO) und die dazugehörigen Modelle sind im Package `at.jku.fim.dictionary.db` zu finden. In den DAOs werden u. a. Methoden von Hibernate ausgeführt, die auf die Datenbank zugreifen. Im Package `at.jku.fim.dictionary.applet` befinden sich alle Klassen, die das Aufnahme-Applet betreffen.

Zwei besonders wichtige Klassen, (`BaseWebPage` und `DictionaryProperties`), befinden sich im Package `at.jku.fim.dictionary` und werden in den folgenden beiden Abschnitten genauer behandelt.

### 4.3.2 Basis-Klasse: `BaseWebPage`

Die Klasse `BaseWebPage` ist die Basis-Klasse für alle Webseiten des Systems, d. h. jede Seite muss von dieser Klasse abgeleitet werden. In Abbildung 4.1 ist die Startseite zu sehen, die bereits von der Basis-Klasse abgeleitet wurde. Die dunkle Fläche rechts vom Menü ist jener Bereich, der von den erbenden Klassen ausgefüllt wird. In diesem Fall von der Such-Seite. Die Komponenten rund um diese Fläche gehören zur Basis-Klasse. So werden Änderungen an der Basis-Klasse automatisch auf alle abgeleiteten Klassen übertragen. Vererbt wird dabei sowohl der Java-Code, als auch der HTML-Code.


**Abbildung 4.1:** Startseite des Systems

### 4.3.3 Hilfs-Klasse: `DictionaryProperties`

Die Klasse `DictionaryProperties` hat drei wichtige Aufgaben. Eine davon ist es, eine `HashMap` jener Sprachen bereitzustellen, die das System anbieten soll (siehe Listing 4.1).

**Sprachen:** Um die Sprachen zu erweitern, muss lediglich die Sprache selbst und das dazugehörige Kürzel in die `HashMap` hinzugefügt werden. Das Kürzel wird dabei hauptsächlich für die interne Darstellung der Sprache in der Datenbank und für die Ergebnisliste der Auto-Vervollständigung verwendet.

```
1 languages = new HashMap<String, String>();
2
3 languages.put("Austrian", "AT");
4 languages.put("German", "DE");
5 languages.put("English", "EN");
```

**Listing 4.1:** Setzen verfügbarer Sprachen

**Verwaltung der Rollen:** In der Klasse `DictionaryProperties` werden die möglichen Rollen, die ein Anwender einnehmen kann, zentral verwaltet. Änderungen der Rechte müssen in dieser Klasse vorgenommen werden. Im nächsten Abschnitt (4.3.4) wird detailliert auf die unterschiedlichen Rollen eingegangen.

**Zugriff auf DAOs:** `DictionaryProperties` stellt auch Get-Methoden für die Data-Access-Objects zur Verfügung. Wie im Listing 4.2 zu sehen ist, wird bei einem gelockten DAO einfach ein neues DAO angelegt und zurückgegeben, damit der Datenbank-Zugriff nicht warten muss.

```
1 public static UserDAO getUserDao() {
2     if(userDao.hasLock()){
3         return new UserDAO(sessionFactory);
4     }
5     return userDao;
6 }
```

**Listing 4.2:** Beispiel für eine Get-DAO-Methode

### 4.3.4 Rollenkonzept

Das System kann zwischen vier verschiedenen Rollen unterscheiden: dem normalen nicht-angemeldeten Benutzer, dem angemeldeten Benutzer, dem Sprecher und dem Administrator. Wie man in den Abbildungen 4.2(a) - 4.2(d) sehen kann, stehen den verschiedenen Rollen unterschiedliche Funktionen zur Verfügung. Ein anonymer, nicht-angemeldeter Benutzer kann sich z. B. nur einloggen, sich registrieren, nach Wörtern suchen, Wörter vorschlagen und das Impressum ansehen. Beim angemeldeten Benutzer (Abb. 4.2(b)) werden die Menüpunkte *Login* und *Registrieren* ausgeblendet, dafür werden aber die Menüpunkte *Logout* und *Meine Vokabeln* eingeblendet. Der Sprecher hat zusätzlich den Menüpunkt *Wort aufnehmen*. Der Administrator hat Zugriff zu allen Menüpunkten und vereint daher auch alle Rollen. Je nach Rolle werden auch bei der Such- und Ergebnisseite unterschiedliche Funktionen freigeschaltet. Dazu jedoch mehr im nächsten Abschnitt (4.4.1).

Ein Rollenkonzept geht meist mit Authentifizierung einher. Um in Wicket den Authentifizierungsmechanismus nutzen zu können, muss die Anwendung von der Klasse `AuthenticatedWebApplication` und die Session von der Klasse `AuthenticatedWebSession` abgeleitet werden. Diese Klassen stellen einige abstrakte Methoden für die Verwaltung des Web-Anwendungszustands zur Verfügung.


(a) Anonymer Benutzer (b) Angemeldeter Benutzer (c) Sprecher (d) Administrator

**Abbildung 4.2:** Menüs nach den verschiedenen Rollen

## 4.4 Umsetzung

Dieser Abschnitt beschäftigt sich mit der Umsetzung des Systems. Es werden jedoch nur ausgewählte Anwendungsfälle dargestellt, um den Umfang dieser Arbeit in Grenzen zu halten.

### 4.4.1 Such- und Ergebnisseite

In diesem Abschnitt wird nun das Herzstück, die Such- und Ergebnisseite des Systems, erklärt. In Abbildung 4.3 sind die Ergebnisse für den Suchbegriff *ha* zu sehen. Außerdem

wurden vier Bereiche mit Nummern markiert, die nun näher betrachtet werden.

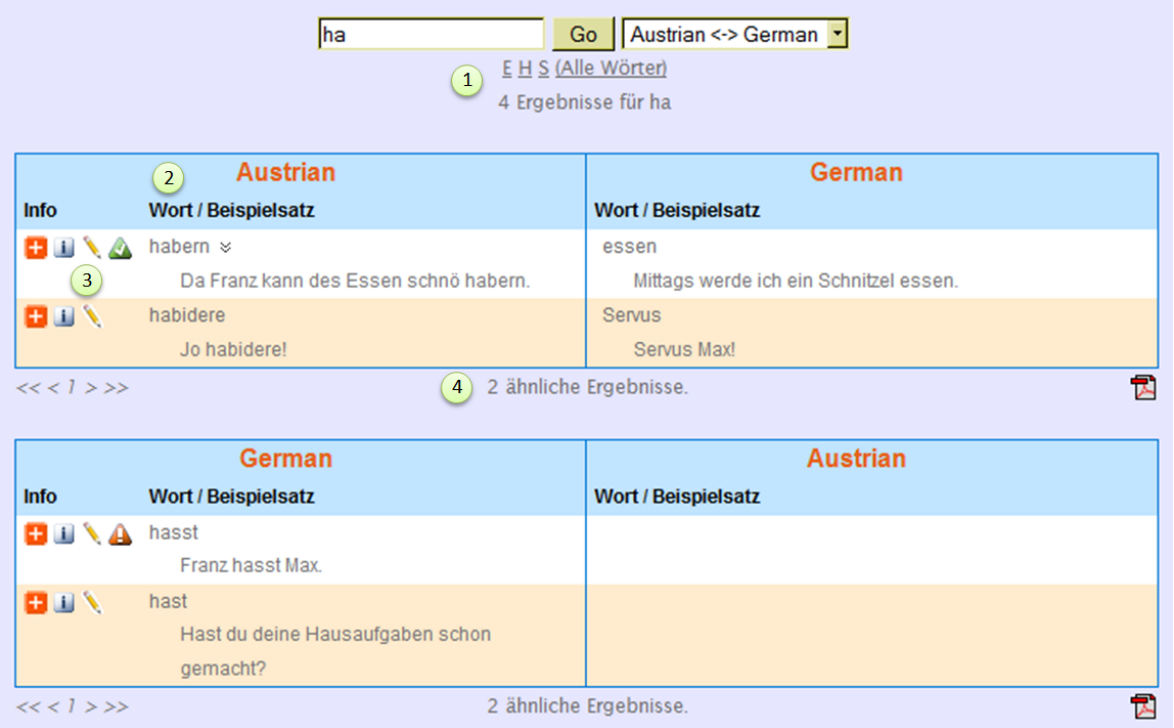



**Abbildung 4.3:** Such- und Ergebnisseite

**Punkt ① (Eingabemaske):** Die Eingabemaske besteht aus fünf Komponenten. Im Textfeld können Suchbegriffe eingegeben werden. Der Benutzer wird dabei durch eine Auto-Vervollständigung unterstützt. Diese Komponente und deren Verhalten wird in Abschnitt 4.4.2 noch genau beschrieben. Im Drop-Down-Menü können die Sprachen ausgewählt werden. In diesem Fall wird von Oberösterreichisch nach Deutsch und umgekehrt übersetzt. Es besteht aber auch die Möglichkeit, nur eine Sprache zu wählen, dadurch erhält man Synonyme als Ergebnis. Der Go-Button löst die Suche aus. Direkt darunter sind die Anfangsbuchstaben jener Wörter aufgelistet, die für die gewählte Sprachkombination in der Datenbank vorhanden sind. In diesem Fall sind nur Wörter mit den Anfangsbuchstaben E, H und S vorhanden. Klickt man auf einen dieser Buchstaben, dann werden alle Wörter, die mit diesem Buchstaben beginnen, angezeigt. Durch eine Änderung der Sprachkombination wird diese Liste automatisch per Ajax aktualisiert. Für angemeldete Benutzer gibt es auch die Möglichkeit, sich alle Wörter anzeigen zu lassen.

**Punkt ② (Tabelle allgemein):** Die Ergebnistabelle ist eine in sich abgeschlossene, wiederverwendbare Komponente. Das Verhalten der Tabelle wird dabei nur durch die eingegebenen Parameter bestimmt. Im nächsten Abschnitt werden die Mechanismen hinter dieser Komponente detailliert behandelt. In Abbildung 4.3 sind lediglich zwei Ergebnistabellen zu sehen. Eine Tabelle mit ähnlichen Ergebnissen für die Übersetzung Oberösterreichisch - Deutsch und eine Tabelle mit ähnlichen Ergebnissen für die Übersetzung Deutsch - Oberösterreichisch. Würde der Suchbegriff mit einem Wort in der Datenbank exakt übereinstimmen, dann würde eine weitere Ergebnistabelle mit exakten Ergebnissen eingeblendet werden. Bei einer Suche können also bis zu vier Ergebnistabellen angezeigt werden.

**Punkt ③ (Zeilen):** Eine Zeile besteht immer aus Icons, die im Info-Bereich angezeigt werden und einem Wort in der linken Spalte. Wenn Beispielsätze vorhanden sind, wird nur einer angezeigt. Weitere Beispielsätze können bei Bedarf durch Klicken auf das Pfeil-Icon ausgeklappt werden.

Ist eine Übersetzung vorhanden, so wird diese in der rechten Spalte angezeigt.

