

# Möglichkeiten zur Entwicklung eines etwas flexibleren Vokabeltrainers

MAS-Arbeit

von

Hanspeter Siegfried  
Matr.-Nr. 81-711-582  
Hegianwandweg 49  
8045 Zürich  
siegfried@si-gi.ch

MAS Informatik für Gymnasien  
Universität Fribourg  
Departement für Informatik  
Information System Research Group  
Zürich, 3. Dezember 2010

1. Referent: Prof. Dr. Andreas Meier
  2. Referent: Dr. Martin Guggisberg, Uni Basel
- Betreuer: Edy Portmann

Diese Arbeit befasst sich mit der Frage, ob und wie es möglich ist, einen Vokabeltrainer zu implementieren, der einerseits die Eingaben der Benutzerin maschinell überprüft, der diese Überprüfung aber andererseits nicht durch den blossen Vergleich zweier Textketten bewerkstelligt. Es wird gezeigt, dass sich eine tippfehler-tolerante Überprüfung der Eingaben gut realisieren lässt. Dafür gibt es schon seit längerem vielfach erprobte Konzepte. Schwieriger ist die maschinelle Beurteilung der Frage, ob eine Benutzereingabe als Synonym einer richtigen Wortbedeutung akzeptiert werden kann. Ein solches Urteil ist semantischer Art, und semantische Fragen sind nur im Zusammenhang von Texten genau zu beurteilen. In aller Regel stehen einem Vokabeltrainer aber nicht Texte, sondern einzelne Wörter zur Verfügung.

Eine Möglichkeit Synonyme zu identifizieren, die allerdings nicht ohne menschliche Eingriffe auskommt, wird detailliert beschrieben; weitere, technisch anspruchsvollere Möglichkeiten werden skizziert.

Schliesslich wird ein Prototyp beschrieben, in dem die oben erklärte Funktionalität implementiert ist, und es wird erwogen, wie der Prototyp zur Release-Reife gebracht werden könnte.

Stichworte: Computerlinguistik, Vokabeltrainer, Fehlertoleranz, Synonym

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>5</b>
<b>2. Effizientes Vokabeltraining</b>	<b>6</b>
2.1. Lernen und Vergessen . . . . .	6
2.2. Die Entwicklung von Vokabeltrainern 1990 bis 2010 . . . . .	7
2.2.1. Das «Abfragetool» . . . . .	7
2.2.2. Das «Repetitionstool» . . . . .	7
2.3. Anforderungen an «Vokabeltrainer der 3. Generation» (Lastenheft) . . . . .	8
2.3.1. Trainings- und Abfragemodi . . . . .	8
2.3.2. Lern-Management . . . . .	8
2.3.3. Fehlertoleranz . . . . .	8
2.3.4. Synonym-Fähigkeit . . . . .	9
2.3.5. Einbindung in eine Schul-Infrastruktur . . . . .	9
2.3.6. Einfache Administration der Wörterdatenbank . . . . .	10
2.3.7. Plattform-Unabhängigkeit . . . . .	10
<b>3. Linguistische und technische Grundlagen</b>	<b>11</b>
3.1. «Meinten Sie...» – Wie erkennt ein Computer Tippfehler? . . . . .	11
3.1.1. Arten von Tippfehlern . . . . .	11
3.1.2. Buchstabendreher . . . . .	11
3.1.3. Fehlende, überzählige oder falsche Buchstaben . . . . .	12
3.1.4. Phonetische Suche mit Soundex-Algorithmus und Kölner Phonetik . . . . .	12
3.1.5. Levenshtein-Distanz zwischen Strings . . . . .	12
3.1.6. Fazit: Tippfehler-Erkennung: Kombination von mehreren Verfahren . . . . .	13
3.2. Synonyme erkennen . . . . .	14
3.2.1. Entwicklung des Wortschatzes einer natürlichen Sprache . . . . .	14
3.2.2. Was ist ein Synonym? . . . . .	16
3.2.3. Fazit: Schwierigkeiten im maschinellen Umgang mit Synonymen . . . . .	17
3.2.4. Anforderungen an eine Datenquelle zum Suchen von Synonymen . . . . .	18
3.2.5. Wikipedia . . . . .	20
3.2.6. Wiktionary . . . . .	20
3.2.7. Open Thesaurus . . . . .	21
3.2.8. Fazit: Synonymsuche mit OpenThesaurus . . . . .	21
3.2.9. Problem: Der Kontext . . . . .	21

<b>4. Implementierung eines Prototypen</b>	<b>23</b>
4.1. Die Trainingsmodi . . . . .	23
4.2. Grundsätzliche Design-Entscheide . . . . .	24
4.2.1. Gewählte Programmiersprache . . . . .	24
4.2.2. Data Base Management System . . . . .	25
4.2.3. Grafik . . . . .	25
4.3. Überblick über die Komponenten des Prototypen . . . . .	25
4.4. Was schon vor Beginn dieser Arbeit vorhanden war . . . . .	26
4.4.1. Datenbank . . . . .	26
4.4.2. PHP-Programme . . . . .	26
4.5. Der Aufbau der Datenbank . . . . .	27
4.5.1. Tabellen mit Vokabel-Daten . . . . .	27
4.5.2. Modellierung von Beziehungen zwischen Wörtern . . . . .	27
4.5.3. Tabellen mit Benutzerdaten und Lernkontrolle . . . . .	32
4.6. Die Übungssteuerung . . . . .	32
4.7. Modellierung einer Vokabel als PHP-Klasse . . . . .	34
4.8. Prüfung von Wortbedeutungen: Die Klasse Pruefer . . . . .	35
4.9. Verfahren zur tippfehler-toleranten Prüfung von Eingaben . . . . .	35
4.9.1. Erkennung von Buchstabendrehern, Vergleich nach Kölner Phonetik: Die Klasse Alphabetizer . . . . .	35
4.9.2. Identifikation weiterer Tippfehler . . . . .	37
4.10. Synonyme suchen: Die Klasse Synonymfinder . . . . .	39
4.10.1. Synonym-Abfrage bei OpenThesaurus.de . . . . .	39
4.10.2. Eine Schwierigkeit: Konsistente Zeichensatzcodierung . . . . .	41
4.11. Web Interface . . . . .	43
4.11.1. Startseite, Login, Navigation . . . . .	43
4.11.2. Wörter suchen, anzeigen, exportieren . . . . .	43
4.11.3. Wörter trainieren . . . . .	43
4.11.4. Datenbank-Administration . . . . .	44
4.12. Schnittstellen zu anderen Datenbanken . . . . .	44
<b>5. Zusammenfassung und Ausblick</b>	<b>46</b>
5.1. Grundsätzliches . . . . .	46
5.2. Weiterentwicklung des Prototypen . . . . .	46
<b>A. Anhang</b>	<b>48</b>
A.1. Beilagen . . . . .	49
A.2. website . . . . .	49
Literatur . . . . .	53
Abbildungsverzeichnis . . . . .	54
Codeverzeichnis . . . . .	55

# 1. Einleitung

Vokabeltrainer sind etwas aus der Mode gekommen. In den 90er Jahren, als Computer in den Schulen langsam «Einzug hielten», war das anders: Kaum ein Verlag gab ein Sprachlehrbuch heraus, ohne ihm zumindest die Demoversion eines Vokabeltrainers beizulegen, der auf das Buch zugeschnitten war. Bald aber merkte man, dass die Vokabeltrainer die Versprechen, mit denen sie angepriesen wurden, nicht zu halten vermochten:

Auf Seiten der Verlage musste man erkennen, dass Software – zumindest grafisch – sehr schnell veraltet, wenn man sie nicht permanent für teures Geld weiterentwickeln lässt.

Auf Seiten der Nutzerinnen und Nutzer war nach anfänglicher Begeisterung für computerunterstütztes Lernen rasch eine Abkühlung spürbar. Die Tatsache allein, dass man am Computer lernen konnte, steigerte die Motivation nur kurz. Die damals verfügbaren Programme waren zu starr und technisch und grafisch oft schon beim Erscheinen nicht auf dem Stand dessen, was sich junge Computeranwender gewohnt waren.

Der Einsatz von Computern kann die Motivation beim Lernen durchaus steigern, die Voraussetzung dafür ist aber, dass das Lernen dank dem Computer leichter vonstatten geht – und das war mit den Vokabeltrainern der ersten Generation nicht immer gegeben.

Das Ziel dieser Arbeit ist es, zu prüfen, ob und wie unter Nutzung aktueller Web-Techniken eine Weiterentwicklung zu «Vokabeltrainern der 3. Generation» möglich ist. Die beschriebenen technische Neuerungen werden zudem grösstenteils in einem Prototypen implementiert.

Im Kapitel 2 wird kurz erläutert, was aus Sicht der Lerntheorie den Erfolg beim Vokabellernen fördern kann, danach werden kurz die zwei wichtigsten Typen von Vokabeltrainern beschrieben, die seit den 1990er Jahren auf den Markt gekommen sind. Aus deren Mängeln wird ein Anforderungsprofil für einen neuen Vokabeltrainer entwickelt: Wesentlich ist einerseits das Management der Repetitionen jeder Vokabel, andererseits eine grössere Flexibilität der Software im Prüfen von Benutzereingaben. Kapitel 3 ist den technischen und linguistischen Grundlagen einer flexibleren Überprüfung von Benutzereingaben gewidmet. Kapitel 4 beschreibt die wichtigsten Komponenten des erstellten Prototypen und Kapitel 5 zieht Bilanz über das Erreichte und skizziert, wohin der Weg führen könnte.

## 2. Effizientes Vokabeltraining

### 2.1. Lernen und Vergessen

Schon seit dem 19. Jahrhundert ist die Ebbinghaus'sche Vergessenskurve[34] bekannt: Die Menschen vergessen, was sie lediglich einmal gelernt haben, sehr schnell wieder. Wird das Gelernte in bestimmten Zeitabständen repetiert, bleibt es deutlich länger in Erinnerung. In den 1970er Jahren wies [Ves78, 43 – 68] unter Beizug von Ergebnissen der Hirnphysiologie nach, dass neu erworbene Lerninhalte mehrmals repetiert werden müssen, damit sie ins Langzeitgedächtnis gelangen – und zwar in anfangs sehr kurzen, dann wachsenden Zeitabständen. Diese Erkenntnis ist seither in unzähligen Didaktik- und Lerntrainingsbüchern (z.B. [End08]) aufgenommen und angewandt worden.

Als «Management-System», um Vokabeln mehrmals in wachsenden Zeitabständen zu repetieren, reicht schon eine Vokabelkartei aus, Generationen von Schülerinnen und Schülern haben sie mit mehr oder weniger Erfolg angewandt. Vokabelkarteien haben aber zwei gravierende Nachteile:

- Es braucht recht viel Zeit und Disziplin, eine Vokabelkartei sauber zu führen, sie vollständig, geordnet und aktuell zu halten. Die Zeit und Energie, die eine Schülerin hierfür aufwendet, kommt nicht zwingend dem Lernen selbst zugute.
- Üblicherweise werden Vokabelkarteien in Bereiche wie «neu», «1-mal gekonnt» etc. unterteilt. Dabei wird die Zeit, die seit der letzten Repetition einer Vokabel verstrichen ist, nicht berücksichtigt. Es ist recht schwierig, sicherzustellen, dass eine Vokabel nach dem ersten Lernen 1 Tag später, danach 3 Tage, 6 Tage, 10 Tage etc. später wieder abgefragt wird. Auch hier absorbiert die Verwaltung der Kartei Zeit und Energie, die besser fürs Lernen selbst zur Verfügung stünden.

[Tha06] betont, dass eine grosse Chance des e-Learnings darin bestehe, dass das Management dieser Repetitionen von einer Software übernommen wird: Dem Benutzer wird dieselbe Vokabel mindestens sechsmal zur Repetition vorgelegt, wobei die Abstände zwischen den Repetitionen wachsen: Anfangs beträgt der Abstand maximal einen Tag, gegen Ende mehrere Monate. [WDH<sup>+</sup>08] stellen fest, dass vor allem beim Auswendiglernen und Üben von Vokabeln und Formen eine Unterstützung durch Computerprogramme hilfreich sein kann, da die Lernenden schnell Feedback über ihren Lernerfolg erhalten.