Gleichzeitig werden maximal vier Icons angezeigt. Das Plus-Icon wird nur angezeigt, wenn der Benutzer eingeloggt ist. Wird auf das Icon gedrückt, dann wird dieses Wort zur persönlichen Vokabelliste hinzugefügt. Das i-Icon wird immer angezeigt. Es kann ein Info-Fenster mit weiterführenden Verknüpfungen und Anmerkungen zum Wort öffnen. Mehr dazu in Abschnitt 4.4.6. Das Bleistift-Icon wird nur angezeigt, wenn man als Administrator eingeloggt ist. Durch Drücken dieses Icons kann der Datensatz dieses Wortes bearbeitet werden. Das Icon rechts außen kann drei Ausprägungen haben: Ein grünes Häkchen bedeutet, dass das Wort im positiven Sinne verwendet wird. Ein oranges Rufzeichen bedeutet, dass es im negativen Sinne verwendet wird und ein rotes $x$ bedeutet, dass dieses Wort eher nicht verwendet werden soll.

**Punkt ④ (Fußzeile):** Die Fußzeile besteht aus drei Elementen. Links befindet sich eine Nummerierung, die es erlaubt, zwischen mehreren Tabellenseiten zu navigieren. Dies ist jedoch nur möglich, wenn die Tabelle mehr als acht Einträge hat.

In der Mitte wird angezeigt, wie viele Ergebnisse die Tabelle enthält und ob es sich um exakte oder nur ähnliche Ergebnisse handelt.

Auf der rechten Seite befindet sich ein PDF-Icon. Über dieses Icon kann jeder Benutzer die Ergebnistabelle als PDF-Datei herunterladen. Die Anzahl der Wörter

wird, außer bei Administratoren, auf 200 begrenzt.

### Parametrisierung der Ergebnistabellen-Komponente

Die Ergebnistabelle ist eine in sich abgeschlossenen Komponente und kann daher einfach wiederverwendet werden. Durch Ändern ihrer Parameter, z. B. des Suchbegriffes, kann das Verhalten der Komponente beeinflusst werden. Die Komponente befindet sich im Package `at.jku.fim.dictionary` und heißt `SearchResultsPanel`. Die Daten bezieht die Komponente aus der Klasse `DialectDataProvider`. Durch diese Klasse ist es möglich, dass nur jene Wörter und Beispielsätze geladen werden, die auch tatsächlich angezeigt werden. Verlangt der Benutzer nach einer anderen Tabellenseite, dann wird diese nachgeladen.

Gibt ein Benutzer einen Suchbegriff ein und schickt ihn ab, so werden die Parameter der Klasse `DialectDataProvider` neu gesetzt. Genauer gesagt werden der Suchbegriff, die beiden Sprachen und die Suchmethode gesetzt. Es kann entweder nach exakten Übereinstimmungen oder nach ähnlichen Wörtern gesucht werden. Die eigentliche Suche wird aber durch Wicket ausgelöst, nämlich zu dem Zeitpunkt, in dem die Komponente gerendert wird.

### Der Datenfluss in der Ergebnistabellen-Komponente

Bei einer Suche erfolgt der Datenbankzugriff über die Klasse `DialectDAO`, wobei die Anweisungen vom `DialectDataProvider` kommen. Zuerst wird nur eine Liste mit Datenbank-IDs jener Wörter geladen, die den Suchkriterien entsprechen. Die Ergebnistabellen-Komponente fordert dann eine Teilliste an, die von der Komponente anzeigt wird. Wenn die Komponente gerendert wird, können die restlichen Daten wie z. B. Wörter und Beispielsätze von Wicket nachgeladen werden. Bei diesem Vorgang kommt ein spezielles Wicket-Modell (`LoadableDetachableModel`) zum Einsatz, welches im vorigen Kapitel bereits behandelt wurde. Dieses Modell kann nach Bedarf auf das `DialectDAO` zugreifen und die nötigen Daten laden.

### Der Suchvorgang

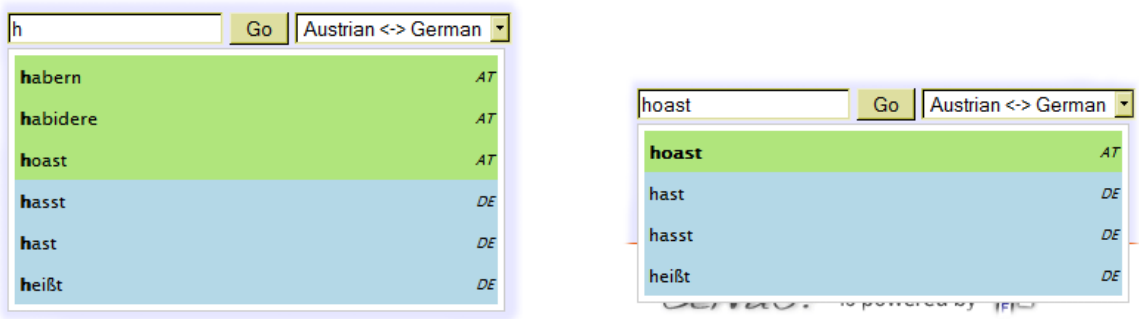
Beim Suchen wird grundsätzlich zwischen zwei Methoden unterschieden. Eine exakte Suche und eine Ähnlichkeitssuche. Bei der exakten Suche werden nur exakt mit dem

Suchbegriff übereinstimmende Wörter geladen. Bei der Ähnlichkeitssuche hingegen kommt der Double-Metaphone-Algorithmus zum Einsatz. Zuerst wird jedoch überprüft, ob der Suchbegriff aus nur einem Zeichen besteht. Ist das der Fall, dann werden jene Wörter geladen, die dieses Zeichen als Anfangsbuchstaben haben. Hat der Suchbegriff mehrere Zeichen, dann werden die Wörter geladen, die den Suchbegriff an irgendeiner Stelle aufweisen. Wörter, deren phonetische Codes mit dem Code des Suchbegriffs übereinstimmen, werden hinten angereiht. Somit sind die Prioritäten für die Ähnlichkeitssuche in folgender Reihenfolge festgelegt:

1. Wörter, die mit dem Suchbegriff beginnen
2. Wörter, die den Suchbegriff enthalten
3. Wörter, deren phonetische Codes mit dem des Suchbegriffs übereinstimmen

### 4.4.2 Auto-Vervollständigung

Bei der Eingabe eines Suchbegriffs wird der Benutzer durch Vorschläge unterstützt. Wie in Abbildung 4.4 zu sehen ist, werden die Vorschläge nach der Sprache aufgeteilt. In diesem Fall wird mit *AT* bzw. *DE* angedeutet, zu welcher Sprache die Wörter gehören. Außerdem werden jene Teilstrings fett dargestellt, die den Suchbegriff enthalten. Mit jedem weiteren Buchstaben, der eingegeben wird, wird die Liste der Vorschläge neu berechnet und angezeigt.




(a) Eingabe: h (b) Eingabe: hoast

**Abbildung 4.4:** Auto-Vervollständigung

Wie man in der Abbildung 4.4(b) sieht, werden nicht nur jene Wörter angezeigt, die

exakt mit dem Suchbegriff übereinstimmen, sondern auch ähnlich klingende Wörter. Im nachfolgenden Abschnitt wird nun gezeigt, wie bei der Suche vorgegangen wird.

#### Erstellen der Vorschläge

Wie im folgenden Pseudo-Code zu sehen ist, wird beim Erstellen der Vorschläge zuerst versucht, exakte Übereinstimmungen zu finden. Erst dann werden ähnlich klingende Wörter hinzugefügt.

1. Füge jener Wörter zu einer Liste hinzu, die den Suchbegriff an irgendeiner Stelle des Wortes enthalten.
    - Wörter, die den Suchbegriff am Anfang haben, werden dabei vorgereiht.
2. Wenn die Liste mit den Vorschlägen noch nicht voll ist:
    - Berechnen der beiden phonetischen Codes durch den Double-Metaphone-Algorithmus.
    - Auswahl jener Wörter, deren phonetischer Code an irgendeiner Stelle einen der beiden berechneten Codes enthält.
    - Ordnen der ähnlich klingenden Wörter mit Hilfe der Damerau-Levenshtein-Distanz.
    - Füge solange Wörter zur Liste hinzu bis sie voll ist.
3. Ausgabe der List

### 4.4.3 Wörter einem Sprecher zuweisen

Ein Administrator hat die Möglichkeit, einem Benutzer Wörter zuzuweisen, die vertont werden sollen. Der Benutzer muss dabei die Rolle eines Sprechers haben, um überhaupt in der Auswahl zu erscheinen. In Abbildung 4.5 ist die Eingabemaske für diesen Anwendungsfall zu sehen. Zuerst muss die Sprache gewählt werden, aus der die Wörter vertont werden sollen. Jedes Mal, wenn die Sprache geändert wird, wird auch die Liste der verfügbaren Wörter aktualisiert. Außerdem wird die Liste der Sprecher angepasst, da nur jene angezeigt werden sollen, die auch tatsächlich die ausgewählte Sprache sprechen

können. Neben der Sprache kann auch die maximale Anzahl der Wörter ausgewählt werden, die in der Liste angezeigt werden soll. Das soll verhindern, dass in einer neu angelegten Datenbank alle Wörter einer Sprache in die Auswahl geladen werden. Außerdem gibt es eine Option, die es erlaubt, jene Wörter anzuzeigen, die bereits vertont oder einem Sprecher zugewiesen worden sind. Das ermöglicht einerseits, falsch oder schlecht vertonte Wörter neu zu vertonen und andererseits, falsch zugewiesene Wörter einem anderen Sprecher zuzuweisen.


**Abbildung 4.5:** Wörter einem Sprecher zuweisen

In Abbildung 4.5 wurde z. B. das Wort *essen* ausgewählt. In der DropDown-Box unten im Bild, kann letztendlich der Sprecher ausgewählt werden. Wie bereits erwähnt, werden nur jene Sprecher in der Auswahl angezeigt, welche die ausgewählte Sprache sprechen können. Wird auf den Button *Wort zuweisen* geklickt, dann wird dem ausgewählten Sprecher das Wort *essen* zugeteilt und aus der Auswahl entfernt.

### 4.4.4 Aufnahme-Applet

Das System verwendet ein Applet zur Aufnahme von Wörtern und Beispielsätzen. Obwohl Applets den Nachteil haben, dass sie und die JVM relativ lange zum Laden benötigen, haben Sie auch einige Vorteile. Ein Hauptargument für das Applet ist z. B., dass die gesamt Java-API zur Verfügung steht. Da die Web-Anwendung ebenfalls in Java implementiert wurde, müssen die aufgenommenen Daten nicht weiter in ein anderes Format konvertiert werden. Der Nachteil der längeren Ladezeit relativiert sich, wenn ein Sprecher das Applet mehrmals öffnet, da der Web-Browser das Applet normalerweise im Cache bereit hält.

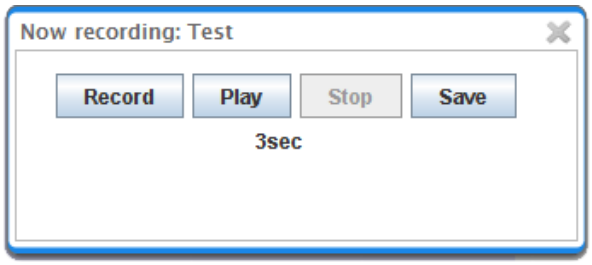



**Abbildung 4.6:** Das Aufnahme-Applet

Das Aufnahme-Applet, wie es in Abbildung 4.6 zu sehen ist, hat zur Bedienung vier Buttons, nämlich *Record*, *Play*, *Stop* und *Save*.

#### Aufnehmen

Durch Betätigen des Record-Buttons wird die Aufnahme gestartet und der Sprecher kann das Wort oder den Beispielsatz in das Mikrofon sprechen. Zur Aufnahme wird die Bibliothek `javax.sound` verwendet. Sie ermöglicht den Zugriff auf ein Mikrofon und die Aufnahme. Im Listing 4.3 ist der Ablauf bei der Aufnahme zu erkennen. Zuerst wird eine „line“ (das Mikrofon) mit einem bestimmten Audioformat geöffnet. Danach wird innerhalb einer while-Schleife solange aufgenommen, bis der Sprecher den Stop-Button gedrückt hat. Beim Aufnehmen werden die Daten in einen byte-Array (`buffer`) geschrieben. Wenn Elemente im `buffer` sind, wird dieser in einen `ByteArrayOutputStream` geschrieben, wo die Daten dann weiter verarbeitet werden können. Über den Play-Button können die Daten beispielsweise wieder ausgelesen und über die Lautsprecher ausgegeben werden.

```
1 line = (TargetDataLine) AudioSystem.getLine(info);
2 line.open(format);
3 line.start();
4 ...
5 while (!stopPressed) {
6    int j = line.read(buffer, 0, buffer.length);
7    if (j > 0) {
8       out.write(buffer, 0, j);
9       ...
10   }
11 }
```

**Listing 4.3:** Aufnahme

### Versand zum Server

Zum Speichern werden die aufgenommenen Daten wieder in einen Byte-Array konvertiert und in ein `RecordedDate`-Objekt kopiert. Dieses serialisierbare Objekt wird auch noch mit anderen wichtigen Daten befüllt, wie z. B. dem Audioformat, dem Wort bzw. Beispielsatz und der Information, ob gerade ein Wort oder ein Beispielsatz aufgenommen wurde. Über die Klasse HttpMessage von Jason Hunter [Hun98] wird das serialisierte `RecordedDate`-Objekt dann per `post` zum Server geschickt. Als Content-Type wird im Request der String *application/fimdictionary* gesetzt. Dadurch kann der Server entscheiden, ob der Request vom Applet oder von einem anderen Client stammt.

### Requestverarbeitung am Server

Am Server werden die Daten eines hereinkommenden Requests zuerst analysiert. Wenn aus den Daten ein `RecordedDate`-Objekt erzeugt werden kann, dann wird mit der Verarbeitung fortgefahren und ein Name für die WAV-Datei erzeugt. Der Name setzt sich aus folgenden Komponenten zusammen:

- Die ersten 16 Buchstaben des Wortes bzw. Beispielsatzes
- Die ID des dazugehörigen Datenbank-Objekts
- Der Buchstabe *W* für ein Wort bzw. *S* für einen Beispielsatz

Für den Beispielsatz „Das ist ein Beispiel“ mit der ID 17 sieht der Dateiname z. B. so aus: `DASISTEINBEISPIE17S.WAV`

Um einen eindeutigen Namen zu erhalten, würde es auch genügen, nur die ID und die Buchstaben W bzw. S heranzuziehen. Auf Grund der Lesbarkeit wurden jedoch die ersten 16 Buchstaben hinzugefügt.

Nachdem die WAV-Datei gespeichert wurde, kann sie mit dem LAME-MP3-Encoder [N.N06b] in eine MP3-Datei umgewandelt werden. Wie in Listing 4.4 zu sehen ist, wird der Befehl zum Konvertieren über das `Runtime`-Objekt von Java ausgeführt.

```
1 Runtime rt = Runtime.getRuntime();
2 String fullCmd = pathToLame + "\\" + "lame.exe" + " " + absolutePath
3             + "\\" + waveFileName + " " + absolutePath + "\\"
4             + mp3FileName;
5 Process process = rt.exec(fullCmd);
```

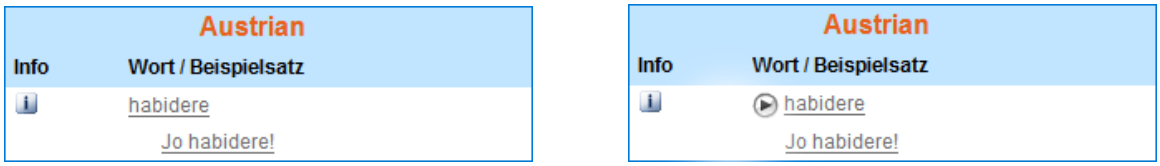
**Listing 4.4:** MP3-Konvertierung

Nach erfolgreicher Konvertierung wird die WAV-Datei gelöscht und ein Eintrag in der Datenbank vorgenommen, dass das Wort bzw. der Beispielsatz vertont wurde.

### 4.4.5 Abspielen vertonter Wörter

Beim Abspielen der vertonten Wörter kommt der MP3-Player Wimpy-Button [N.N06d] zum Einsatz. Wie in Abbildung 4.7 zu sehen ist, werden vertonte Wörter in den Suchergebnissen unterstrichen, also als Link dargestellt. Damit die Tabelle der Suchergebnisse nicht zu unübersichtlich ist, werden die Wimpy-Buttons zunächst ausgeblendet (Abbildung 4.7(a)). Erst wenn der Benutzer das Wort anhören möchte, wird der Player eingeblendet und die MP3-Datei wird automatisch abgespielt (Abbildung 4.7(b)). Der Wimpy-Button ermöglicht es, das Abspielen jederzeit zu stoppen und wieder fortzusetzen.

Das Einblenden des Wimpy-Buttons wurde durch zwei Wicket-Komponenten realisiert. Einem `AjaxLink`, der ausgelöst wird, wenn ein Benutzer auf das Wort klickt und einem `Label`, welches die Konfiguration des Wimpy-Buttons als einfachen String enthält. In Listing 4.5 ist zu sehen, wie die beiden Komponenten angelegt und verknüpft werden. Zeile 2 ist besonders wichtig, da an dieser Stelle eingestellt wird, dass der Inhalt des Labels unverändert in die HTML-Datei übernommen wird. Ab Zeile 10 beginnt die




(a) Wimpy-Button deaktiviert (b) Wimpy-Button aktiviert

**Abbildung 4.7:** MP3-Dateien abspielen mit Wimpy-Button

Methode, die bei einem Klick auf den Link ausgelöst wird. Am Beginn der Methode wird ein Objekt `wimpyContent` angelegt und die Konfiguration des Wimpy-Buttons dynamisch erzeugt. In den Zeilen 22 bis 24 wird beispielsweise der Pfad und der Name der MP3-Datei gesetzt. Nachdem die Konfiguration abgeschlossen wurde, wird diese in Zeile 27 vom Label übernommen. Zum Schluss wird die Label-Komponente per AJAX ausgetauscht und im Browser angezeigt.

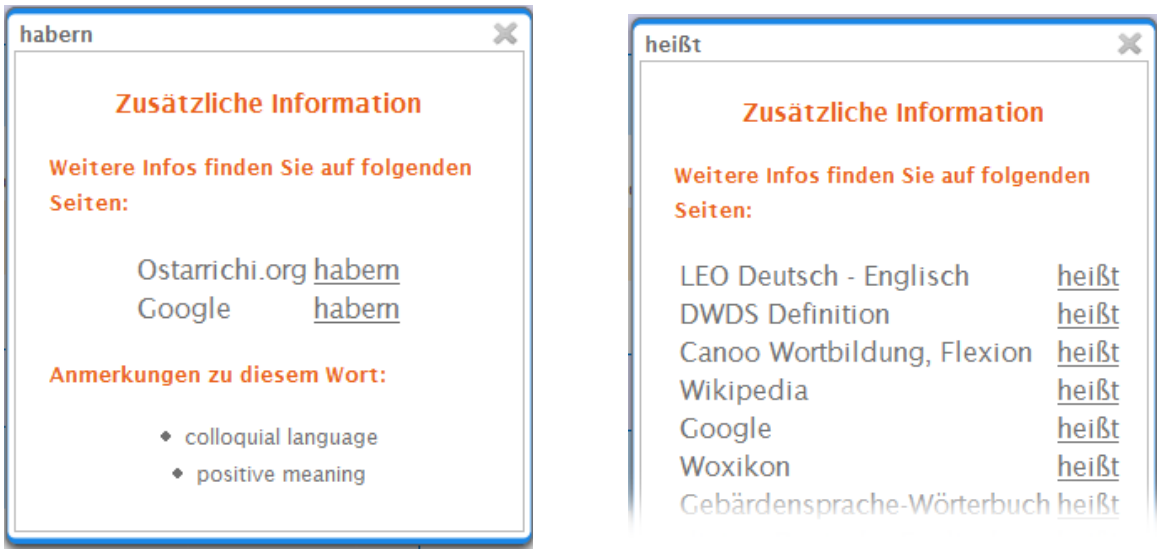
```
1  final Label wimpyLabel = new Label("wimpy", new Model(""));
2  wimpyLabel.setEscapeModelStrings(false);
3  wimpyLabel.setOutputMarkupId(true);
4
5  final AjaxLink exLink = new AjaxLink("wordLink") {
6
7      private static final long serialVersionUID = 592718977598163743L;
8
9      @Override
10     public void onClick(AjaxRequestTarget target) {
11         String pathToWimpy = new ResourceModel("pathToWimpy")
12                 .getObject(this).toString();
13
14         StringBuilder wimpyContent = new StringBuilder();
15         ...
16         wimpyContent.append("<param name=\"movie\" value=\"");
17         wimpyContent.append(pathToWimpy);
18         wimpyContent.append("wimpy_button.swf\" />");
19         wimpyContent.append("<param name=\"loop\" value=\"false\" />");
20         wimpyContent.append("<param name=\"menu\" value=\"false\" />");
21         ...
22         wimpyContent.append("<param name=\"flashvars\" value=\"theFile=");
23         wimpyContent.append(waveModel.getObject(this).toString());
24         wimpyContent.append(fileName);
25         ...
```

```
26
27          wimpyLabel.setModel(new Model(wimpyContent.toString()));
28          target.addComponent(wimpyLabel);
29      }
30  };
```

**Listing 4.5:** Wimpy-Button per AjaxLink einbinden

### 4.4.6 Info-Fenster

Wie in Abbildung 4.8 zu sehen ist, zeigt das Info-Fenster externe Links, die eine weiterführende Suche ermöglichen. Wenn der Benutzer z. B. in Abbildung 4.8(a) auf das Wort *habern* neben Google klickt, dann wird ein neues Fenster im Browser geöffnet und es werden die Suchergebnisse von Google angezeigt. In Abbildung 4.8(b) hingegen sind viel mehr Online-Wörterbücher und Suchmaschinen verlinkt. Das hat den Grund, weil für die deutsche Sprache einfach mehr weiterführende Links vorhanden sind als für den oberösterreichischen Dialekt. Die Links, die im Info-Fenster angezeigt werden, können in der Datei `DictionaryApplication.properties` definiert werden, wie im Abschnitt 4.2 bereits erklärt wurde.