## 2.2. Die Entwicklung von Vokabeltrainern 1990 bis 2010

Im folgenden sollen kurz die Typen von Vokabeltrainern beschrieben werden, die seit den 1990er Jahren auf den Markt gebracht wurden.

### 2.2.1. Das «Abfragetool»

Die allerersten Vokabeltrainer enthielten meistens das Vokabular eines Lehrbuches. Sie ermöglichten es, sich ein bestimmtes Pensum zum Einprägen anzeigen und danach abfragen zu lassen. Als Darstellungsmetapher diente oft die Karteikarte. Beim Abfragen musste die Benutzerin die zu trainierenden Inhalte (Wortbedeutung, Formen) in Eingabefelder eintippen, das Programm verglich die Benutzereingaben danach ganz einfach mit den gespeicherten Daten. In der Regel war es möglich, sich eine Liste der noch nicht beherrschten Vokabeln auszudrucken.

Die Nachteile dieser Art Trainingssoftware liegen auf der Hand:

- Da die Beurteilung der Benutzereingaben auf einem einfachen String-Vergleich basiert, ist eine Lösung, die einen Tippfehler enthält, «gleich falsch» wie ein falsches Wort.
- Wenn mehrere Wortbedeutungen abgefragt werden und diese (was damals üblich war) in der Datenbank als eine einzige fortlaufende Textkette gespeichert sind, wird schon eine Eingabe der richtigen Bedeutungen in veränderter Reihenfolge als falsch gewertet.
- Wenn die Benutzerin ein (passendes) Synonym einer gespeicherten Wortbedeutung eingibt, wird auch diese Eingabe als falsch gewertet.

Trainingssoftware dieser Art ist mithin zu starr, die Aufmerksamkeit der lernenden Person wird zu sehr durch das Eintippen dessen, «was die Maschine erwartet» in Anspruch genommen.

### 2.2.2. Das «Repetitionstool»

Wegen der erheblichen Nachteile der «Abfragetools» sind in der Folge die meisten Entwickler von Vokabel-Trainingssoftware von der Idee abgerückt, dass der Computer die Lösungsvorschläge des Benutzers prüfen soll. In der Regel wird ihm nun eine Frage präsentiert, auf die er sich die richtige Antwort überlegen oder in ein Textfeld notieren soll. Auf Knopfdruck wird die gespeicherte Antwort angezeigt, und der Benutzer gibt (ebenfalls per Knopfdruck) selbst an, ob die gedachte Lösung richtig war. Auf diese Weise können das Tippfehler- und das Synonym-Problem gleichsam an den Benutzer zurück-delegiert werden.

Zusätzlich führen Programme dieser Art neu ein *Lern-Management* wie oben beschrieben ein. Neuere Vokabeltrainer wie etwa Phase6<sup>1</sup> oder babbel<sup>2</sup> «führen» über die Repetitionen

<sup>1</sup><http://www.phase-6.ch/opencms/was-ist-phase-6/wissenschaftlicher-hintergrund/>

<sup>2</sup><http://www.babbel.com>

jeder Vokabel «Buch»; sie fordern die Nutzerin in jeder Trainingssitzung auf, die fälligen Lerninhalte zu repetieren.

Das ist ein guter Ansatz, aber auch diese Vorgehensweise hat Nachteile:

- Das Problem, dass Tippfehler als «richtige» Fehler angesehen werden, ist nicht gelöst, sondern wird umgangen.
- Weil der Nutzer zwingend selbst entscheidet, ob seine Lösung richtig ist, kann diese Art von Software nicht für (Selbst-) Tests benutzt werden.

## **2.3. Anforderungen an «Vokabeltrainer der 3. Generation» (Lastenheft)**

Ein Vokabeltrainer sollte folgende Anforderungen erfüllen:

### **2.3.1. Trainings- und Abfragemodi**

Der Benutzer soll wählen können, ob er die zu trainierenden Wörter wie Karteikarten durchblättern oder sich abfragen lassen will. Beim Abfragen sollten die Eingaben des Benutzers durch das System geprüft werden; der Benutzer soll genaues Feedback erhalten, welche Inhalte er bereits beherrscht und welche noch nicht.

### **2.3.2. Lern-Management**

Die Wörter sollen wiederholt und jeweils bei Erfolg in wachsenden Abständen abgefragt werden (vgl. Kap. 2.1).

### **2.3.3. Fehlertoleranz**

Die Benutzerin sollte beim Abfragen wählen können,

- ob die Orthographie ihrer Angaben strikte geprüft werden soll
- oder ob Tippfehler, solange sie nicht sinnverfälschend sind, toleriert werden sollen.

Die erste Variante ist z.B. dann sinnvoll, wenn das Schreiben fremdsprachlicher Wörter geübt werden soll, die zweite eher dann, wenn zu fremdsprachlichen Wörtern deutsche Bedeutungen abgefragt werden.

### 2.3.4. Synonym-Fähigkeit

Es kommt beim Abfragen von Vokabeln regelmässig vor, dass ein Nutzer als Wortbedeutung ein Wort angibt, das der Trainer nicht als Bedeutung der betreffenden Vokabel «kennt», das aber als Synonym einer gespeicherten Bedeutung ebenso «richtig» ist. Die Software soll, wenn irgend möglich, auch derartige Lösungen als richtig anerkennen. Das heisst, dass sie fähig sein muss, Synonyme als solche zu erkennen, auch wenn die zugrunde liegende Datenbank diese Information noch nicht enthält.

### 2.3.5. Einbindung in eine Schul-Infrastruktur

Vokabeln werden häufig in Sprachkursen und mit Lehrmitteln gelernt. Es kann deshalb vorteilhaft sein, wenn ein Vokabeltrainer durch Anbindung an Datenbanken auf folgende Information zugreift:

- Die Zugehörigkeit von Vokabeln zu den Lernpensen bestimmter Lektionen bestimmter Lehrbücher
- Die Zugehörigkeit der Benutzerin zu einer Lerngruppe
- Die Zuteilung bestimmter Lernpensen zu einer Lerngruppe

Die erste der obigen Anforderungen (Information über die Zugehörigkeit einer Vokabel zu bestimmten Lernpensen) hat nur technische Implikationen. Ihre Realisierung wird deshalb nicht auf Widerstand vonseiten der Nutzerinnen und Nutzer stossen. Die weiteren Punkte sind etwas heikler: Es ist zwar praktisch, wenn die Vokabelpensen der Schülerinnen und Schüler in einer Datenbank gespeichert und einfach verwaltet werden können. Wird eine solche Lösung angestrebt, muss sie aber sorgfältig eingeführt werden. Besonders wichtig ist, dass alle Betroffenen wissen, wer zu ihren Daten welchen Zugriff hat. Zudem muss die Software sorgfältig dokumentiert werden; wie bei jeder Anwendung von ICT in der Schule ist eine behutsame Schulung der Lehrpersonen entscheidend. Schliesslich muss die Nutzung eines solchen Vokabeltrainers für die Schülerinnen und Schüler selbstverständlich freiwillig bleiben.

Wird ein Vokabeltrainer wie oben skizziert an eine Schul-Datenbank angebunden, so ist es möglich, den Schülerinnen und Schülern einer Lerngruppe ein Pensum zuzuteilen, das diese dann in ihrem individuellen Rhythmus trainieren.

Für die Benutzung der Datenbank des Vokabeltrainers sind folgende Rollen und Berechtigungen sinnvoll:

- *Schülerin / Schüler* mit folgenden Rechten:
  - Beliebige Lernpensen über das Web-Interface anzeigen
  - In der Wörterdatenbank suchen (über das Web-Interface)

- Vokabeln über das Web-Interface oder ein anderes GUI trainieren. Dabei werden Daten zum Trainingserfolg gespeichert.
- Die Daten zum eigenen Trainingserfolg anzeigen
- *Lehrerin / Lehrer*  
Präzisierung: «Lehrer/-in» meint hier eine Rolle in Bezug auf die Vokabeltrainings-Datenbank, nicht eine Rolle im sozialen System einer Schule. Eine Lehrperson kann in Bezug auf die Datenbank allenfalls sowohl die Lehrer/-innenrolle als auch die Schüler/-innenrolle innehaben. Die Rolle der Lehrerin / des Lehrers hat folgende Rechte:
  - Beliebige Lernpensen über das Web-Interface anzeigen
  - In der Wörterdatenbank suchen (über das Web-Interface)
  - Eigene Lernpensen erfassen, bearbeiten und löschen
  - Vorhandene Vokabeln den eigenen Lernpensen zuteilen und diese Zuteilung entfernen
  - Neue Vokabeln zu den eigenen Lernpensen erfassen, bearbeiten und löschen
  - Einer Lerngruppe Pensens zum Lernen zuteilen
  - *Nicht aber:* Die Lernerfolgsdaten der Schülerinnen und Schüler anzeigen
- *Administratorin / Administrator* mit folgenden Rechten:
  - Alle Rechte der Lehrer/-innenrolle.
  - Lernpensens zu den verwendeten Lehrbüchern erfassen und bearbeiten
  - Die Zuteilung von Vokabeln zu den Lernpensens bearbeiten
  - Beliebige Vokabeln erfassen, bearbeiten und löschen
  - Die Datenbank über ein Administrations-Interface wie PHPMyAdmin pflegen. Theoretisch schliesst das den Zugang zu den Lernerfolgsdaten der Schülerinnen und Schüler ein.

### **2.3.6. Einfache Administration der Wörterdatenbank**

Damit ein Vokabeltrainer nicht nur während des Grundkurses mit dem Lehrbuch, sondern auch danach genutzt wird, muss es auch Personen ohne jede Datenbank-Kenntnis möglich sein, weitere Vokabeln zu importieren und allenfalls vorhandene zu bearbeiten.

### **2.3.7. Plattform-Unabhängigkeit**

Selbstverständlich sollte ein Vokabeltrainer wenn immer möglich unabhängig von der Art des verwendeten Computers und vom Betriebssystem verwendet werden können. Besondere Aufmerksamkeit verdient hier, dass die Software auch auf Kleingeräten wie Smartphones laufen soll.

## 3. Linguistische und technische Grundlagen

### 3.1. «Meinten Sie...» – Wie erkennt ein Computer Tippfehler?

#### 3.1.1. Arten von Tippfehlern

Es ist (wie oben Kapitel 2.3.1 erklärt) je nach Art des gewünschten Vokabeltrainings sinnvoll, wenn die Trainingssoftware Tippfehler von «richtigen, sachlichen» Fehler unterscheiden kann. Man wird also für die Schwere des Fehlers ein Mass finden müssen.

Tippfehler lassen sich in folgende Gruppen einteilen:

1. Buchstabendreher («nciht» statt «nicht»)
2. Fehlende oder überzählige Buchstaben («nich» oder statt «nicht» oder «pfeiffen» statt «pfeifen»)  
Dazu gehören auch fehlende oder überzählige Konsonantenverdoppelungen oder Vokallängungen.
3. Falsche Buchstaben («Satein» statt «Latein»)

#### 3.1.2. Buchstabendreher

Am einfachsten zu handhaben sind die Buchstabendreher: Wörter mit verdrehten Buchstaben bestehen aus denselben Buchstaben wie das jeweils richtige Wort – einfach in falscher Reihenfolge. Um das jeweils richtige Wort zu finden, reicht es, Die Buchstaben der falschen Eingabe in alphabetische Reihenfolge zu bringen und in der Datenbank zu jedem Wort eine «alphabetisierte» Version vorzuhalten. So kann das richtige Wort mit einer einfachen Suche in der Tabellenspalte mit den «alphabetisierten» Strings gefunden werden.

Theoretisch ist es möglich, dass eine falsch gedachte Lösung von der Software als richtige Lösung mit verdrehten Buchstaben interpretiert wird (z.B. «siede» für «diese»), aber die Wahrscheinlichkeit, dass eine Benutzerin genau solche Fehler macht, ist so klein, dass dieser Fall ignoriert werden darf.

Wie die alphabetische Sortierung der Buchstaben eines Wortes technisch umzusetzen ist, wird in Kap. 4.9.1 beschrieben.