(a) oberösterreichischer Dialekt (b) Deutsch

**Abbildung 4.8:** Info-Fenster unterschiedlicher Sprachen

Außerdem werden Anmerkungen angezeigt, die dieses Wort betreffen. Die Anmerkungen können beim Hinzufügen oder Ändern der Wörter gesetzt werden.

### 4.4.7 Verknüpfen von Wörtern

Beim Anwendungsfall *Wörter hinzufügen und ändern* können neben dem Wort auch Anmerkungen und Beispielsätze hinzugefügt werden. Außerdem kann das neue Wort auch mit bereits bestehenden Wörtern in der Datenbank verknüpft werden. In diesem Abschnitt wird jedoch nur auf das Verknüpfen der Wörter eingegangen, da dies der komplexere Teil dieses Anwendungsfalls ist.

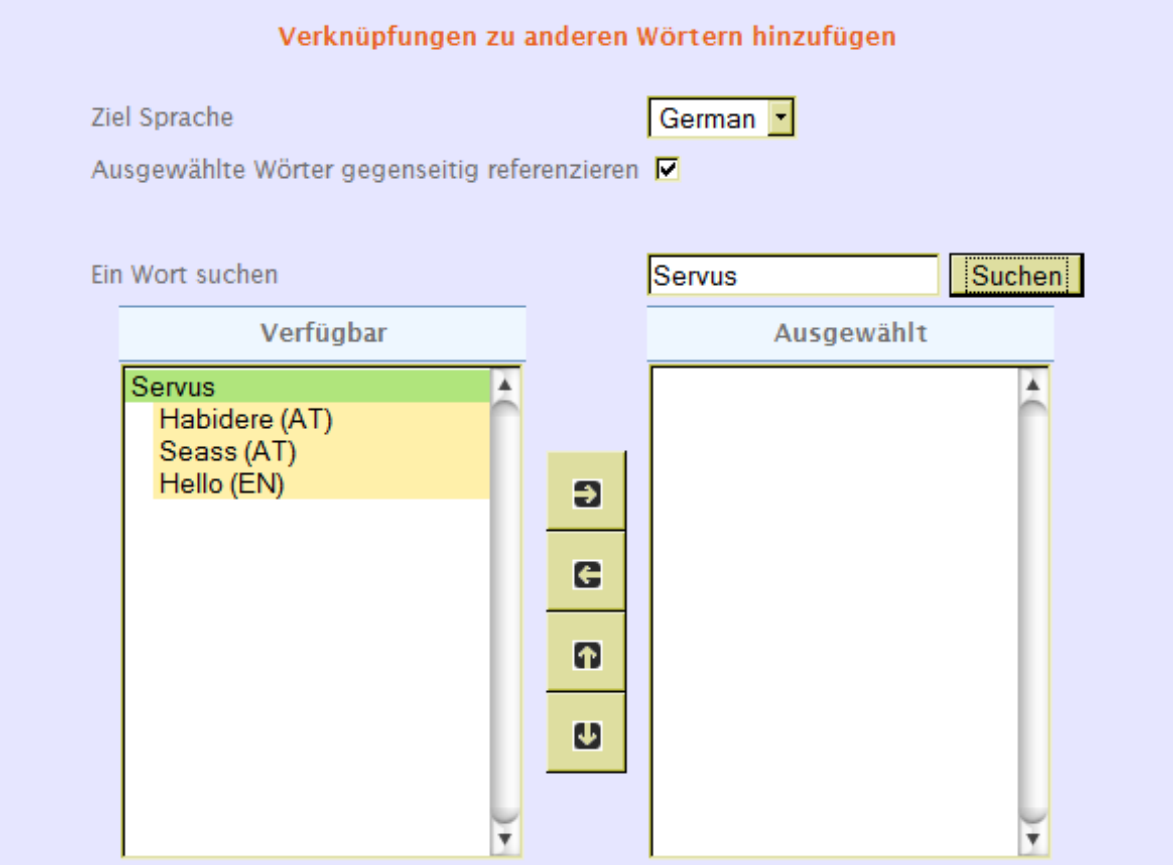



**Abbildung 4.9:** Verknüpfungen hinzufügen

In Abbildung 4.9 ist die Eingabemaske zu sehen. Zuerst kann die Sprache ausgewählt werden, in der die zu verknüpfenden Wörter gesucht werden sollen. In diesem Fall wird nach dem Wort *Servus* gesucht. Die Suchergebnisse werden in der linken Spalte der Palette angezeigt. Wie in der Abbildung zu sehen ist, wurde der Suchbegriff *Servus* gefunden und daher grün hinterlegt. Darunter, etwas abgesetzt, werden jene Wörter angezeigt, die mit dem Suchbegriff bereits verknüpft sind. In diesem Fall die Wörter *Habidere* und *Seass* aus dem oberösterreichischen Dialekt und *Hello* aus dem Eng-

lischen. Dieses Vorgehen ermöglicht es dem Benutzer, die vier gefunden Wörter auf einmal auszuwählen, ohne vier Suchvorgänge zu starten. Außerdem kann in einer beliebigen Sprache gesucht werden. Würde *Hello* als Suchbegriff eingegeben werden, dann würden die selben vier Wörter angezeigt werden.

### 4.4.8 Druckfertiges Wörterbuch erzeugen

Die Webseite zum Erzeugen eines Wörterbuchs ist in Abbildung 4.10 zu sehen. Neben zwei DropDown-Boxen für die Auswahl der Sprachen gibt es noch eine weitere DropDown-Box mit der ein Ausgabestil gewählt werden kann. In dieser Abbildung wurde ein A5-Querformat gewählt. Über eine Check-Box kann gewählt werden, ob auch jene Wörter in das PDF-Dokument übernommen werden, die keine Übersetzung haben. Über den Button *Wörterbuch erzeugen* wird das PDF-Dokument erstellt und im Webbrowser angezeigt. Außerdem besteht die Möglichkeit einzelne Kapitel eines Wörterbuchs zu erzeugen. Ganz unten in Abbildung 4.10 sind jene Anfangsbuchstaben aufgelistet, die in der Datenbank vorkommen. Durch Klicken eines Anfangsbuchstaben wird das Kapitel im Web-Browser angezeigt. Die Liste der Anfangsbuchstaben wird übrigens in Abhängigkeit von den ausgewählten Sprachen angezeigt. Beim Ändern einer Sprache wird die Liste per AJAX aktualisiert.

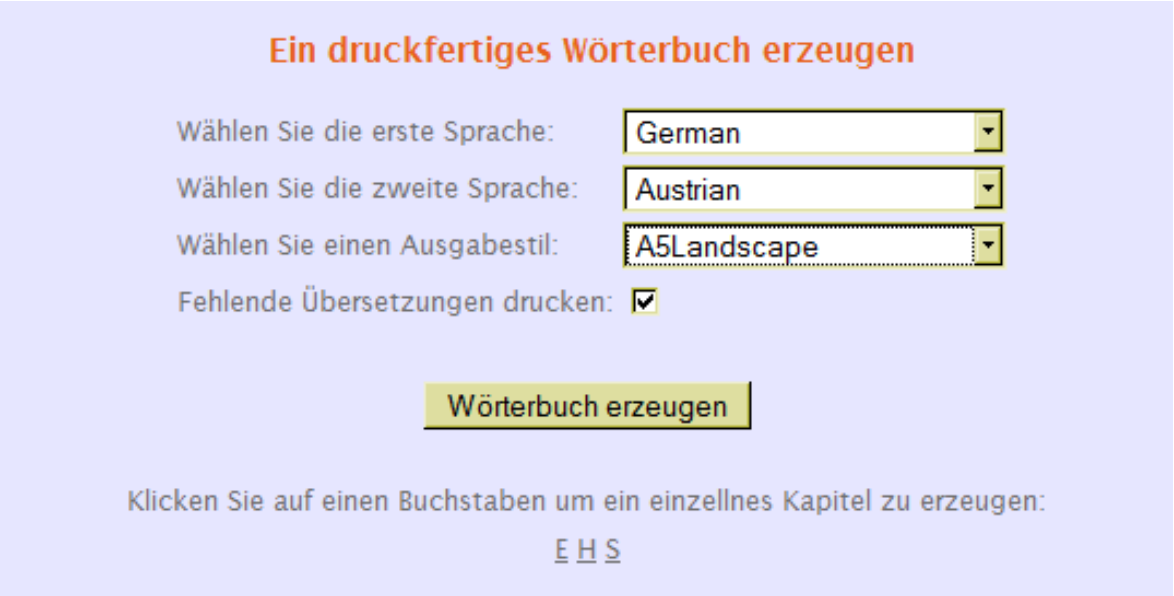



**Abbildung 4.10:** Wörterbuch im PDF erzeugen

Das Erzeugen der PDF-Dokumente wurde mit der Bibliothek iText realisiert, die bereits in Kapitel 3.7 ausführlich behandelt wurde. Damit ein PDF-Dokument im Web-Browser angezeigt werden kann, müssen die Daten auf den Output-Stream eines Response-Objekts geschrieben werden. Dazu wurde eine eigene Klasse (`ExportPage`) entwickelt, die rein für das Erzeugen und Anzeigen eines PDF-Dokuments zuständig ist.

**Ausgabestile:** `ExportPage` enthält sowohl die Logik für den Aufbau der PDF-Tabellen als auch die Informationen über das Layout des resultierenden PDF-Dokuments. Die Klasse `ExportPage` wurde so gestaltet, dass Einstellungen wie z. B. Schriftart, Seitenformat und Tabellenfarbe über öffentliche get-Methoden geladen werden. Um nun einen neuen Ausgabestil für ein PDF-Dokument zu entwickeln, muss lediglich eine weitere Klasse von `ExportPage` abgeleitet werden. Die Logik für das Erzeugen der PDF-Tabellen bleibt so erhalten. Einstellungen, die das Layout betreffen, können durch gezieltes Überschreiben der get-Methoden bewirkt werden. Außerdem gibt es noch mehrere `dataCellDecorator()`-Methoden. Diese sind für das Aussehen der Tabellenzellen zuständig. Diese können ebenfalls überschrieben werden.

Damit eine Klasse als Ausgabestil erkannt wird und auf der Webseite zur Auswahl steht, muss diese von `ExportPage` abgeleitet sein. Außerdem muss der Klassenname mit *ExportPage.class* enden. Zum Beispiel `A5LandscapeExportPage.class`. Wie im Listing 4.6 zu sehen ist, können die Dateinamen sehr einfach durch einen `Filename-Filter` ausgwählt werden. Im String-Array `styleFileNames` befinden sich dann nur jene Dateinamen, die der Namenskonvention entsprechen.

```
1 File directory = new File(pathToClasses);
2 String[] styleFileNames;
3
4 FilenameFilter filter = new FilenameFilter() {
5    public boolean accept(File dir, String name) {
6       return name.endsWith("ExportPage.class");
7    }
8 };
9
10 styleFileNames = directory.list(filter);
```

**Listing 4.6:** Laden der Ausgabestile

Nachdem der Benutzer einen Ausgabestil und somit einen Dateinamen ausgewählt hat,

wird über Reflection ein neues Exemplar der `ExportPage`-Klasse erzeugt. Anschließend wird das PDF-Dokument generiert.

**Daten auf das Response-Objekt schreiben:** Da das Webframework Wicket den Request-Response-Zyklus intern durchführt, hat der Entwickler keinen direkten Zugriff auf das Response-Objekt, wie das bei einem Servlet der Fall wäre. In Wicket muss dem Request-Response-Zyklus daher ein neues `RequestTarget`-Objekt zugewiesen werden, wie in Listing 4.7 zu sehen ist. In Zeile 4 kann dann direkt auf das Response-Objekt zugegriffen werden. Der Typ des Response wird in Zeile 8 gesetzt, damit der Webbrowser weiß, welchen Dateityp er empfängt. Das erzeugte PDF-Dokument wird temporär im Objekt `byteArrayOS` gehalten. In Zeile 12 ist zu sehen, wie die Daten vom `byteArrayOS` letztendlich in den Output-Stream des Response geschrieben werden.

```
1  RequestCycle.get().setRequestTarget(new IRequestTarget() {
2
3      public void respond(RequestCycle requestCycle) {
4          Response response = requestCycle.getResponse();
5          //reset internal response data
6          response.reset();
7
8          response.setContentType("application/pdf");
9
10         try {
11             //write data to response
12             byteArrayOS.writeTo(response.getOutputStream());
13             byteArrayOS.close();
14         } catch (IOException e) {
15             e.printStackTrace();
16         }
17     }
18     ...
19 });
```

**Listing 4.7:** Daten auf das Response-Objekt schreiben

# Kapitel 5

# Nachbetrachtung

## 5.1 Mögliche Erweiterungen

Während der Entwicklung des Online-Wörterbuch-Systems entstanden einige Ideen, wie das System weiter entwickelt und so der Service für den Benutzer erweitert werden könnte. Aus Zeitmangel wurden diese Erweiterungen jedoch nicht implementiert. Die folgenden Ideen können aber eventuell in einem Folgeprojekt verwirklicht werden.

**Podcast** Eine nützliche Erweiterung wäre es, einen Podcast anzubieten, der in bestimmten Abständen automatisch erzeugt wird. Ein solcher Podcast könnte z. B. „Das Wort des Tages“ oder „Die fünf meist gesuchten Wörter der Woche“ enthalten. Damit muss aber auch eine Art Steuerung geschaffen werden, die es erlaubt, dass bestimmte Wörter nicht oder nur an bestimmten Tagen für einen Podcast verwendet werden. Außerdem muss eine Lösung für das Verknüpfen von MP3-Dateien gefunden werden. Das Java Media Framework API (JMF) bietet zwar unter Windows eine Möglichkeit, MP3-Dateien zu lesen und zu verknüpfen, jedoch nicht unter Linux [N.N07c]. Da das System aber hauptsächlich auf einem Linux-Server läuft, kann es nicht verwendet werden.

**Vokabeltrainer** Eine sehr sinnvolle Erweiterung wäre ein Vokabeltrainer. Der Vokabeltrainer könnte z. B. Wörter aus der persönlichen Vokabelliste abfragen. Des weiteren wäre denkbar, dass Vokabeln in verschiedene Schwierigkeitsklassen eingeteilt werden. Dadurch könnten Benutzer aus bereits vorbereiteten Vokabellisten

auswählen. Eine weitere Möglichkeit wäre es, ein Kreuzworträtsel mit Vokabeln und deren Übersetzungen zu implementieren.

**Forum** Wenn Benutzer Wörter oder Phrasen nicht im Wörterbuch finden können, dann wäre ein Forum, in dem Fragen gestellt werden können, eine ideale Ergänzung. Außerdem könnte bei einer erfolglosen Suche im Wörterbuch auch im Forum weiter gesucht werden.

**Synonyme** Die Suchergebnisse könnten auch um Synonyme ergänzt werden. Das Projekt Open-Thesaurus[1] stellt beispielsweise ein Synonym-Verzeichnis in vielen verschiedenen Formaten zum Herunterladen zur Verfügung. Darunter auch einen MySQL-Dump, um die Daten in die eigenen Datenbank zu integrieren. Außerdem sind auf der Webseite des Projekts Links zu vielen anderssprachigen Synonym-Verzeichnissen zu finden.

**MySpell** Bei der Eingabe neuer Wörter könnte ein Administrator durch eine Rechtschreibkorrektur unterstützt werden. MySpell-Wörterbücher stehen in vielen verschiedenen Sprachen zur Verfügung.

**Wörter durch Benutzer hinzufügen** Eine Möglichkeit das Wörterbuch rasch zu füllen, wäre, wenn nicht nur Administratoren, sondern auch normale Benutzer Wörter hinzufügen könnten. Dazu muss allerdings ein Mechanismus geschaffen werden, dass keine falschen Wörter oder Beispielsätze in das Wörterbuch aufgenommen werden. Beim Online-Wörterbuch dict.cc müssen neue Wörter z. B. von zehn Benutzern als richtig gewertet werden, damit es fix aufgenommen wird.

**Statistik** Eine Statistik der Suchbegriffe würde es einem Administrator z. B. erlauben, die meistgesuchten Wörter, die nicht in der Datenbank enthalten sind, zu identifizieren. Somit könnte gezielter auf die Wünsche der Benutzer eingegangen werden.

[1] `http://www.openthesaurus.de/`

## 5.2 Resümee

Wie in dieser Magisterarbeit gezeigt wurde, tendiert die Entwicklung von Web-Anwendungen immer mehr zur Verwendung von bereits vorgefertigten Bibliotheken und Frameworks. Alleine die Implementierung des Online-Wörterbuch-Systems setzt sich aus über zwölf verschiedenen Bibliotheken und Frameworks zusammen. Einerseits wird dem Entwickler dadurch ein großer Teil der Implementierung abgenommen, was zunächst auf eine kurze Entwicklungszeit schließen lässt. Andererseits nimmt mit jeder Komponente die Komplexität des Systems zu. Außerdem bedeutet jede Komponente einen mehr oder weniger großen Einarbeitungsaufwand, der nicht zu unterschätzen ist. Vor allem die verschiedenen Webframeworks erscheinen auf den ersten Blick sehr komplex. Ein Webframework für ein Projekt auszuwählen ist daher sehr zeitaufwendig, da man zunächst die Architektur und die Vor- und Nachteile kennen muss. Hat man die Konzepte jedoch verstanden, dann geht die Entwicklung relativ zügig voran.

Trotzdem stößt man während der Entwicklung immer wieder auf Probleme, die vom hohen Abstraktionsgrad der Webframeworks herrühren. Auf ein Response-Objekt zuzugreifen ist in einem Servlet beispielsweise eine Leichtigkeit. In Webframeworks wird der Request-Response-Zyklus aber oft intern durchgeführt, somit ist kein direkter Zugriff auf das Response-Objekt mehr möglich. Ein Zugriff muss dann über spezielle Klassen und Methoden erfolgen.

Egal, für welches Webframework man sich entscheidet, man muss Kenntnisse über die Basis-Web-Technologien mitbringen, um die Vorteile der Komponenten-orientierten Softwareentwicklung nutzen zu können und um die Komponenten effizient einzusetzen.

# Anhang A

# Konfigurations-Dateien

# A.1 Hibernate Konfiguration

## A.1.1 Hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- <property name="hibernate.bytecode.use_reflection_optimizer"
            >false</property> -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc
            .Driver</property>
        <property name="hibernate.connection.password">root</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost/
            dictionary</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.dialect">org.hibernate.dialect.
            MySQLDialect</property>

      <!-- JDBC connection pool (use the built-in)
        <property name="hibernate.connection.pool_size">1</property> -->
                <!-- Enable Hibernates automatic session context
                    management -->

         <property name="hibernate.c3p0.max_size">30</property>
         <property name="hibernate.c3p0.min_size">2</property>
         <property name="hibernate.c3p0.timeout">5000</property>
         <property name="hibernate.c3p0.max_statements">100</property>
         <property name="hibernate.c3p0.idle_test_period">300</property>
         <property name="hibernate.c3p0.acquire_increment">2</property>


        <property name="hibernate.current_session_context_class">thread</
            property>

        <!-- Disable the second-level cache  org.hibernate.cache.
            NoCacheProvider-->
        <property name="hibernate.cache.provider_class">org.hibernate.
            cache.EhCacheProvider</property>

        <!-- Echo all executed SQL to stdout -->
        <property name="hibernate.show_sql">false</property>

        <mapping resource="Dialect.hbm.xml" ></mapping>
        <mapping resource="References.hbm.xml" ></mapping>
        <mapping resource="Annotation.hbm.xml" ></mapping>
        <mapping resource="User.hbm.xml" ></mapping>
        <mapping resource="Vocab.hbm.xml" ></mapping>
        <mapping resource="Assignment.hbm.xml" ></mapping>


```

```
42
43     </session-factory>
44 </hibernate-configuration>
```

**Listing A.1:** `Hibernate.cfg.xml`

### A.1.2 Annotation.hbm.xml

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5 <hibernate-mapping package="at.jku.fim.dictionary.db">
6
7     <class name="Annotation" table="annotations" catalog="dictionary">
8       <cache usage="read-write" />
9       <id name="id" type="java.lang.Integer" unsaved-value="0">
10         <column name="id" />
11         <generator class="identity" />
12      </id>
13      <property name="annotation" type="string">
14         <column name="annotation" not-null="true">
15            <comment></comment>
16         </column>
17      </property>
18      <property name="comment" type="string">
19         <column name="comment">
20            <comment></comment>
21         </column>
22      </property>
23   </class>
24 </hibernate-mapping>
```

**Listing A.2:** `Annotation.hbm.xml`

### A.1.3 Assignment.hbm.xml

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5 <hibernate-mapping package="at.jku.fim.dictionary.db">
6   <class name="Assignment" table="assignment" catalog="dictionary">
7    <cache usage="read-write" />
8       <id name="id" type="java.lang.Integer" unsaved-value="0">
9          <column name="id" />
10         <generator class="identity" />
11      </id>
12      <property name="userId" type="java.lang.Integer">
13         <column name="userId" not-null="true">
14            <comment></comment>
15         </column>
16      </property>
```