### 3.1.3. Fehlende, überzählige oder falsche Buchstaben

Die Erkennung des richtigen Wortes ist weniger einfach, wenn das falsch getippte Wort nicht dieselben Zeichen enthält wie das richtige, sondern weniger, mehr oder andere. In diesem Fall kann nicht einfach zwischen Übereinstimmung und Nicht-Übereinstimmung entschieden werden, sondern es muss festgestellt werden, ob die Eingabe noch *nahe genug* bei der richtigen Lösung ist – eine Art «fuzzy» Prüfung also. Dafür bieten sich zwei Verfahren an: Die phonetische Suche und die Messung des Levenshtein-Abstands.

### 3.1.4. Phonetische Suche mit Soundex-Algorithmus und Kölner Phonetik

Eine Möglichkeit, die noch ohne Abstandsmessung zwischen Strings auskommt, ist die Verwendung eines Algorithmus zur phonetischen Suche. Vereinfacht gesagt, werden Strings so in eine Art Hashcodes umgerechnet, dass Strings, die gleich ausgesprochen werden, denselben Code ergeben. Auf diese Weise können die üblichen Rechtschreibfehler (Siehe Aufzählung oben Punkt 2) neutralisiert werden (vgl. [Mic94, Mic07]). Der bekannteste Algorithmus hierfür ist der *Soundex-Algorithmus*, der in den gängigen Programmiersprachen als Funktion implementiert ist. Soundex eignet sich allerdings vor allem für englische Wörter, für das Deutsche bietet sich stattdessen die *Kölner Phonetik* (siehe [33]) an, für die [Zim08] eine PHP-Implementierung zur Verfügung gestellt hat. Man wird allerdings beachten müssen, dass die Kölner Phonetik gelegentlich für durchaus verschiedene Wörter denselben Code generiert (z.B. «lesen» und «lassen»). Bessere Resultate sind deshalb mit einer Kombination von phonetischer Suche und Abstandsmessung zu erzielen.

### 3.1.5. Levenshtein-Distanz zwischen Strings

Schon 1965 schlug der sowjetische Mathematiker W. I. Levenshtein ein Verfahren vor, wie man den Abstand zwischen zwei Zeichenketten als die «minimale Anzahl von Löschungen, Einfügungen und Ersetzungen, die das Wort X in das Wort Y umwandeln» ([Mic94]) messen kann: Diese *Levenshtein-Distanz* lässt sich messen, indem man in eine Matrix die zwei Zeichenketten und alle notwendigen Änderungsoperationen einträgt. In Tabelle 3.1 wird dies am Beispiel der Strings «Pfahl» und «Pfad» demonstriert.

	ε	P	f	a	h	l
ε	0	1	2	3	4	5
P	1	0	1	2	3	4
f	2	1	0	1	2	3
a	3	2	1	0	1	2
d	4	3	2	1	1	2

Tabelle 3.1.: Matrix zur Messung des Levenshtein-Abstands

Der Abstand zwischen den Strings «Pfahl» und «Pfad» beträgt 2.

Die Berechnung der Levenshtein-Distanz wird in so gut wie allen höheren Programmiersprachen als vor-implementierte Funktion zur Verfügung gestellt. Ihre Komplexität ist, wie aus der Matrix leicht ersichtlich wird,  $\Theta((|String1| + 1) \times (|String2| + 1))$  – kein Problem, solange nicht Tausende solcher Vergleiche gemacht werden müssen.

### 3.1.6. Fazit: Tippfehler-Erkennung: Kombination von mehreren Verfahren

Um Tippfehler zu identifizieren, bietet sich für einen Vokabeltrainer eine Kombination aller beschriebenen Verfahren an:

1. Vergleich der «alphabetisierten» Eingabestring mit den «alphabetisierten» Strings der gespeicherten Lösungen. Gibt es eine Entsprechung, so gilt die Eingabe als richtige Lösung.
2. Prüfung, ob der Wert der Eingabe gemäss Kölner Phonetik dem Wert einer der richtigen Lösungen entspricht. In diesem Fall gilt die Eingabe als richtige Lösung.
3. Messung des minimalen Levenshtein-Abstands der Eingabe zu den gespeicherten Lösungen und allenfalls Prüfung, ob die Datenbank ein Wort mit noch kleinerem Levenshtein-Abstand enthält. In diesem letzten Fall gilt die Lösung als falsch. Die zuletzt genannte Prüfung muss auf wahrscheinliche Fälle eingeschränkt werden, damit das Programm nicht zu langsam reagiert (genauerer dazu: Kapitel 4.9.2).

## 3.2. Synonyme erkennen

### 3.2.1. Entwicklung des Wortschatzes einer natürlichen Sprache

Natürliche Sprachen sind im Vergleich zu Fachsprachen furchtbar ungenau: Erstens kann ein Wort, je nach gegebenem Zusammenhang, sehr verschiedenes bedeuten; zweitens gibt es für ein und dieselbe Vorstellung oft mehrere praktisch gleichbedeutende Wörter. Schliesslich ist drittens die Bedeutung prinzipiell jedes natürlich-sprachlichen Wortes einem permanenten historischen Wandel unterworfen.

Schon Platon [Pla90, Stephanus 430a ff] macht diese Beobachtung: In seinem Dialog «Krátýlos» lässt Sokrates seine Gesprächspartner langsam erkennen, dass die Wörter je nach Ort und Zeit unterschiedliche Bedeutungen haben. Von dieser Tatsache ausgehend und davon, dass es verschiedene «richtige» Sprachen gibt, entwirft Platon ein Theorem, das viel später von [DS67, 76ff] aufgenommen und wirkungsmächtig formuliert wird: Sprachliche «Entitäten» sind *Zeichen*. Sie *verweisen* auf aussersprachliche Gegenstände, wobei die genaue Art dieses Verweisens erst einmal ungeklärt ist. Auf jeden Fall ist das sprachliche Zeichen «etwas im Geist tatsächlich Vorhandenes, das zwei Seiten hat» [DS67, 78 f]: Die (geistige) Vorstellung und das (als akustisches Ereignis produzierte) Lautbild oder, mit von ihm neu eingeführten Fachbegriffen: das Bezeichnete (signifié) und das Bezeichnende (signifiant). Eine der Grundeigenschaften des so definierten Zeichens ist folgende:

Das Band, welches das Bezeichnete mit dem Bezeichnenden<sup>1</sup> verknüpft, ist beliebig; und da wir unter Zeichen das durch die assoziative Verknüpfung eines Bezeichnenden mit einem Bezeichneten erzeugte Ganze verstehen, so können wir dafür auch einfacher sagen: *das sprachliche Zeichen ist beliebig*. [DS67, 79] (Hervorhebung im Original)

In der Regel visualisiert man die Beziehung zwischen Bezeichnendem, Bezeichnetem und aussersprachlichem Gegenstand durch das Semiotische Dreieck (siehe Abb. 3.1).

Sprechen zwei Menschen miteinander, so drückt der Sprecher den gedachten Inhalt seiner Botschaft (signifiés) in artikuliertem Schall (signifiants) aus; der Hörer empfängt den Schall, (re-)konstruiert den gedachten Inhalt (signifiés) und bezieht diesen Inhalt gemäss seinem eigenen Weltwissen auf aussersprachliche Gegenstände. Missverständnisse entstehen oft dadurch, dass Sprecher und Hörer dasselbe signifiant nicht auf dasselbe signifié beziehen. Der Hörer konstruiert in diesem Fall aus den signifiants einen wesentlich anderen Inhalt als den vom Sprecher gemeinten. Gerade derartige Missverständnisse zeigen deutlich, dass die Beziehung zwischen signifiant und signifié grundsätzlich beliebig ist. – Allerdings zeigt das normalerweise vorkommende Gelingen von Kommunikation, dass unter «beliebig» nicht «total willkürlich» zu verstehen ist. Dass (sprachliche) Zeichen beliebig sind, bedeutet nur, dass die Beziehung zwischen Lautgestalt und Vorstellung in keiner Weise natürlich begründet ist, sondern auf der impliziten Konvention einer Sprachgemeinschaft beruht.

---

<sup>1</sup>De Saussure verwendet an dieser Stelle das Wort «Bezeichnung» anstelle von «Bezeichnendes».

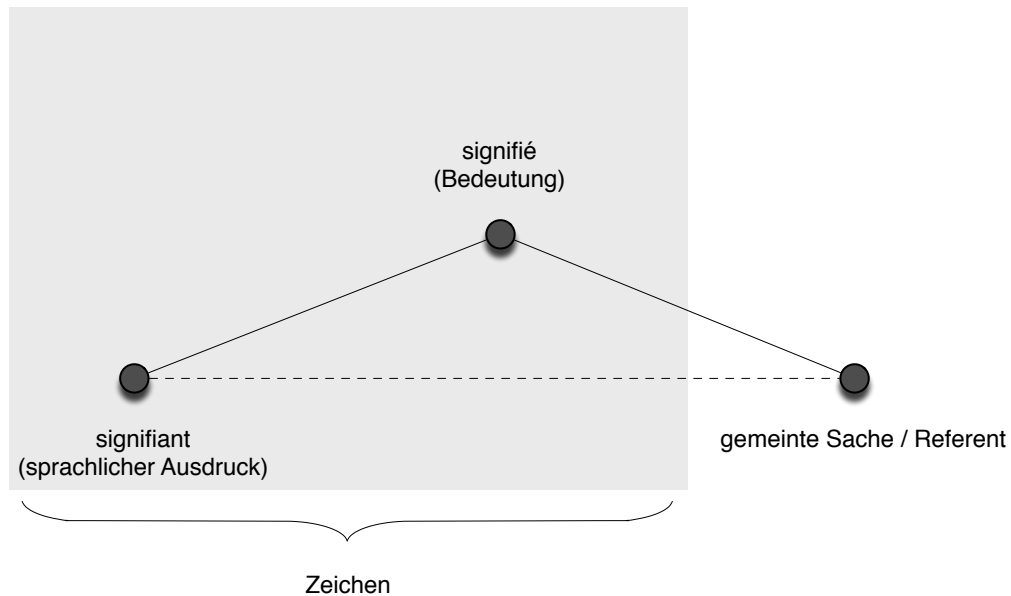


Abbildung 3.1.: Semiotisches Dreieck gemäss [WSU06, 52, fig. 2]

Diese Beliebigkeit natürlich-sprachlicher Zeichen mag für den wissenschaftlichen Diskurs eine Erschwernis sein – für die alltägliche Sprache ist sie die Ursache ihrer ungeheuren Leistungs- und Entwicklungsfähigkeit. Sie eröffnet nämlich die Möglichkeit, für eine neue Vorstellung (signifiant) einen schon vorhandenen sprachlichen Ausdruck (signifié) zu verwenden. Solche Bedeutungsverschiebungen geschehen in der Regel anhand einer «bildlichen» Ähnlichkeit (Metapher) oder anhand einer irgendwie gearteten «inhaltlichen» Beziehung (Metonymie). Zwei Beispiele mögen illustrieren, wie wichtig die Möglichkeit solcher Bedeutungsverschiebungen ist:

- In den 1980er Jahren wurde ein Gerät entwickelt, das es erlaubte, einen Pointer auf einem Computerbildschirm in beliebigen Richtungen zu bewegen: Etwas, was es auf der Welt noch nicht gegeben hatte. In einer Sprache ohne Metaphern hätte man dafür ein «noch nicht besetztes» Wort erfinden müssen. In Realität aber hat man das neue Ding aufgrund einer entfernten Ähnlichkeit «Maus» genannt. Eine schöne Metapher.
- Metonymien erlauben oft sprachliche Kürze: So sagen wir nicht: «Vergiss das Buch, das Andreas Meiers Text über Datenbanken enthält, nicht!», sondern einfach: «Vergiss den Meier nicht!».

Durch die Verwendung von Metaphern und Metonymien kann sich eine natürliche Sprache gelegentlich sehr schnell an neue Gegebenheiten anpassen. Da dies ständig geschieht, ist zumindest potentiell jedes Wort immer *polysem*: Jedes Wort hat ein gewisses Spektrum von Bedeutungen, und je nach historischer Entwicklung können diese sehr weit auseinander liegen.