```
17        <property name="wordId" type="java.lang.Integer">
18            <column name="wordId" not-null="true">
19                <comment></comment>
20            </column>
21        </property>
22        <property name="fullyRecorded" type="java.lang.Boolean">
23            <column name="fullyRecorded">
24                <comment></comment>
25            </column>
26        </property>
27    </class>
28 </hibernate-mapping>
```

**Listing A.3:** `Assignment.hbm.xml`

### A.1.4 Dialect.hbm.xml

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
      3.0//EN"
3 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
4 <!-- Generated 15.02.2007 15:21:29 by Hibernate Tools 3.2.0.b9 -->
5 <hibernate-mapping package="at.jku.fim.dictionary.db">
6
7     <class name="Dialect" table="dialect" catalog="dictionary">
8         <cache usage="read-write" />
9
10        <id name="id" type="int" unsaved-value="0">
11            <column name="id" />
12            <generator class="identity" />
13        </id>
14        <property name="word" type="string">
15            <column name="word" length="45" not-null="true">
16                <comment></comment>
17            </column>
18        </property>
19        <property name="lang" type="string">
20            <column name="lang" length="4">
21                <comment></comment>
22            </column>
23        </property>
24
25        <property name="phoneticId" type="string">
26            <column name="phoneticId">
27                <comment></comment>
28            </column>
29        </property>
30        <property name="annotations" type="java.lang.Integer">
31            <column name="annotations">
32                <comment></comment>
33            </column>
34        </property>
35        <array name="moreSampleSentences" cascade="all">
36            <key column="parent_id" />
```

```
37          <index column="idx" />
38
39          <one-to-many class="SampleSentences" />
40
41       </array>
42       <array name="annotationIDs" cascade="all">
43          <key column="parent_id" />
44          <index column="idx" />
45
46          <one-to-many class="IntArray" />
47
48       </array>
49       <property name="isRecorded" type="java.lang.Boolean">
50          <column name="recorded">
51             <comment></comment>
52          </column>
53       </property>
54       <property name="comment" type="string">
55          <column name="comment" length="255">
56             <comment></comment>
57          </column>
58       </property>
59    </class>
60
61
62    <class name="SampleSentences" table="samplesentences">
63       <cache usage="read-write" />
64       <id name="id" unsaved-value="0">
65          <generator class="identity" />
66       </id>
67       <property name="sentence" type="string">
68          <column name="sentence" length="255">
69             <comment></comment>
70          </column>
71       </property>
72       <property name="isRecorded" type="java.lang.Boolean">
73          <column name="recorded">
74             <comment></comment>
75          </column>
76       </property>
77    </class>
78
79    <class name="IntArray" table="integerarray">
80       <cache usage="read-write" />
81       <id name="id" unsaved-value="0">
82          <generator class="identity" />
83       </id>
84       <property name="val" type="java.lang.Integer">
85          <column name="val">
86             <comment></comment>
87          </column>
88       </property>
89    </class>
```

```
90 </hibernate-mapping>
```

**Listing A.4:** `Dialect.hbm.xml`

### A.1.5 References.hbm.xml

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5 <hibernate-mapping package="at.jku.fim.dictionary.db">
6   <class name="References" table="references_" catalog="dictionary">
7    <cache usage="read-write" />
8       <id name="id" type="java.lang.Integer" unsaved-value="0">
9          <column name="id" />
10         <generator class="increment" />
11      </id>
12      <property name="dialect_id" type="java.lang.Integer">
13         <column name="dialect_id" not-null="true">
14            <comment></comment>
15         </column>
16      </property>
17      <property name="foreign_id" type="java.lang.Integer">
18         <column name="foreign_id" not-null="true">
19            <comment></comment>
20         </column>
21      </property>
22  </class>
23 </hibernate-mapping>
```

**Listing A.5:** `References.hbm.xml`

### A.1.6 User.hbm.xml

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5 <hibernate-mapping package="at.jku.fim.dictionary.db">
6    <class name="User" table="userdata" catalog="dictionary">
7      <cache usage="read-write" />
8      <id name="id" type="java.lang.Integer" unsaved-value="0">
9         <column name="id" />
10        <generator class="identity" />
11     </id>
12     <property name="userName" type="string">
13        <column name="userName" not-null="true">
14           <comment></comment>
15        </column>
16     </property>
17     <property name="encryptedPassword" type="string">
18        <column name="encryptedPassword" not-null="true">
19           <comment></comment>
```

```
20          </column>
21       </property>
22             <property name="role" type="string">
23          <column name="role" not-null="true">
24             <comment></comment>
25          </column>
26       </property>
27             <property name="realName" type="string">
28          <column name="realName">
29             <comment></comment>
30          </column>
31       </property>
32             <property name="email" type="string">
33          <column name="email">
34             <comment></comment>
35          </column>
36       </property>
37             <property name="phone" type="string">
38          <column name="phone">
39             <comment></comment>
40          </column>
41       </property>
42       <property name="street" type="string">
43          <column name="street">
44             <comment></comment>
45          </column>
46       </property>
47       <property name="streetNr" type="string">
48          <column name="streetNr">
49             <comment></comment>
50          </column>
51       </property>
52             <property name="postCode" type="string">
53          <column name="postCode">
54             <comment></comment>
55          </column>
56       </property>
57                   <property name="city" type="string">
58          <column name="city">
59             <comment></comment>
60          </column>
61       </property>
62                   <property name="country" type="string">
63          <column name="country">
64             <comment></comment>
65          </column>
66       </property>
67       <property name="firstLanguage" type="string">
68          <column name="firstLanguage">
69             <comment></comment>
70          </column>
71       </property>
72             <property name="secondLanguage" type="string">
73          <column name="secondLanguage">
```

```
74              <comment></comment>
75          </column>
76      </property>
77              <property name="thirdLanguage" type="string">
78          <column name="thirdLanguage">
79              <comment></comment>
80          </column>
81      </property>
82  </class>
83 </hibernate-mapping>
```

**Listing A.6:** `User.hbm.xml`

### A.1.7 `Vocab.hbm.xml`

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5 <hibernate-mapping package="at.jku.fim.dictionary.db">
6 <class name="Vocab" table="vocablist" catalog="dictionary">
7       <cache usage="read-write" />
8       <id name="id" type="java.lang.Integer" unsaved-value="0">
9          <column name="id" />
10         <generator class="identity" />
11      </id>
12      <property name="userId" type="java.lang.Integer">
13         <column name="userId" not-null="true">
14            <comment></comment>
15         </column>
16      </property>
17      <property name="wordId" type="java.lang.Integer">
18         <column name="wordId" not-null="true">
19            <comment></comment>
20         </column>
21      </property>
22              <property name="listName" type="string">
23         <column name="listName">
24            <comment></comment>
25         </column>
26      </property>
27  </class>
28 </hibernate-mapping>
```

**Listing A.7:** `Vocab.hbm.xml`

### A.1.8 `ehcache.xml`

```
1 <ehcache>
2
3     <diskStore path="java.io.tmpdir"/>
4
5     <defaultCache
```

```
6           maxElementsInMemory="500"
7           eternal="false"
8           timeToIdleSeconds="300"
9           timeToLiveSeconds="1800"
10          overflowToDisk="false"
11          diskPersistent="false"
12          diskExpiryThreadIntervalSeconds="120"
13          memoryStoreEvictionPolicy="LRU"
14          />
15
16      <cache name="at.jku.fim.dictionary.db.Dialect"
17          maxElementsInMemory="3000"
18          eternal="false"
19          timeToIdleSeconds="1200"
20          timeToLiveSeconds="10000"
21          overflowToDisk="false"
22          />
23
24      <cache name="at.jku.fim.dictionary.db.SampleSentences"
25          maxElementsInMemory="3000"
26          eternal="false"
27          timeToIdleSeconds="1200"
28          timeToLiveSeconds="10000"
29          overflowToDisk="false"
30          />
31
32      <cache name="at.jku.fim.dictionary.db.References"
33          maxElementsInMemory="6000"
34          eternal="false"
35          timeToIdleSeconds="1200"
36           timeToLiveSeconds="10000"
37          overflowToDisk="false"
38          />
39
40     <cache name="at.jku.fim.dictionary.db.Annotation"
41          maxElementsInMemory="100"
42          eternal="true"
43          timeToIdleSeconds="0"
44          timeToLiveSeconds="0"
45          overflowToDisk="false"
46          />
47
48 </ehcache>
```

**Listing A.8:** `ehcache.xml`

## A.2 Jetty Konfiguration (`jetty-config.xml`)

```
1 <?xml version="1.0"  encoding="ISO-8859-1"?>
2 <!DOCTYPE Configure PUBLIC
3  "-//Mort Bay Consulting//DTD Configure 1.1//EN"
4  "http://jetty.mortbay.org/configure_1_2.dtd">
5
```

```
<Configure class="org.mortbay.jetty.Server">
<!--
Add and configure a HTTP listener to port 8081
The default port can be changed using: java -Djetty.port=80
 -->
  <Call name="addListener">
    <Arg>
      <New class="org.mortbay.http.SocketListener">
        <Set name="Port"><SystemProperty name="jetty.port" default="8081"
            /></Set>
        <Set name="MinThreads">50</Set>
        <Set name="MaxThreads">100</Set>
        <Set name="MaxIdleTimeMs">30000</Set>
        <Set name="LowResourcePersistTimeMs">5000</Set>
        <Set name="ConfidentialPort">8443</Set>
        <Set name="IntegralPort">8443</Set>
      </New>
    </Arg>
  </Call>

<!--
Add and configure a specific web application
+ Set Unpack WAR files
+ Set Default Descriptor.  Resource, file or URL
+ Set Virtual Hosts. A Null host or empty array means all hosts
 -->
  <Call name="addWebApplication">
    <Arg>/dictionary</Arg>
    <Arg>src/webapp</Arg>
  </Call>

  <!-- ============================================= -->
  <!-- Configure the Other Server Options            -->
  <!-- ============================================= -->
  <Set name="requestsPerGC">500</Set>
  <Set name="statsOn">false</Set>

</Configure>
```

**Listing A.9:** `jetty-config.xml`

## A.3 Wicket Konfiguration (`web.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <display-name>FIM Dictionary</display-name>

 <!--
```

```
10        There are three means to configure Wickets configuration mode and
              they are
11        tested in the order given.
12        1) A system property: -Dwicket.configuration
13        2) servlet specific <init-param>
14        3) context specific <context-param>
15        The value might be either "development" (reloading when templates
              change)
16        or "deployment". If no configuration is found, "deployment" is the
              default.
17 -->
18    <context-param>
19      <param-name>configuration</param-name>
20      <param-value>deployment</param-value>
21    </context-param>
22
23     <servlet>
24         <servlet-name>dictionary</servlet-name>
25         <servlet-class>wicket.protocol.http.WicketServlet</servlet-class>
26         <init-param>
27           <param-name>applicationClassName</param-name>
28           <param-value>at.jku.fim.dictionary.DictionaryApplication</param
                 -value>
29         </init-param>
30         <load-on-startup>1</load-on-startup>
31     </servlet>
32
33     <servlet-mapping>
34         <servlet-name>dictionary</servlet-name>
35         <url-pattern>/app/*</url-pattern>
36     </servlet-mapping>
37
38 </web-app>
```