Ein Beispiel hierfür möge genügen: Das französische Wort «la serviette» bedeutet sowohl «Tuch zum Abwischen des Mundes», als auch «Mappe». Tatsächlich hat sich die Bedeutung «Mappe» über eine Metonymie aus «Tuch» entwickelt. Um das zu verstehen, muss man wissen, dass diese «Mappen» ursprünglich leinene Umschlagtücher waren, in denen sich allerhand transportieren liess. Sie bestanden also aus dem selben Material wie die Mund- und Tischtücher. Die Benennung nach ihrem ursprünglichen Material behielten sie auch dann noch, als sie längst aus Leder und Karton gefertigt wurden (vgl. [Bar96, 115 ff]).

Zusätzlich zu polysemen Wörtern, deren Bedeutungen sich aus einer gemeinsamen Wurzel historisch voneinander entfernt haben, gibt es in jeder Sprache auch Homonymien: Als *Homonymie* wird eine Gleichheit zweier signifiants bezeichnet, die durch die historische Veränderung einer Sprache entstanden ist: Einerseits verändert sich der Lautbestand einer Sprache nach und nach. Ursprünglich verschiedene Wörter können durch die Lautentwicklung über die Jahrhunderte in Klang und Schriftbild zusammenfallen. Andererseits übernimmt jede Sprache immer Wörter aus anderen Sprachen, was ebenfalls zu Homonymien führen kann.

Hier haben sich gleichsam nicht signifiés voneinander weg- sondern signifiants aufeinander zubewegt. Ein Beispiel dafür wären etwa die Substantive «Tau» (Kondenswasser) und «Tau» (starkes Seil), die sich in Aussprache und Schrift durch nichts unterscheiden, aber verschiedenen Ursprungs sind (Details in [Klu02, s.v. Tau]).

Aus einer rein synchronen Perspektive unterscheiden sich Homonymie und Polysemie kaum voneinander. Im Zusammenhang der vorliegenden Arbeit genügt es festzustellen, dass es signifiants – in unserem Fall Zeichenketten – gibt, die auf viele, zum Teil weit auseinander liegende signifiés verweisen.

### 3.2.2. Was ist ein Synonym?

Ein Synonym, so lesen wir z.B. im [DR74, s.v. Synonym], ist ein bedeutungsgleiches oder -ähnliches Wort. Ob zwei Wörter synonym sind, lässt sich auf die Frage reduzieren, ob sie durcheinander austauschbar sind, ohne dass die betreffende Aussage ihren Sinn ändert oder syntaktisch falsch wird (so z.B. [Nab Ja]).

Die oben gemachten Erklärungen über Metapher und Metonymie zeigen aber, dass diese Bedeutungs-Ähnlichkeit nicht unabhängig von einem *Kontext* gegeben ist:

Ausser in Wörterbüchern und Vokabeltrainern steht ein Wort nie allein da, sondern es ist immer Teil eines *Textes*. Ein «Text» im hier verwendeten Sinne braucht nicht schriftlich fixiert zu sein, auch ein Gespräch oder eine Folge mehrerer Gespräche ist ein Text. Dieser weiter gefasste Begriff von «Text» ist schon im Wort «Text» selbst angelegt: Das lateinische *textus* heisst ursprünglich «Gewebe». Ein Text ist ein Gewebe sprachlicher Äusserungen und durch sie evozierter Bedeutungen.

Der Text, innerhalb dessen ein Wort steht oder geäussert wird, bildet seinen unmittelbaren Kontext. Zum Kontext einer sprachlichen Äusserung gehört jedoch nicht nur der eigentliche «umgebende Text»: Der Kontext einer Äusserung hat immer auch eine *pragmatische*

Dimension: Die Person, die sie äussert, die Rede- oder Schreib-Absicht, in der sie geäussert wird, die Zeit, der Ort und die Umstände, unter denen das geschieht. Erst in diesem Kontext lässt sich die genaue Bedeutung eines Wortes bestimmen:

So ist z.B. «Blatt», wenn von journalistischen Medien die Rede ist, synonym zu «Zeitung», in einer Messerhandlung synonym zu «Klinge» und in der Graphentheorie zu «kinderloser Knoten einer Baumstruktur» (man beachte im letzten Beispiel den Metaphernreichtum der informatischen Fachsprache!). Die Bedeutung von «Blatt» ist also je nach Ort und Zusammenhang verschieden. Je nach Zeit verschieden ist z.B. die Bedeutung des Wortes «geil»: Vom Mittelhochdeutschen bis ins 16. Jh. ist es synonym zu «fröhlich» und «kräftig», vom 17. bis ins 20. Jh. bedeutet es vorwiegend «lüstern» ([Gri04, s.v. geil]). Seit Ende des 20. Jh. wird es synonym zu «grossartig» verwendet und nicht mehr auf das Subjekt, sondern auf das Objekt eines Gefühls bezogen.

Wortbedeutungen variieren aber nicht nur nach Zeit, Ort, Redeabsicht, sprechender Person etc., sondern können sogar innerhalb eines Textkorpus variieren: Jeder Text, ob geschrieben oder gesprochen erzeugt «lokale» Synonymien und Antonymien, besonders intensiv und planvoll geschieht dies in kunstvoll komponierten Texten (Vgl. z.B. [Lot72, 125 ff]).

Solange man mit Alltagstexten in einer einzigen Sprache zu tun hat, entstehen dadurch gelegentlich Missverständnisse, aber das Verstehen scheitert nicht vollständig. Dies kann aber beim Lernen von Fremdsprachen, zumal beim Lesen auch historisch sehr weit entfernter Texte, durchaus geschehen. So entsteht z.B. ein gravierendes Missverständnis, wenn jemand das lateinische «ire» (gehen) als «funktionieren» auffasst, obwohl «gehen» und «funktionieren» im Deutschen durchaus synonym sein können. Die Bedeutungsspektren des lateinischen und des deutschen Wortes überschneiden sich nur teilweise.

### **3.2.3. Fazit: Schwierigkeiten im maschinellen Umgang mit Synonymen**

Aus den oben gemachten Ausführungen über Bedeutungswandel, über Synonymie, Polysemie und Homonymie wird deutlich, welche Schwierigkeiten sich der maschinellen Erkennung von Synonymen entgegen stellen:

- Wenn in einer Synonyme-Datenbank nur nach einem einzelnen Wort ohne jeden Zusammenhang gesucht werden kann, dann führen Polysemien und Homonymien mit grosser Wahrscheinlichkeit dazu, dass zu jeder Bedeutung eines Wortes (zu jedem signifié eines signifiant) eine Gruppe von Synonymen gefunden wird. Dass dies tatsächlich so ist, lässt sich leicht experimentell zeigen, indem man z.B. in <http://www.openthesaurus.de> nach dem Wort «Tau» sucht. Dies ist im Zusammenhang eines Vokabeltrainers deswegen ein Problem, weil nicht algorithmisch entschieden werden kann, welche dieser Synonymgruppen zum Bedeutungsspektrum einer bestimmten Vokabel gehören. In Abb. 3.2 wird dies deutlich: Das deutsche «Hafen» gehört zu drei Synonymgruppen, aber nur die Gruppe mit dem Bedeutungszentrum «Landeort für Schiffe» kann als Bedeutung von «portus» akzeptiert werden.

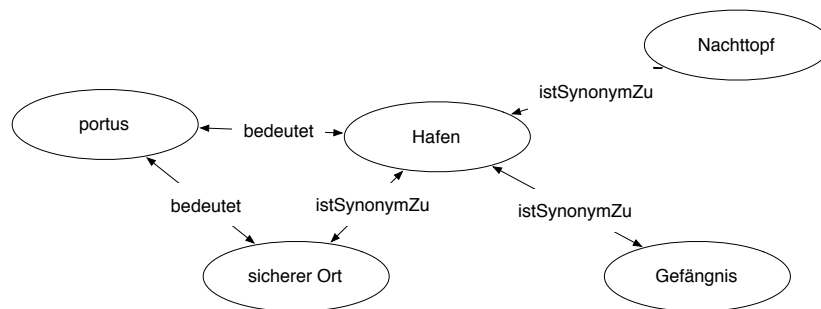


Abbildung 3.2.: Bedeutungsspektren von lat. «portus» und dt. «Hafen»

- Auch abgesehen von Polysemie und Homonymie ist die Bedeutung jedes Wortes und damit seine Zugehörigkeit zu Synonymgruppen kontext-abhängig.
- Jedes Wort hat eine «Bedeutungs-Wolke», und die Bedeutungswolken von Wörtern verschiedener Sprachen sind oft nicht deckungsgleich.
- Wortbedeutungen und damit Synonymien sind ständigem, manchmal raschem Wandel unterworfen.

### 3.2.4. Anforderungen an eine Datenquelle zum Suchen von Synonymen

Damit eine Datenquelle zur Prüfung von Synonymien durch einen Vokabeltrainer geeignet ist, sollte sie folgende Anforderungen erfüllen:

- *Hohe Aktualität*  
Die Synonymgruppen müssen dem jeweils aktuellen Sprachgebrauch entsprechen.
- *Einbezug von Kontexten in die Einteilung von Synonymgruppen*  
Idealerweise werden für jede Synonymgruppe kurze Kontexte mitgegeben, die in irgendeiner Weise mit einem Kontext der gesuchten Vokabel verglichen werden können.
- *Funktional korrekte Synonymgruppen*  
Dass ein Verb nicht Synonym eines Substantivs sein kann, leuchtet sofort ein. Dass hingegen eine Präposition wie «nach» nicht Synonym eines Adverbs wie «nachher» sein kann, scheint weniger klar zu sein – jedenfalls finden sich etliche Web-Lexika, die diese beiden Wörter als Synonyme aufführen. Das mindert ihren Wert als Datenquelle für einen Vokabeltrainer:

Gerade bei «Partikeln» ist die genaue Kenntnis der Wortart nämlich entscheidend für das Textverständnis: «Partikeln» organisieren gleichsam einen Text, sie markieren dessen Strukturelemente. Wer eine Nebensatz-einleitende Konjunktion als Adverb

missversteht, kann die Hauptstruktur eines Satzes nicht erkennen. Tatsächlich sind Wortart-Verwechslungen bei Partikeln beim Übersetzen aus einer (alten) Fremdsprache ins Deutsche eine der häufigen Quellen gravierender Missverständnisse.

Vor allem die Anforderung hoher Aktualität ist mit einer lokalen Datenbank, die nur in relativ langen Zeitabständen und nur von Hand aktualisiert wird, schwer zu erfüllen. Hier scheint die Nutzung von Daten, die über das Web verfügbar sind, attraktiv.

Für den automatischen Datenaustausch zwischen Programmen über das Internet sind in den letzten Jahren unter der Bezeichnung «Web Services» etliche zum Teil recht komplexe Standards entstanden, die für den Betrieb umfangreicher verteilter Systeme gedacht sind. Für die Suche nach Synonymen innerhalb eines Vokabeltrainers genügt jedoch ein einfacher Service nach dem REST-«Standard», den [Fie00] in seiner Dissertation formuliert hat. «Standard» steht hier deswegen in Anführungszeichen, weil REST vom W3C gerade nicht als Standard übernommen worden ist, sich andererseits aber wegen seiner Einfachheit als Quasi-Standard etabliert hat. Die Idee von REST ist, dass über einen URI sogenannte *temporale Ressourcen* in einem Format angeboten werden, in dem sie vom Client weiter verarbeitet werden können, ohne dass die aufseiten des Servers gespeicherten Daten vollständig offen gelegt werden. «Temporal» sind die Ressourcen deswegen, weil sie nicht nur dauernd gespeicherte Dateien sein können, sondern auch Ergebnisse von Datenbankabfragen, die aber dennoch über eine URI eindeutig identifiziert sind. Selbstverständlich kann eine temporale Ressource auch leer sein, wenn z.B. eine Datenbank auf die betreffende Abfrage eine leere Menge zurückgibt.

Für textförmige Daten wird Austauschformat in der Regel XML oder JSON verwendet. Die Schnittstelle zu einem REST-Webservice ist so einfach wie diejenige zu irgendeiner Ressource im Web: Ein simpler HTTP-Request.