**Listing A.10:** `web.xml`

# Anhang B

# Datenbank-Schema

```
1 -- MySQL Administrator dump 1.4
2 --
3 -- ------------------------------------------------------
4 -- Server version 5.0.27-community-nt
5
6
7 /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
8 /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
9 /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
10 /*!40101 SET NAMES utf8 */;
11
12 /*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
13 /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
       FOREIGN_KEY_CHECKS=0 */;
14 /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO'
       */;
15
16
17 --
18 -- Create schema dictionary
19 --
20
21 CREATE DATABASE IF NOT EXISTS dictionary;
22 USE dictionary;
23
24 --
25 -- Definition of table `annotations`
26 --
27
28 DROP TABLE IF EXISTS `annotations`;
29 CREATE TABLE `annotations` (
30   `id` int(10) unsigned NOT NULL auto_increment,
31   `annotation` varchar(4) NOT NULL,
32   `comment` varchar(45) default NULL,
33   PRIMARY KEY (`id`),
34   UNIQUE KEY `Index_anno` USING BTREE (`annotation`)
35 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
36
37 --
38 -- Dumping data for table `annotations`
39 --
40
41 /*!40000 ALTER TABLE `annotations` DISABLE KEYS */;
42 INSERT INTO `annotations` (`id`,`annotation`,`comment`) VALUES
43   (2,'pos','positive meaning'),
44   (3,'neg','negative meaning'),
45   (4,'not','you should not use this word'),
46   (6,'con','context dependent'),
47   (7,'rur','rural area'),
48   (8,'urb','urban area'),
49   (9,'col','colloquial language'),
50   (10,'med','medical area'),
51   (11,'tec','technical area'),
52   (12,'bio','biology');
53 /*!40000 ALTER TABLE `annotations` ENABLE KEYS */;
54
55
56 --
57 -- Definition of table `assignment`
58 --
59
60 DROP TABLE IF EXISTS `assignment`;
61 CREATE TABLE `assignment` (
62   `id` int(10) unsigned NOT NULL auto_increment,
63   `userId` int(10) unsigned NOT NULL default '0',
64   `wordId` int(10) unsigned NOT NULL default '0',
65   `fullyRecorded` tinyint(1) default '0',
66   PRIMARY KEY (`id`)
67 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
68
69 --
70 -- Dumping data for table `assignment`
71 --
72
73 /*!40000 ALTER TABLE `assignment` DISABLE KEYS */;
74 /*!40000 ALTER TABLE `assignment` ENABLE KEYS */;
75
76
77 --
78 -- Definition of table `dialect`
79 --
80
81 DROP TABLE IF EXISTS `dialect`;
82 CREATE TABLE `dialect` (
83   `id` int(10) unsigned NOT NULL auto_increment,
84   `word` varchar(120) character set latin1 NOT NULL,
85   `phoneticId` varchar(6) collate latin1_german1_ci default NULL,
86   `annotations` int(10) unsigned default NULL,
87   `lang` varchar(4) collate latin1_german1_ci NOT NULL,
88   `recorded` tinyint(1) NOT NULL default '0',
89   `comment` varchar(255) collate latin1_german1_ci default '',
```

```
90   PRIMARY KEY  (`id`),
91   KEY `Index_phonetic_lang` (`phoneticId`,`lang`),
92   KEY `Index_dialect` USING BTREE (`word`,`lang`)
93 ) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_german1_ci;
94
95 --
96 -- Dumping data for table `dialect`
97 --
98
99 /*!40000 ALTER TABLE `dialect` DISABLE KEYS */;
100 /*!40000 ALTER TABLE `dialect` ENABLE KEYS */;
101
102
103 --
104 -- Definition of table `integerarray`
105 --
106
107 DROP TABLE IF EXISTS `integerarray`;
108 CREATE TABLE `integerarray` (
109   `id` int(10) unsigned NOT NULL auto_increment,
110   `parent_id` int(10) unsigned default '0',
111   `idx` int(10) unsigned default '0',
112   `val` int(10) unsigned NOT NULL default '0',
113   PRIMARY KEY  (`id`),
114   KEY `Index_2` (`parent_id`,`idx`)
115 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
116
117 --
118 -- Dumping data for table `integerarray`
119 --
120
121 /*!40000 ALTER TABLE `integerarray` DISABLE KEYS */;
122 /*!40000 ALTER TABLE `integerarray` ENABLE KEYS */;
123
124
125 --
126 -- Definition of table `references_`
127 --
128
129 DROP TABLE IF EXISTS `references_`;
130 CREATE TABLE `references_` (
131   `id` int(10) unsigned NOT NULL auto_increment,
132   `dialect_id` int(10) unsigned NOT NULL,
133   `foreign_id` int(10) unsigned NOT NULL,
134   PRIMARY KEY  (`id`),
135   KEY `Index_reference` USING BTREE (`dialect_id`,`foreign_id`),
136   KEY `Index_reference2` (`foreign_id`,`dialect_id`)
137 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
138
139 --
140 -- Dumping data for table `references_`
141 --
142
143 /*!40000 ALTER TABLE `references_` DISABLE KEYS */;
```

```
144 /*!40000 ALTER TABLE `references_` ENABLE KEYS */;
145
146
147 --
148 -- Definition of table `samplesentences`
149 --
150
151 DROP TABLE IF EXISTS `samplesentences`;
152 CREATE TABLE `samplesentences` (
153   `id` int(10) unsigned NOT NULL auto_increment,
154   `parent_id` int(10) unsigned default '0',
155   `sentence` varchar(255) NOT NULL default '',
156   `idx` int(10) unsigned default '0',
157   `recorded` tinyint(1) NOT NULL default '0',
158   PRIMARY KEY (`id`),
159   KEY `Index_parentId` (`parent_id`,`idx`)
160 ) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Array holding the sample
      sentences';
161
162 --
163 -- Dumping data for table `samplesentences`
164 --
165
166 /*!40000 ALTER TABLE `samplesentences` DISABLE KEYS */;
167 /*!40000 ALTER TABLE `samplesentences` ENABLE KEYS */;
168
169
170 --
171 -- Definition of table `userdata`
172 --
173
174 DROP TABLE IF EXISTS `userdata`;
175 CREATE TABLE `userdata` (
176   `id` int(10) unsigned NOT NULL auto_increment,
177   `userName` varchar(45) NOT NULL,
178   `encryptedPassword` varchar(45) NOT NULL,
179   `role` varchar(45) NOT NULL,
180   `realName` varchar(45) default '',
181   `email` varchar(45) default '',
182   `phone` varchar(45) default '',
183   `street` varchar(45) default '',
184   `streetNr` varchar(45) default '',
185   `postCode` varchar(45) default '',
186   `city` varchar(45) default '',
187   `country` varchar(45) default '',
188   `firstLanguage` varchar(45) default '',
189   `secondLanguage` varchar(45) default '',
190   `thirdLanguage` varchar(45) default '',
191   PRIMARY KEY (`id`),
192   KEY `Index_1` USING BTREE (`userName`,`encryptedPassword`)
193 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
194
195 --
196 -- Dumping data for table `userdata`
```

```
197 --
198
199 /*!40000 ALTER TABLE `userdata` DISABLE KEYS */;
200 /*!40000 ALTER TABLE `userdata` ENABLE KEYS */;
201
202
203 --
204 -- Definition of table `vocablist`
205 --
206
207 DROP TABLE IF EXISTS `vocablist`;
208 CREATE TABLE `vocablist` (
209   `id` int(10) unsigned NOT NULL auto_increment,
210   `userId` int(10) unsigned NOT NULL,
211   `wordId` int(10) unsigned NOT NULL,
212   `listName` varchar(45) default 'my words',
213  PRIMARY KEY (`id`),
214  KEY `Index_listName` USING BTREE (`listName`,`wordId`,`userId`),
215  KEY `Index_1` USING BTREE (`userId`,`wordId`)
216 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
217
218 --
219 -- Dumping data for table `vocablist`
220 --
221
222 /*!40000 ALTER TABLE `vocablist` DISABLE KEYS */;
223 /*!40000 ALTER TABLE `vocablist` ENABLE KEYS */;
224
225
226
227
228 /*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
229 /*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
230 /*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
231 /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
232 /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
233 /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
234 /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

**Listing B.1:** SQL-Dump

# Literaturverzeichnis


[Amb07] Ambler, Scott W.: *The Object-Relational Impedance Mismatch*. AgileData, 2007. http://www.agiledata.org/essays/impedanceMismatch.html (letzter Aufruf: 4.8.07).

[Bec07] Becker, Peter: *Information Retrieval*. Unterlagen zur Vorlesung, 2007. FH Bonn-Rhein-Sieg, www2.inf.fh-rhein-sieg.de/~pbecke2m/retrieval/textalgorithmen1.pdf (letzter Aufruf: 27.7.07).

[Ber02] Bergsten, Hans: *JavaServer Pages*. O'Reilly, 2. Auflage, 2002. ISBN: 0-596-00317-X.

[Bha95] Bhatti, Saleem: *Channel coding — Hamming distance*, 1995. http://www.cs.ucl.ac.uk/staff/S.Bhatti/D51-notes/node30.html (letzter Aufruf: 27.7.07).

[BK07] Bauer, Christian und Gavin King: *Java Persistence with Hibernate*. Manning Publications Co., 2007. ISBN: 1-932394-88-5.

[CjK03] Choi, Key-Sun und Byung ju Kang: *Phonetic distance calculation method for similarity comparison between phonetic transcriptions of foreign words*. United States Patent 6581034, 2003. http://www.patentstorm.us/patents/6581034-fulltext.html (letzter Aufruf: 31.7.07).

[CLJ+05] Chopra, Vivek, Sing Li, Rupert Jones, Jon Eaves und John T. Bell: *Beginning JavaServer Pages*. Wiley Publishing Inc., 2005. ISBN: 0-7645-7485-X.

[Col07] Colton, Simon: *Introduction to Bioinformatics*. Department of Computing, Imperial College, London, 2007. www.doc.ic.ac.uk/~sgc/teaching/341/lecture3.ppt (letzter Aufruf: 3.8.07).