Für die Verwendung als Synonymquelle für einen Vokabeltrainer müsste ein REST-Webservice folgende Anforderungen erfüllen:

- *Gute Strukturierung:*  
Der Webservice sollte die Daten in einer stabilen, konsistenten Struktur halten und in einem gut zugänglichen Format ausgeben. Praktisch sind gut strukturierte XML- oder JSON-Ausgaben.
- *Wenig Datenverkehr:*  
Es sollte möglich sein, gezielt auf die gewünschten Eigenschaften eines Wortes oder einer Wortkategorie zuzugreifen. Ein Webservice ist wenig geeignet, wenn dafür erst einmal ganze Lexikon-Artikel heruntergeladen und in einem Parsing nach Schlüsselwörtern durchsucht werden müssen.

Momentan stellen vor allem die folgenden Websites Daten zur Synonymie deutscher Wörter zur Verfügung:

1. Wikipedia (<http://www.mediawiki.org/wiki/API>)

2. Wiktionary (<http://www.mediawiki.org/wiki/API>)
3. Open Thesaurus (<http://www.openthesaurus.de/about/api>)

### 3.2.5. Wikipedia

Die Möglichkeit, für die Suche nach Synonymen oder den Aufbau von Ontologien auf die Wikipedia zurückzugreifen, wurde schon mehrfach untersucht. [GHP10, 71] listen die wichtigsten Arbeiten auf. Im Zusammenhang der vorliegenden Arbeit ist besonders interessant, dass die Wikipedia die gesuchte Flexibilität aufweist: Sie wächst nicht nur schnell, sondern Veränderungen in den Wortbedeutungen werden zumindest potentiell nachgeführt. Dazu sind die Einträge sehr gut erschlossen und meist über Kategorisierungen miteinander verbunden. Die Untersuchung, ob zwei gegebene Wörter synonym sind, könnte also erfolgen, indem die Kategorien des einen Wortes abgefragt werden und dann für jede gefundene Kategorie geprüft wird, ob ihr das zweite Wort zugeordnet ist.

Dazu sind die Bestände über die Wikimedia-API relativ gut zugänglich: Die Kategorien, zu denen ein Wort gehört, können leicht abgefragt werden, ebenso die Wörter, die zu einer Kategorie gehören.[Med J]

Trotzdem ist die Wikipedia zur Suche von Synonymen für einen Vokabeltrainer nur bedingt nutzbar. Sie enthält nämlich, wie jede Enzyklopädie, vorwiegend Substantive. Verben fehlen oft, Adverbien, Präpositionen und Konjunktionen fehlen praktisch völlig.

### 3.2.6. Wiktionary

Im Unterschied zur Wikipedia ist Wiktionary nicht eine Enzyklopädie, sondern ein Wörter«buch». Es enthält deshalb auch die Wortarten, die in der Wikipedia fehlen. Es hat andererseits den Nachteil, dass die Wörter nicht in semantische, sondern in formale Kategorien eingeteilt sind, und dies nicht immer zuverlässig: Zum Beispiel gibt eine Suche nach den Kategorien von «nach» Einträge wie «Präposition (Deutsch)» und leider sogar das falsche «Adjektiv (Deutsch)» zurück.

Um im Wiktionary nach Synonymen zu suchen, muss jeweils der ganze Text eines Wörterbuchartikels abgerufen und nach den Schlüsselwörtern `{{Synonyme}}`<sup>2</sup>, `{{Sinnverwandte Wörter}}`<sup>3</sup> und `{{Bedeutungen}}` durchsucht werden. Dies erzeugt unnötig viel Datenverkehr und ist aufwändig. Vor allem aber scheint die Struktur der Artikel nicht ganz konsistent zu sein, und es ist denkbar, dass in einigen Artikeln die Synonyme auch unter `{{bedeutungsähnliche Wörter}}` oder ähnlich aufgeführt sind. man kann also nicht darauf vertrauen, dass die vorhandene Information durch eine Suche nach den genannten Schlüsselwörtern wirklich zutage gefördert wird.

---

<sup>2</sup>Zum Beispiel: <http://de.wiktionary.org/wiki/schicken>

<sup>3</sup>Zum Beispiel: <http://de.wiktionary.org/wiki/Graph>

Dazu kommt, dass im Wiktionary die Synonymgruppen von «Partikeln» ohne genaue Beachtung der Wortart-Grenzen zusammengestellt sind. So werden z.B. Adverbien als Synonyme von Präpositionen genannt.

### 3.2.7. Open Thesaurus

OpenThesaurus.de, ebenfalls ein Wörter-«buch» bietet einen REST-Webservice an, der die Synonymgruppen, zu denen ein Wort gehört, im Format XML zurückgibt [Nab Jb]. Die Prüfung, ob zwei Wörter synonym sind, kann deshalb recht einfach unter Verwendung der DOM-Funktionalität der jeweiligen Programmiersprache geschehen.

Leider nennt aber auch der OpenThesaurus bei «Partikeln» Wörter mit verschiedenen syntaktischen Funktionen als Synonyme. So fanden sich bis zum 11.11.10 z.B. unter den Synonymen der Präposition «nach» auch die *Adverbien* «dahinter», «hinten» und die *Konjunktion* «nachdem», ohne dass die Wortart dieser Wörter irgendwie gekennzeichnet gewesen wäre.

Dieser Umstand macht den OpenThesaurus zu einer wenig zuverlässigen Synonymquelle für einen Vokabeltrainer. Allerdings lassen sich die falsch zusammengestellten Synonymgruppen mühelos bearbeiten – OpenThesaurus ist wirklich «open». Wird der OpenThesaurus als Synonymquelle verwendet, dann muss deshalb (zumindest bei «Partikeln») ein Synonym, bevor es von der Software als richtige Lösung akzeptiert wird, einer qualifizierten Person zur Prüfung vorgelegt werden. Dieser Person steht es frei, die entsprechende Synonymgruppe zu korrigieren.

### 3.2.8. Fazit: Synonymsuche mit OpenThesaurus

Trotz der genannten Bedenken wird in dieser Arbeit der OpenThesaurus als Synonymquelle verwendet. Die PHP-Klasse, die die Suche nach Synonymen übernimmt, wird aber so programmiert, dass sich die Suche mit wenig Aufwand auf andere Webservices ausdehnen oder übertragen lässt. Denkbar wäre etwa, für die Suche nach Synonymen von Substantiven den in 3.2.5 angetönten Weg über die Kategorien der Wikipedia zu wählen.

### 3.2.9. Problem: Der Kontext

Wie schon in den Kapiteln 3.2.1 und 3.2.4 dargelegt, sind zwei Wörter kaum je schlechthin synonym, sondern in der Regel *mit Bezug auf einen bestimmten Kontext*. Worum es in einem Text, der ein bestimmtes Wort enthält, geht, lässt sich mittlerweile maschinell ermitteln, wie [GHP10] gezeigt haben. In der Datenbank eines Vokabeltrainers – jedenfalls des vorliegenden – sind aber keine Texte, sondern nur einzelne Wörter gespeichert. Um beurteilen zu können, ob zwei deutsche Wörter in Beziehung auf eine Vokabel tatsächlich synonym sind, müsste da, wo die Beziehung zwischen einer Vokabel und einem deutschen

Wort gespeichert ist, jeweils noch ein kurzer Text gespeichert werden, aus dem das Thema des betreffenden Kontextes extrahiert werden könnte.

Überhaupt stösst man hier prinzipiell an die Grenze dessen, was ein Vokabeltrainer leisten kann: Letztlich generiert jeder Text in sich ein Geflecht *lokaler* Synonyme und Antonyme, und ob in einer Übersetzung des Textes ein bestimmtes deutsches Wort richtig, schlecht oder gar falsch gewählt ist, hängt von diesem Geflecht ab. Die Wortbedeutungen, deren Kenntnis man mit einem Vokabeltrainer üben kann, sind immer nur Anhaltspunkte zu einer vertieften inhaltlichen Auseinandersetzung mit einem Text.

# 4. Implementierung eines Prototypen

## 4.1. Die Trainingsmodi

Der vorliegende Prototyp erlaubt nur das Trainieren deutscher Bedeutungen zu fremdsprachlichen Vokabeln. Eine Benutzerin kann am Anfang einer Trainingssequenz auswählen, was und wie sie trainieren will:

- Es stehen drei Varianten für den Inhalt des Trainings zur Verfügung:
  - Das Lernpensum memorieren und trainieren, das der betreffenden Lerngruppe zuletzt zum Lernen zugeteilt worden ist
  - Ein frei ausgewähltes Lernpensum memorieren und trainieren (noch nicht implementiert)
  - Diejenigen Vokabeln trainieren, deren Repetition gemäss «Lernmanagement» (siehe Kapitel 2.1) fällig ist
- Für den Trainingsmodus stehen zwei Varianten zur Verfügung:
  - Training mit strikter Prüfung:  
Die eingegebenen deutschen Bedeutungen dürfen keine Tippfehler enthalten. Synonyme der gespeicherten Bedeutungen werden nicht akzeptiert. (noch nicht implementiert)
  - fehler-tolerantes Training:  
Jede der eingegebenen deutschen Bedeutungen wird zuerst mit den gespeicherten Bedeutungen verglichen. Ergibt sich bei einer Eingabe keine Übereinstimmung, werden folgende Prüfungen gemacht:
    1. Prüfung auf Buchstabendreher
    2. Prüfung auf Übereinstimmung mit einer gespeicherten Bedeutung gemäss Kölner Phonetik
    3. Prüfung, ob die eingegebene Bedeutung synonym zu einer gespeicherten Bedeutung ist
    4. Messung des Levenshtein-Abstandes zu den gespeicherten Bedeutungen. Prüfung, ob ein anderes, falsches Wort der Eingabe näher ist als eine der gespeicherten Bedeutungen.

- Für das Training von Verben kann der Benutzer zudem angeben, welche Stammformen<sup>1</sup> er sich abfragen lassen will. Diese werden dann wie zusätzliche Vokabeln abgefragt.

## 4.2. Grundsätzliche Design-Entscheide

In Kapitel 2.3 wurden Anforderungen an einen neuen Vokabeltrainer formuliert. Aus diesen Anforderungen ergibt sich, dass es für einen Prototypen am günstigsten ist, wenn er als Web-Anwendung realisiert wird: Trotz der vielen Unterschiede zwischen den Webbrowsern wird er auf diese Weise plattform-unabhängig realisiert werden können.

### 4.2.1. Gewählte Programmiersprache

Der Prototyp wurde deshalb in PHP implementiert, wobei alle «Entitäten» wie Vokabel, Benutzer, deutsche Bedeutung etc., aber auch alle «Engines» wie z.B. der Synonym-Checker, die Trainingsverwaltung oder der Lösungsprüfer als PHP-Klassen ausgestaltet wurden. Das Ziel war, die Schnittstellen der Klassen so zu gestalten, dass die Software entwicklungs-fähig bleibt und dass die Scripts, die die Webseiten erzeugen, ohne allzu viel Programmcode auskommen, damit auch ihr Quellcode nachvollziehbar bleibt.

PHP ist in den letzten Jahren stark weiterentwickelt worden und bietet jetzt ähnlich viele Möglichkeiten wie irgendeine andere objektorientierte Programmiersprache. Trotzdem ist es nach wie vor eine interpretierte Programmiersprache – deshalb spürbar weniger schnell als C++ und ähnliche. Vor allem aber laufen PHP-Programme auf einem Webserver ab.

Dies zweite hat zur Folge, dass PHP-Programme nahezu zustandslos sind. Objekte «leben» nur so lange, wie das Programm läuft, und dieses wird zwingend beendet, bevor der Server die Antwort auf einen eingegangenen HTTP-Request sendet. Das ist der grösste Nachteil des Designs als Web-Anwendung, denn es macht nötig, dass man die Objekte, die länger als für einen Seitenaufruf gebraucht werden, gleichsam gegen die Intention von PHP im Session-Array speichert. Dafür müssen sie am Ende jedes Programm-durchlaufs serialisiert und am Anfang jedes neuen Durchlaufs de-serialisiert werden, ein Aufwand, der entfiele, wenn das Programm im Speicher eines Clients bliebe.

Als Weiterentwicklung wäre eine Realisierung denkbar, bei der ein Teil der Funktionalität auf den Client verlagert wird, sodass dieser die Objekte, die länger gebraucht werden, speichern kann. Dies wäre dann bereits ein Schritt zur Entwicklung eines Stand-Alone-Programms (Siehe Kapitel 5).

---

<sup>1</sup>Stammformen werden diejenigen Konjugationsformen eines Verbs genannt, aus denen der Stamm für die Bildung weiterer Formen deutlich wird. Im Deutschen sind das in der Regel die 2. Pers. Singular des Präsens, das Präteritum und das Partizip II (z.B. gehst – ging – gegangen), im Lateinischen das Präsens, das Perfekt und das vorzeitig-passive Partizip (z.B. faciō, fēcī, factum).

## 4.2.2. Data Base Management System

Die Datenbank, die der Prototyp nutzt, ist eine MySQL-Datenbank, die auf jedem Webserver eingerichtet werden kann.

## 4.2.3. Grafik

Das grafische Design schliesslich ist das eines technischen Prototypen: Es zeigt die Funktionalität, ist aber sehr schlicht und müsste im Hinblick auf Usability noch weiterentwickelt werden. Damit dies ohne Eingriff in die Scripts möglich ist, wurde darauf geachtet, dass alle wichtigen DOM-Elemente der Webseiten benannt oder Klassen zugeteilt sind, und dass sämtliche Formatierungsangaben in zwei zentralen Stylesheets stehen: einem für normal-grosse Bildschirme, einem für Kleingeräte.

## 4.3. Überblick über die Komponenten des Prototypen

Vorbemerkung: Die Dokumentation sämtlicher PHP-Klassen ist als PHPdoc im Stil von Javadoc auf der beiliegenden CD. Sie kann auch eingesehen werden unter <http://vocabulary.trainerdoc>.

Der hier entwickelte Vokabeltrainer besteht aus folgenden Komponenten:

- MySQL-Datenbank:  
Haupt-Entitäten sind die Vokabel, die deutsche Bedeutung, das Lernpensum einerseits und der Benutzer und Lerngruppe andererseits. Vgl. Kapitel 4.5.
- PHP-Klassen für «Entitäten» und Funktionalität  
Auch hier sind Vokabeln, deutsche Bedeutungen, Lernpensum einerseits und Benutzer/-innen und Lerngruppen andererseits modelliert. Dazu werden «Engines» bereitgestellt, die bestimmte Aufgaben übernehmen:
  - Die Klasse Pruefer, die die Richtigkeit eingegebener deutscher Bedeutungen prüft
  - Die Klasse SynonymChecker, die über Webservices (einstweilen [openThesaurus.de](http://openThesaurus.de)) nach Synonymen zu einem eingegebenen Wort sucht
  - Die Klasse Trainingssteuerung, die für eine Benutzerin die zur Repetition anstehenden Wörter ermittelt und sie ihr zum Memorieren und zum Repetieren vorlegt
  - Die Klasse PostDataChecker, die über Web-Formulare eingegangene Daten auf syntaktische Korrektheit überprüft und schädliche Zeichen herausfiltert
  - Weitere Hilfsklassen mit Methoden zur Verwaltung von Benutzern und Wörtern.
- Webseiten um ...

- auf verschiedenen Wegen nach Vokabeln zu suchen, sie anzuzeigen und zu exportieren
  - Vokabeln zu trainieren
  - Vokabeln zu verwalten und zu bearbeiten
  - sich als Benutzer an- und abzumelden und die Sprache auszuwählen, mit der man arbeiten will
- Einige weitere Webseiten, die Informationen über das Programm und Anleitungen enthalten

## **4.4. Was schon vor Beginn dieser Arbeit vorhanden war**

### **4.4.1. Datenbank**

Im Laufe etlicher Jahre hatte ich begleitend zum Unterricht in Griechisch und Latein eine Vokabel-Datenbank aufgebaut. Dies geschah zuerst mit FileMaker Pro, später dann, da sich in FileMaker Pro die n:m-Beziehungen damals nicht sauber modellieren liessen, mit MySQL.

Auch in der MySQL-Version war die Datenbank noch nicht in 3. Normalform, so wurden etwa die Fremdwörter, die von einer Vokabel abgeleitet sind, in einer Spalte der Vokabel-Tabelle gespeichert. Dazu enthielt die Tabelle 'deutsch' nur diejenigen deutschen Wörter, die einmal als Bedeutungen griechischer oder lateinischer Vokabeln erfasst worden waren.

### **4.4.2. PHP-Programme**

Mit der Umstellung von FileMaker Pro auf MySQL musste das Benutzer-Interface für die Datenbank in PHP programmiert werden. Diese Programmierung wurde von Anfang an in einem objektorientierten Design entworfen.

Die PHP-Klassen und Interface-Seiten waren in jener Version gänzlich auf die Verwaltung und Anzeige der Vokabel-Pensen ausgerichtet. Es gab noch keine Ansätze zu Abfrage- oder Memorierfunktionen. Dazu war für den damaligen Entwurf der Klassen und Interface-Seiten den Fragen der Sicherheit so gut wie keine Beachtung geschenkt worden: Die Benutzereingaben wurden z.B. nicht auf schädliche Zeichenfolgen getestet. Die Verbindung zur Datenbank wurde mit den alten PHP-MySQL-Funktionen hergestellt, die ebenfalls keine Sicherheit gegen SQL- oder code injection boten.

Mit der Aufnahme ins Softwareprojekt im Rahmen dieser Arbeit wurden die vorhandenen PHP-Klassen und Interface-Seiten so angepasst, dass einerseits Datenbankverbindungen nur noch mittels des Msqyli-Objekts von PHP 5 hergestellt werden, und dass andererseits sämtliche Benutzereingaben vor der weiteren Verarbeitung auf schädliche Zeichenfolgen hin untersucht werden. Dazu wurde das objektorientierte Design im Hinblick auf Konsistenz, Effizienz und Skalierbarkeit grundsätzlich überarbeitet. Von Grund auf neu entworfen und pro-

grammiert wurde die gesamte Funktionalität zum Prüfen eingegebener Bedeutungen, zum Trainieren von Vokabeln und zum Verwalten von Benutzern. Schliesslich wurden sämtliche PHP-Klassen mit inline-Kommentaren sauber und möglichst verständlich dokumentiert.

## 4.5. Der Aufbau der Datenbank

Die Datenbank besteht aus zwei Konglomeraten von Tabellen:

- Die Tabelle `vokabel` und die auf sie bezogenen Tabellen mit Daten zu den Vokabeln im engeren Sinne
- Die Tabellen, in denen die Benutzerdaten und die Lernkontrolle gespeichert sind

Dazu enthält sie eine in Anlehnung an die RDF-Syntax (vgl. [HKRY08, 35 ff]) aufgebaute Tabelle, die sämtliche Beziehungen zwischen deutschen Wörtern (Synonymie, Antonymie, Über- und Unterordnung) aufnehmen kann.

### 4.5.1. Tabellen mit Vokabel-Daten

Die Tabelle `vokabel` und die Tabellen für jede Sprache (einstweilen `griechisch` und `lateinisch`) bilden den Kern der Datenbank: `vokabel` ist den Sprachen-Tabellen übergeordnet, die Beziehung ist die einer vollständig-disjunkten Generalisierung. Zur Tabelle `vokabel` stehen weitere Tabellen in n:m-Beziehungen, die zusätzliche Informationen wie deutsche Bedeutungen, Wortstämme, Zugehörigkeit zu Lernpensen etc. enthalten. Der genaue Aufbau der Datenbank wird aus Abb. 4.1 deutlich.

### 4.5.2. Modellierung von Beziehungen zwischen Wörtern

Wie in Kapitel 3.2.4 auf Seite 18 festgehalten, ist es sinnvoll, Synonyme anhand der häufig aktualisierten Information von Webservices zu suchen. Allerdings ist noch unklar, wie maschinell geprüft werden kann, ob eine so gefundene Synonymie in Bezug auf die betreffende Vokabel tatsächlich gültig ist. Man wird also diese Synonymien als provisorisch ansehen und zumindest vorderhand durch einen Menschen prüfen lassen müssen.

Deshalb ist es ebenso sinnvoll, dass Synonymie-Relationen, wenn sie einmal geprüft sind, zumindest über eine gewisse Zeit in der Datenbank festgehalten werden. Das eröffnet zudem die Möglichkeit, die in dieser Arbeit entwickelte Web-Anwendung zu einer Offline-Anwendung für mobile Geräte weiterzuentwickeln, der auch ohne Internet-Verbindung möglichst viel von ihrer Funktionalität erhalten bleiben soll.

Im Rahmen eines klassischen relationalen Datenbankdesigns läge es nahe, die Synonymie-Beziehung als eine weitere Beziehungstabelle zu modellieren, deren Spalten jeweils drei Fremdschlüssel enthalten (siehe Tabelle 4.1)

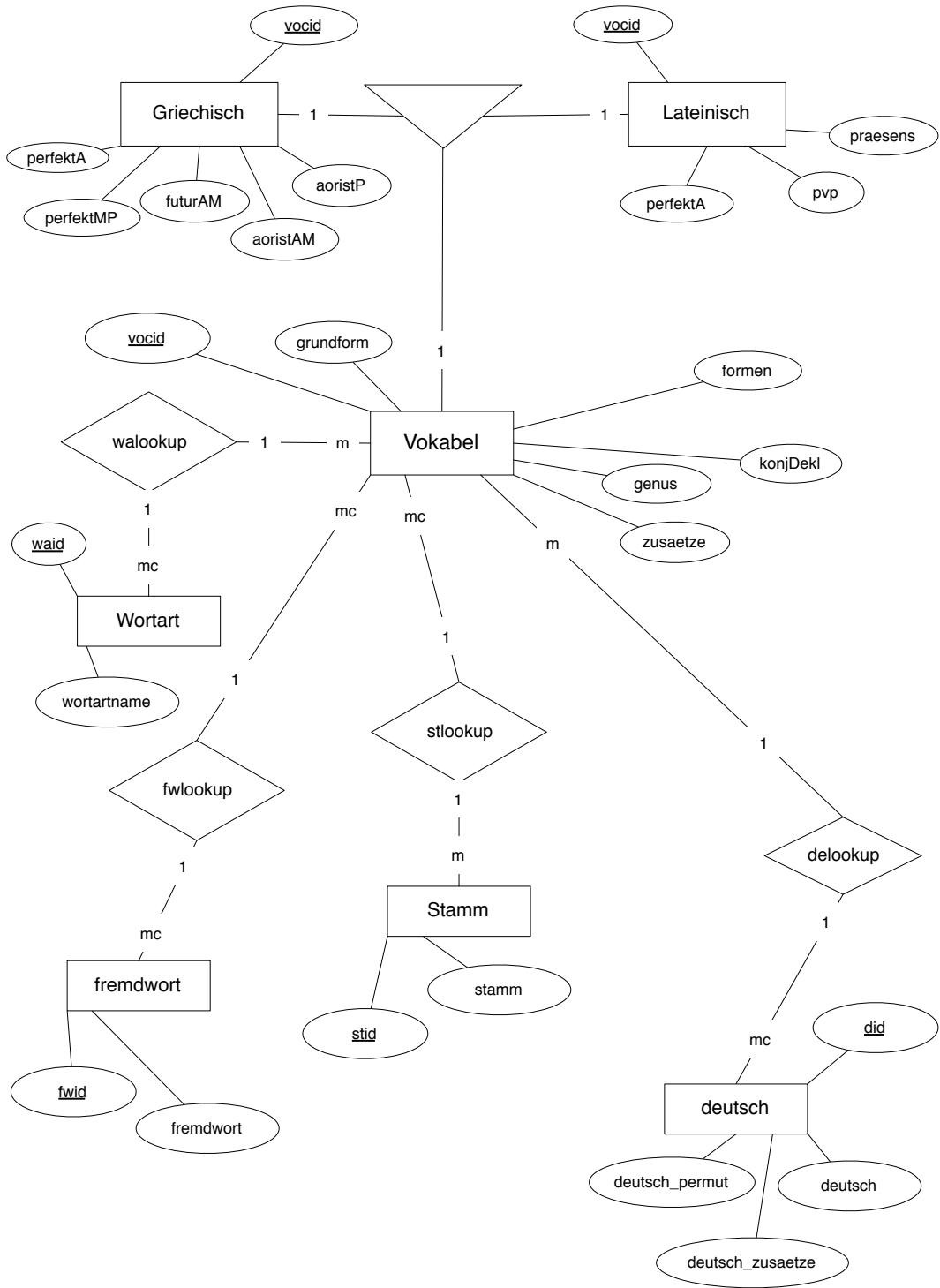


Abbildung 4.1.: ERM Vokabeldatenbank Teil 1: Vokabel, Angaben, Bedeutungen

ist_synonym_in_bezug_auf_vokabel		
deutschID_FK	deutschIDIstSynonym_FK	VocabelID_FK