[CPS05] Cha, ByungRae, KyungWoo Park und JaeHyun Seo: *Neural Network Techniques for Host Anomaly Intrusion Detection Using Fixed Pattern Transformation*. In: *Computational Science and Its Applications*, Lecture Notes in Computer Science, Seiten 254–263. Springer-Verlag Berlin Heidelberg, 2005.

[DH07] Dashorst, Martijn und Eelco Hillenius: *Wicket in Action*. Manning Publications, Manning Early Access Program (MEAP) Auflage, 2007. `http://manning.com/dashorst/meap_wicketch1.pdf` (letzter Aufruf: 16.8.07).

[Dia05] Diaz, Dustin: *Sweet Titles — JavaScript Fading Tooltips*, 2005. `http://www.dustindiaz.com/sweet-titles` (letzter Aufruf: 25.8.07).

[Fri06] Frischalowski, Dirk: *Gewichtige Anmerkungen — Annotations - Neuerung der J2SE 5*. JavaMagazin, April 2006. `http://javamagazin.de/itr/online_artikel/psecom,id,659,nodeid,11.html` (letzter Aufruf: 5.9.07).

[Gee07] Geertzen, Jeroen: *Edit Distance*, 2007. `http://www.cosmion.net/jeroen/software/ed/` (letzter Aufruf: 27.7.07).

[Gle07] Gleis, Stefan: *BeoLingus*. TU Chemnitz, 2007. `http://dict.tu-chemnitz.de/` (letzter Aufruf: 1.8.07).

[Gur06] Gurumurthy, Karthik: *Pro Wicket*. Apress Inc., 2006. ISBN: 978-1-59059-722-4.

[Hem07] Hemetsberger, Paul: *dict.cc English-German dictionary*, 2007. `http://www.dict.cc/` (letzter Aufruf: 1.8.07).

[Hir97] Hirschberg, D. S.: *Serial computations of Levenshtein distances*. In: Apostolico, A. und Z. Galil (Herausgeber): *Pattern matching algorithms*, Seiten 123–141. Oxford University Press, 1997.

[Hun98] Hunter, Jason: *com.oreilly.servlet.HttpMessage*, 1998. `http://www.porcupyne.org/docs/browse_source/JavaServlet/com/oreilly/servlet/HttpMessage.java.html` (letzter Aufruf: 16.8.07).

[Hyy03] Hyyrö, Heikki: *Practical Methods for Approximate String Matching*. Doktorarbeit, Department of Computer Science, University of Tampere, 2003.

[JF06] Jacobi, Jonas und John R. Fallows: *Pro JSF and Ajax — Building Rich Internet Components*. Apress, 2006. ISBN: 978-1-59059-580-0.

[KPRR06] Kappel, Gerti, Birgit Pröll, Siegfried Reich und Werner Retschitzegger (Herausgeber): *Web Engineering — The Discipline of Systematic Development of Web Applications*. John Wiley & Sons Ltd., 2006. ISBN: 978-0-470-01554-4.

[Kur02] Kurniawan, Budi: *Java for the Web with Servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions*. New Riders Publishing, 2002. ISBN: 0-7357-1195X.

[LS07] Lowagie, Bruno und Paulo Soares: *iText, a Free Java-PDF Library*, 2007. `http://www.lowagie.com/iText/` (letzter Aufruf: 23.8.07).

[LTH06] Locke, Jonathan, Chris Turner und Eelco Hillenius: *Wicket 1.2.6 API Javadoc*, 2006. `http://wicketframework.org/apidocs/index.html` (letzter Aufruf: 10.7.07).

[Mah03] Mahmoud, Qusay H.: *Servlets and JSP Pages Best Practices*. Sun Microsystems Inc., March 2003. `http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/` (letzter Aufruf: 13.7.07).

[N.N06a] N.N.: *The Jakarta Projekt — Commons Codec*, 2006. `http://jakarta.apache.org/commons/codec/userguide.html` (letzter Aufruf: 19.7.07).

[N.N06b] N.N.: *LAME MP3 Encoder — The Lame Project*. Sourceforge.net, 2006. `http://lame.sourceforge.net/index.php` (letzter Aufruf: 17.8.07).

[N.N06c] N.N.: *Wicket Models*, 2006. `http://www.wicket-wiki.org.uk/wiki/index.php/Models` (letzter Aufruf: 10.7.07).

[N.N06d] N.N.: *Wimpy MP3 Button*, 2006. `http://www.wimpyplayer.com/products/wimpy_button.html` (letzter Aufruf: 25.8.07).

[N.N07a] N.N.: *HIBERNATE - Relational Persistence for Idiomatic Java*, 2007. `http://www.hibernate.org/hib_docs/v3/reference/en/html/index.html` (letzter Aufruf: 6.8.07).

[N.N07b] N.N.: *Java Community Process — FAQ*, 2007. `http://www.jcp.org/en/introduction/faq` (letzter Aufruf: 13.8.07).

[N.N07c] N.N.: *Java Media Framework (JMF) FAQs*. Sun Developer Network, 2007. `http://java.sun.com/products/java-media/jmf/reference/faqs/index.html`.

[N.N07d] N.N.: *(Ober)österreichisch-Deutsch-Englisches Wörterbuch*. Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM), 2007. `http://www.fim.uni-linz.ac.at/Woerterbuch_oesterr_deut_englisch.htm` (letzter Aufruf: 30.8.07).

[N.N07e] N.N.: *script.aculo.us — web 2.0 javascript*, 2007. `http://script.aculo.us/` (letzter Aufruf: 25.8.07).

[N.N07f] N.N.: *Wicket Examples — Component Reference*. Wicket-Stuff, 2007. `http://wicketstuff.org/wicket13/compref/` (letzter Aufruf: 7.9.07).

[PH06] Peak, Patrick und Nick Heudecker: *Hibernate Quickly*. Manning Publications Co., 2006.

[Phi03] Philips, Lawrence: *The Double Metaphone Search Algorithm*. Dr. Dobb's Portal, 2003. `http://www.ddj.com/dept/cpp/184401251;?pgno=2` (letzter Aufruf: 19.7.07).

[Raj07a] Raja, Shunmuga: *Introduction to Java Persistence API(JPA)*. JavaBeat, April 2007. `http://www.javabeat.net/javabeat/ejb3/articles/2007/04/introduction_to_java_persistence_api_jpa_ejb_3_0_1.php` (letzter Aufruf: 4.8.07).

[Raj07b] Raja, Shunmuga: *Struts 2.0 Introduction and Validations using Annotations*. JavaBeat, Mai 2007. `http://struts.javabeat.net/articles/2007/05/struts-2-0-introduction-annotation-validations/` (letzter Aufruf: 10.8.07).

[Rep02] Repici, Dominic John: *Understanding Classic SoundEx Algorithms — Search Names & Phrases Based on Phonetic Similarity*. Creativyst Inc., 2002. `http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm` (letzter Aufruf: 19.7.07).

[Rie07] RIETHMAYER, HANS: *LEO — Online-Wörterbücher*, 2007. `http://www.leo.org/` (letzter Aufruf: 31.7.07).

[Roo06] ROOK, STEFAN: *Java-Persistenz mit Hibernate*, 2006. `http://www.it-agile.de/fileadmin/docs/Hibernate-Newsletter.pdf` (letzter Aufruf: 7.8.07).

[Rou06] ROUGHLEY, IAN: *Starting Struts 2*. C4Media Inc., 2006. ISBN: 978-1-4303-2033-3, `http://infoq.com/minibooks/starting-struts2` (letzter Aufruf: 9.8.07).

[Röp05] RÖPKE, JÖRG: *Sprachunterstützung für webbasierte interaktive Anwendungen im e-Learning*. Diplomarbeit, Fachhochschule Trier, Angewandte Informatik, 2005. `www.ainformatik.fh-trier.de/~schneider/ausarbeitungen/RoepkeDiplomarbeit.pdf` (letzter Aufruf: 27.7.07).

[Rus07] RUSSWURM, ROLAND: *Sprache in Österreich — Ostarrichi.org*, 2007. `http://www.ostarrichi.org/` (letzter Aufruf: 31.7.07).

[Sch05a] SCHILDT, HERBERT: *Java: A Beginner's Guide*. McGraw-Hill/Osborne, 4. Auflage, 2005. ISBN: 978-0-07-226384-8.

[Sch05b] SCHMID, KARL: *Web-Frameworks bei der Entwicklung von Web-Applikationen anhand von Apache Struts und JavaServer Faces*. Diplomarbeit, Fachhochschule Köln, 2005. `http://www.gm.fh-koeln.de/~faeskorn/diplom/diplom_schmidt.pdf` (letzter Aufruf: 11.8.07).

[SH06] SHAN, TONY C und WINNIE W HUA: *Taxonomy of Java Web Application Frameworks*. In: *IEEE International Conference on e-Business Engineering (ICEBE'06)*, Seiten 378–385, 2006.

[Smi07] SMITH, GLEN: *OpenCSV — AN open source csv parser for Java*, 2007. `http://opencsv.sourceforge.net/` (letzter Aufruf: 25.8.07).

[Sur07] SURESH: *Suresh Online – Essential Developer Resources*, 2007. `http://personal.vsnl.com/sureshms/javasign1.html` (letzter Aufruf: 2.8.07).

[Whi04] WHITE, TOM: *Can't beat Jazzy — Introducing the Java platform's Jazzy new spell checker API*. IBM, 2004. `http://www.ibm.com/developerworks/java/library/j-jazzy/` (letzter Aufruf: 19.7.07).

[Wil03] Williams, Tim: *Tims Servlet Tutorial*. University of Birmingham, May 2003. `http://www.cs.bham.ac.uk/~tmw/servlet_tutorial/helloworld.shtml` (letzter Aufruf: 16.7.07).

# Curriculum Vitae

## Persönliche Daten

| | |
|---|---|
| Name: | Simon Kohlberger, Bakk.techn. |
| Anschrift: | Loibersdorf 25 |
| | 4210 Unterweitersdorf |

## Schulbildung

| | |
|---|---|
| 1987 - 1991 | Volksschule in Unterweitersdorf |
| 1991 - 1995 | Hauptschule in Gallneukirchen |
| 1995 - 2000 | HTL-Leonding, Abteilung für Elektronik und Nachrichtentechnik |

## Studium

| | |
|---|---|
| 2001 - 2006 | Bakkalaureatsstudium der Informatik an der Johannes Kepler Universität in Linz |
| 9/2005 - 3/2006 | Auslandssemester an der University of Reading in England. Teilnahme am Masterstudiengang *Network Centred Computing* |
| Seit 2006 | Magisterstudium der Informatik an der Johannes Kepler Universität in Linz |

## Berufliche Aktivitäten

| | |
|---|---|
| 8/1997 | Chemserv Industrie Service GmbH: Praktikum |
| 8/1999 | Telekom Austria AG: EDV-Techniker (Ferialarbeit) |
| 6/2001 - 9/2001 | VA Tech ELIN EBG: Elektriker |
| 6/2003 - 11/2003 | Schachermayer GmbH: Projektmitarbeiter (IT-Abteilung) |
| 8/2004 | Linz AG Abfallwirtschaft (Ferialarbeit) |
| 10/2004 - 1/2005 | JKU Linz, Institut für Telekooperation: Tutor in Softwareentwicklung |
| 8/2005 - 9/2005 | TenCate Geosynthetics GmbH: Anlagenbetreuer (Ferialarbeit) |
| 4/2006 - 9/2006 | Hagenberg Software GmbH: Softwareentwickler (vollbeschäftigt) |
| 2/2002 - 8/2007 | Technische Akustik Schreiner: EDV-Messtechniker (zusätzlich geringfügig beschäftigt neben dem Studium) |

## Präsenzdienst

| | |
|---|---|
| 9/2000 - 5/2001 | 2. Jägerkompanie in Freistadt |

# Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Magisterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Des weiteren versichere ich, dass ich diese Magisterarbeit weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Linz, am 15. Oktober 2007

Simon Kohlberger