Tabelle 4.1.: Beziehungstabelle für Synonymien

Diese klassische Beziehungsmodellierung hat aber einen gravierenden Nachteil: Die Art der Beziehung zwischen zwei Wörtern ist bereits im Tabellendesign festgelegt. Für Beziehungen einer anderen Art – etwa Antonymien oder Über- und Unterordnungen – müsste je eine neue Tabelle erstellt werden. Jede Erweiterung des Funktionsumfangs hätte damit einen Eingriff ins Datenbankschema zur Folge. Die Art der Beziehung zwischen zwei Wörtern ist nämlich nicht Teil der Daten, sondern der Metadaten (in diesem Fall des Tabellentitels!) Hier verspricht der Ansatz «Metadata is data» ([SET09, 16]) mehr Flexibilität:

In Beziehungs-Modellen, wie sie im Zusammenhang mit dem Semantic Web entwickelt worden sind, wird die Art der Beziehung zwischen zwei Entitäten selbst zu einem Teil der Daten. Genauer: Es wird nicht mehr mit *Entitäten* gearbeitet, sondern mit Tripeln von jeweils einem Subjekt, einem Prädikat und einem Objekt, z.B:

«Tür» → «istSynonymZu» → «Tor».

Aus mehreren solchen Tripeln lassen sich Beziehungsgraphen aufbauen, wie sie für das Semantic Web gebraucht werden (siehe [HKRY08, 35 ff]). Ein Beispiel davon zeigt Abb. 4.2 .

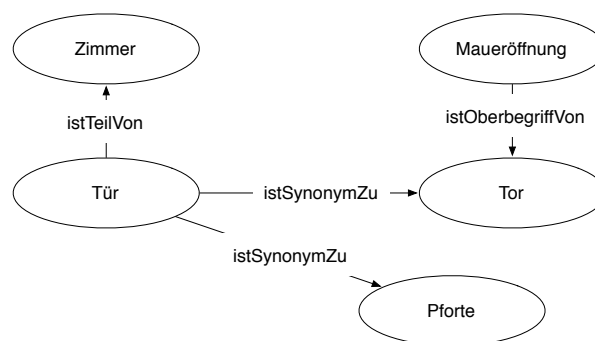


Abbildung 4.2.: Beispiel eines Graphen mit zwei Beziehungstypen

Obwohl diese Art von Beziehungsmodellierung normalerweise in Non-SQL-Datenbanken angewendet wird, lässt sie sich durchaus auch in einer relationalen Datenbank verwenden: Für sämtliche möglichen Beziehungen zwischen Wörtern reicht dann eine dreispaltige Tabelle aus. Tabelle 4.2 zeigt das:

Beziehung		
subjekt	praedikat	objekt
Tür	istSynonymZu	Tor

Tabelle 4.2.: Tabelle Beziehung mit zweiwertigen Beziehungen

Allerdings enthält ein derartiges Tripel noch nicht die gesamte benötigte Information: Eine Beziehung soll ja zwischen drei verschiedenen Entitäten hergestellt werden. Hier zeigt sich der Preis für die grössere Flexibilität: Die Modellierung mehrgliedriger Beziehungen ist aufwändiger, denn das Prädikat «istSynonymZu» selbst muss jetzt zu einer fremdsprachlichen Vokabel in Beziehung gesetzt werden. Das Prädikat der ersten Aussage wird damit zu einem Subjekt oder Objekt gemacht (reifiziert, [HKRY08, 79 ff]). Die gesamte Beziehung im obigen Beispiel muss dann wie in Tabelle 4.3 modelliert werden:

Beziehung		
subjekt	praedikat	objekt
Tür	istSynonymZu	Tor
Synonymie1	hatSubjekt	Tür
Synonymie1	hatObjekt	Tor
porta	akzeptiert	Synonymie1

Tabelle 4.3.: Tabelle Beziehung mit mehrwertigen Beziehungen

Im Zusammenhang mit dem Semantic Web sind in den letzten Jahren Programmbibliotheken entwickelt worden, mit denen sich die Information zu einem gesuchten Graphen aus einer derartigen Tabelle extrahieren lassen ([SET09, 183 ff], [31]). In der Umgebung des hier entwickelten Programms lässt sich die gewünschte Beziehung aber auch mit einem SQL-Statement finden. Die Codesequenz 4.1 zeigt, wie sich alle Wörter finden lassen, die in Bezug auf die Vokabel «porta» synonym zur Bedeutung «Tor» sind.

#### Codesequenz 4.1: SQL-Statement zum Finden von Synonymen aus Tripel-Tabelle

```
1 SELECT 'objekt' AS 'TERM', 'praedikat' AS 'P', 'subjekt' AS 'SYN', 'ID'
2 FROM Beziehung,
3     (SELECT 'subjekt' as ID
4     FROM Beziehung
5     WHERE 'praedikat' = "hatObjekt" AND 'objekt' = "Tor"
6     ) AS beziehungsID
7 WHERE 'objekt' = "Tor" AND 'praedikat' = "istSynonymZu"
8 AND ID IN
9     (SELECT 'objekt' AS 'ID'
10    FROM Beziehung
11    WHERE 'subjekt' = "porta" AND 'praedikat' = "akzeptiert")
12
13 UNION
14
15 SELECT 'subjekt' AS 'TERM', 'praedikat' AS 'P', 'objekt' AS 'SYN', 'ID'
16 FROM Beziehung,
17     (SELECT 'subjekt' as 'ID'
18     FROM Beziehung
19     WHERE 'praedikat' = "hatSubjekt" AND 'objekt' = "Tor"
20     ) AS beziehungsID
21 WHERE 'subjekt' = "Tor"
22 AND 'praedikat' = "istSynonymZu"
23 AND ID IN
24     (SELECT 'objekt' AS 'ID'
25    FROM Beziehung
26    WHERE 'subjekt' = "porta" AND 'praedikat' = "akzeptiert")
```

Die Komplexität dieses Statements zeigt, dass die Suche nach Synonymen mit dieser Technik aufwändiger ist, als wenn für die Synonymie eine eigenständige, «traditionelle» Tabelle geführt wird (siehe Codesequenz 4.2).

#### Codesequenz 4.2: SQL-Statement zum Finden von Synonymen (traditionell)

```
1 SELECT 'deutsch'.'deutsch' AS 'TERM', 'syn'.'deutsch_syn' AS 'SYN'
2 FROM deutsch, ist_synonym_zu, deutsch AS syn
3 WHERE 'deutsch'.'deutsch' = "Tor"
4 AND 'deutsch'.'deutschID' = 'ist_synonym_zu'.'deutschID_FK'
5 AND 'ist_synonym_zu'.'deutschIDIstSynonym_FK' = 'syn.deutschID'
6 AND 'ist_synonym_zu'.'VokabelID_FK' = 1
```

Der interne Aufwand des Datebankservers wäre noch zu untersuchen, dürfte aber in beiden Fällen gering sein, da die Spalten, in denen gesucht wird, indiziert sind und die Joins jeweils nur zwischen temporären Tabellen mit sehr wenigen Zeilen gemacht werden (vgl. [Mei07, 98 ff]). Deshalb darf der Zusatzaufwand in Kauf genommen werden für den Vorteil, dass

die Arten von Beziehungen zwischen Wörtern nicht von vornherein feststehen müssen.

Allerdings muss man, wenn die Daten in solchen Tripeln gespeichert werden, von Anfang an dafür sorgen, dass nicht ein Wildwuchs an Prädikaten entsteht, sonst wird es extrem schwierig, aus der Tabelle überhaupt noch wertvolle Information zu extrahieren. Für grossangelegte Anwendungen im Semantic Web wird das durch eine recht strikt festgelegte Syntax und klar definierte Namensräume erreicht ([HKRY08, 39 ff]). Im Rahmen dieser Arbeit reicht es aus, dass die Benutzer die Prädikate nicht frei wählen können, sondern aus einem Menü übernehmen müssen. Für die erste Version des Programms werden nur die in Tabelle 4.5.2 vorhandenen Prädikate erlaubt; im Prototypen ist das Interface zur Bearbeitung der Beziehungen noch nicht implementiert.

### 4.5.3. Tabellen mit Benutzerdaten und Lernkontrolle

Ein weiterer Teil der Datenbank modelliert die Benutzer /-innen. Die Personendaten sämtlicher Personen sind in der Tabelle `person` zusammengefasst, die zu den Tabellen `SchülerIn` bzw. `LehrerIn` in der Beziehung einer überlappend-vollständigen Generalisierung steht. Schülerinnen oder Schüler gehören mindestens einer Lerngruppe an, Lehrerinnen oder Lehrer unterrichten mindestens eine Lerngruppe. Jede Lerngruppe hat 0, 1 oder mehrere Pensen behandelt, und jede Schülerin / jeder Schüler lernt 0, 1 oder mehrere (viele!) Vokabeln. Die Beziehung `'lernt'` enthält weiter Informationen über die Stufe, die jemand im Lernen einer Vokabel erreicht hat (siehe S. 6) und das Datum, in der er/sie dieses Stufe erreicht hat. Die Gesamtheit der Entitäten und Beziehungen ist ersichtlich im ERM auf Abbildung 4.3 .

## 4.6. Die Übungssteuerung

Für die Steuerung des Trainings ist eine eigene PHP-Klasse zuständig. Ein Objekt dieser Klasse nimmt im Konstruktor Objekte der Klassen `Benutzer`, `Sprache`, `PostDataChecker`, und `Mysqli` entgegen, dazu Einstellungen wie die Wahl des Trainingsmodus und des gewünschten Trainingsinhaltes.

Ihre wichtigste Methode ist `memoriereNaechsteVokabel()`: Sie bildet Gruppen von Vokabeln (Grösse momentan fix auf 3, in zukünftigen Versionen wählbar), die zusammen je nach Trainingsmodus zuerst zum Memorieren, dann zum Repetieren vorgelegt werden. Sie gibt ein Array zurück:

1. die zu memorierende oder zu repetierende Form
2. das Wort-Objekt (Vgl. Kap. 4.7)
3. die String «Memoriere» oder «Repetiere». Diese String dient dem Script des Web-Interface als Flag zum Auswählen des Präsentiermodus: Zum Memorieren soll die

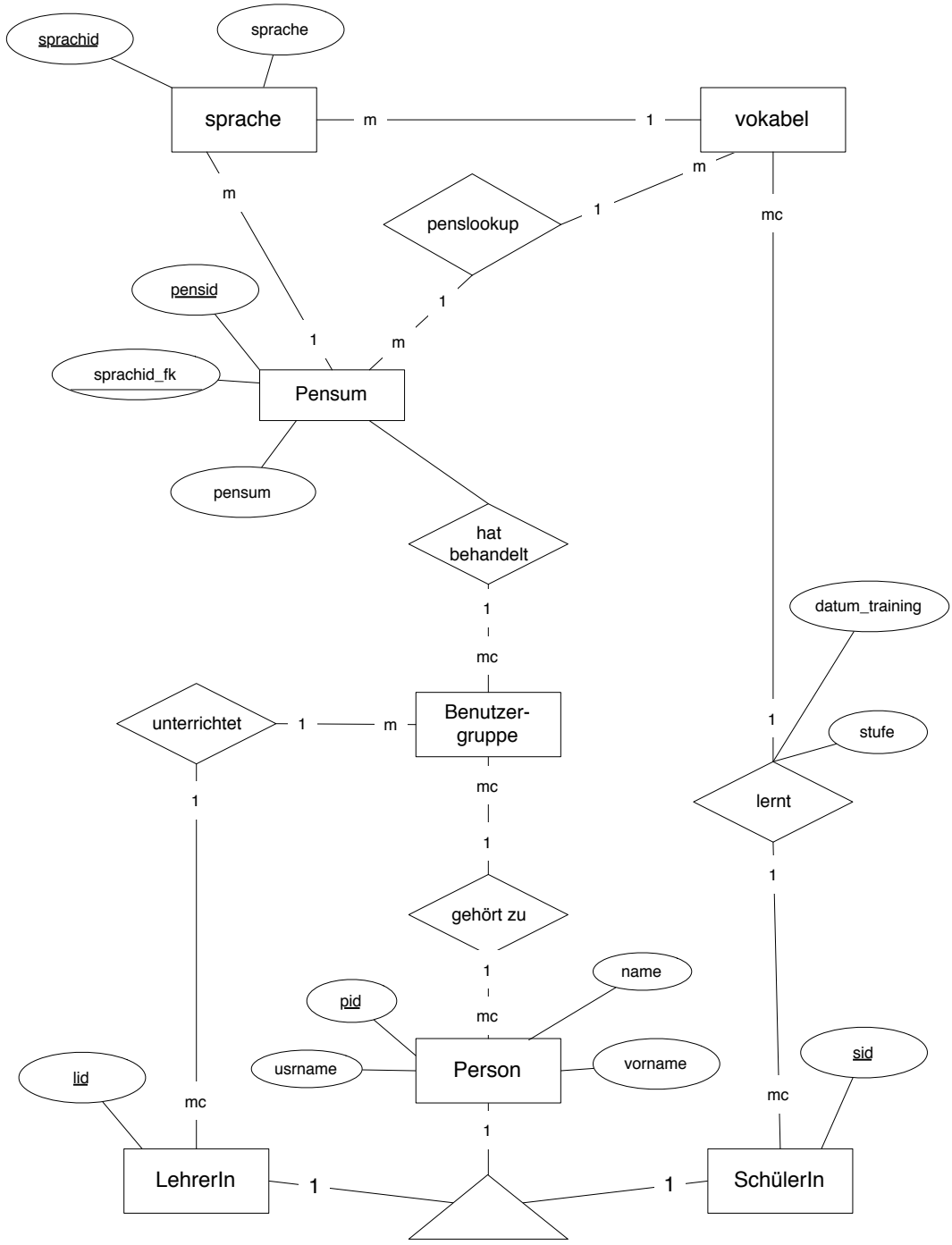


Abbildung 4.3.: ERM Vokabeldatenbank Teil 2: Benutzer, Gruppen, Lernkontrolle

Vokabel mit allen Angaben und Bedeutungen gross präsentiert werden, zum Repetieren braucht es Texteingabefelder und die Prüf-Funktionalität.

Beim Memorieren enthält das erste Element der Rückgabe jeweils das Lemma mit allen Angaben (Flexionsformen etc), beim Repetieren nur das Lemma allein. Beim Repetieren werden die gewünschten Stammformen der Verben als zusätzliche Vokabeln zurückgegeben; das erste Element der Rückgabe enthält in diesem Fall die jeweilige Stammform, die übrigen Elemente wiederum das Wort-Objekt und das Modus-Flag.

## 4.7. Modellierung einer Vokabel als PHP-Klasse

Die PHP-Klasse Wort enthält all diejenigen Eigenschaften, die für Wörter mehrerer Sprachen relevant sind – etwa die deutschen Bedeutungen, Wortartzuordnung, Deklinationsformen, abgeleitete Fremdwörter. Sie bietet umfangreiche Methoden zu deren Anzeige und Bearbeitung an.

Zur Klasse Wort gibt es für jede Sprache eine Subklasse. Sie enthält die exklusiven Eigenschaften von Wörtern der jeweiligen Sprache – etwa die vielen Stammformen griechischer Verben – und stellt auch die Methoden zu ihrer Anzeige und Bearbeitung zur Verfügung.

Die Klasse Wort und ihre Subklassen nehmen im Konstruktor Objekte der Typen Mysqli und PostDataChecker entgegen. Die Übergabe des Mysqli-Objekts ist besonders wichtig: Wenn nämlich jedes Wort-Objekt sein eigenes Mysqli-Objekt erzeugte, würden für die Anzeige von Wörterlisten in kürzester Zeit Dutzende von Verbindungen zum Mysqli-Server aufgebaut, was dieser in den meisten Installationen nicht zulässt.

Die Scripts und Klassen, die Wort-Objekte einer bestimmten Sprache erzeugen sollen, halten ein Sprach-Objekt der anfangs ausgewählte Sprache in einer Variablen und nutzen es zum dynamischen Erzeugen eines Wort-Objekts der jeweiligen Sprache. Die Codesequenz 4.3 zeigt dies.

Codesequenz 4.3: PHP-Code zum Erzeugen eines Wort-Objekts

```
1 // Das Sprach-Objekt aus dem Session-Array laden
2 $sprache = $_SESSION['sprache'];
3 // Die Klartext-Bezeichnung der Sprache als String
4 // in eine Variable laden:
5 $sprachString = $sprache->getSprache();
6 // Ein Wort-Objekt erzeugen.
7 // "$sprachString" wird nach "griechisch"
8 // oder "lateinisch" aufgelöst.
9 $wort = new $sprachString($mysqli, $postDataChecker);
```

## 4.8. Prüfung von Wortbedeutungen: Die Klasse Pruefer

Die Klasse Pruefer fasst alle Vorgänge zur Überprüfung von Wortbedeutungen zusammen. Sie enthält eine einzige öffentliche Methode: `deutschPruefen(array)`. Diese Methode führt alle Prüfungen der Eingaben gemäss Kapitel 4.1 durch. Die wichtigsten davon werden in der Folge erläutert.

## 4.9. Verfahren zur tippfehler-toleranten Prüfung von Eingaben

### 4.9.1. Erkennung von Buchstabendrehern, Vergleich nach Kölner Phonetik: Die Klasse Alphabetizer

In Kapitel 3.1.2 wurde bereits erklärt, dass für die Erkennung von Buchstabendrehern die Eingabe-Strings und die richtigen Lösungen so permutiert werden, dass die Buchstaben in alphabetischer Reihenfolge stehen. Dafür ist die Klasse Alphabetizer zuständig.

Sie enthält hauptsächlich zwei Methoden: Die Methode `string alphabetize(string)`<sup>2</sup> und die Methode `string colognePhon(string)` ([Zim08]). Die erste dient dem alphabetischen Sortieren der Buchstaben eines Wortes, die zweite der Berechnung des Kölner-Phonetik-Wertes zu einer Zeichenkette (siehe 3.1.4).

Die Methode `alphabetiNormalize(string)` entfernt zusätzlich Leerschläge und nicht-alphanumerische Zeichen aus den Strings, die Methode `normalize(string)` entfernt diese Zeichen, ohne die String zu «alphabetisieren».

Hier wird Methode `alphabetize(string)` mit vorhergehendem `initialize()` kurz vorgestellt (vgl. Codesequenz 4.4):

Zuerst wird in der Methode `initialize()` auf Zeilen 5 und 6 ein Array zum Zählen der Buchstaben initialisiert: Die Schlüssel sind die möglichen Zahlenwerte der Buchstaben in ISO-8859-1-Codierung (PHP 5 versteht leider noch kein UTF-8!), die Werte werden alle auf 0 gesetzt.

In der Methode `alphabetize(string)` wird zuerst zu jedem Character der Eingabestring der Zahlenwert nach ISO-8859-1 ermittelt, z.B. für ein a der Wert 97. In diesem Beispiel wird darauf im Array `$charArray` der Wert auf Position 97 um 1 erhöht.

In der zweiten Phase wird `$charArray` durchlaufen; ist ein Wert grösser als 0, so wird sein Schlüssel in einen Character umgewandelt und so oft der Ausgabestring hinzugefügt, wie der Wert angibt. In unserem Beispiel ist der Wert auf Position 97 eine 1, also wird der Buchstabe a 1-mal zur Ausgabestring hinzugefügt.

Damit die Gross- und Kleinbuchstaben nicht getrennt gezählt und ausgegeben werden,

---

<sup>2</sup>Die Idee zu dieser Methode verdanke ich der Vorlesung «Programmieren 1» von Prof. Helmar Burkhart, Uni Basel

#### Codesequenz 4.4: Zwei Methoden der Klasse Alphabetizer

```
1 private $charArray = array();
2
3 private function initialize()
4 {
5     for ($ch = 0; $ch < 256; $ch ++) {
6         $this->charArray[$ch] = 0;
7     }
8 } // Ende der Methode initialize
9
10 public function alphabetize($wort) {
11     $this->initialize();
12     $wort_iso = utf8_decode($wort);
13     // Buchstaben zaehlen:
14     for ($i = 0; $i < strlen($wort_iso); $i++) {
15         $character = substr($wort_iso,$i,1);
16         $this->charArray[ord($character)] ++;
17     }
18     // Buchstaben in alphabetischer Reihenfolge ausgeben:
19     $ausgabe = "";
20     for ($i = 0; $i < 256; $i ++) {
21         while($this->charArray[$i] > 0) {
22             $ausgabe .= utf8_encode(chr($i));
23             $this->charArray[$i] --;
24         }
25     }
26     return $ausgabe;
27 } // Ende der Methode alphabetize
```

muss eine Zeichenkette, bevor sie dieser Methode übergeben wird, in Kleinbuchstaben umgewandelt werden.

## 4.9.2. Identifikation weiterer Tippfehler

In gewissen Fällen enthält ein eingegebenes Wort nicht dieselben Buchstaben wie eine richtige Lösung, ist nicht synonym zu einer richtigen Lösung und hat auch nicht denselben Wert nach Kölner Phonetik. In diesen Fällen ist es sinnvoll, mittels des Levenshtein-Algorithmus die Nähe zu gespeicherten Wörtern zu prüfen: Solange die Eingabe näher bei einer richtigen Bedeutung als bei einem anderen in der Datenbank gespeicherten Wort liegt, wird sie akzeptiert.

Zu diesem Zweck wird erst einmal das Minimum der Levenshtein-Distanzen zu den als richtig gespeicherten Bedeutungen gesucht – siehe Codesequenz 4.5:

Codesequenz 4.5: PHP-Code zum Messen der geringsten Levenshtein-Distanz

```
1 // Diejenige Bedeutung suchen ,
2 // zu der die Levenshtein-Distanz am kleinsten ist.
3 $levenshteinVergleiche = array();
4 $levenshteinVergleiche[0] = 100;
5 $eingabeNormal = $this->alpha->normalize($bedeutung)
6 for($i = 0; $i < count($this->deutschStrings); $i ++ ) {
7     $lev = levenshtein($eingabeNormal ,
8         $this->alpha->normalize($this->deutschStrings[$i]))
9     if(($lev < $levenshteinVergleiche[0]) {
10         $levenshteinVergleiche[0] = $lev;
11         $levenshteinVergleiche[1] = $this->deutschStrings[$i];
12     }
13 }
```

Der hier abgedruckte Code ist Teil einer Methode der Klasse Pruefer. Die Variable `$bedeutung` wurde dieser Methode als Benutzer-Eingabe übergeben. Das Array `$levenshteinVergleiche` nimmt auf Index 0 die jeweils kürzeste Levenshtein-Distanz auf. Dieser Wert wird mit 100 initialisiert, damit der erste gemessene Wert auf jeden Fall gespeichert wird. Auf Index 1 wird es die jeweils nächst-gelegene deutsche Bedeutung aufnehmen. Die Benutzereingabe wird zuerst mithilfe der Methode `normalize(string)` der Klasse Alphabetizer «normalisiert» und in einer Variablen gespeichert.

In der Schleife wird darauf jede als richtig gespeicherte Bedeutung ebenfalls «normalisiert», und es wird ihre Levenshtein-Distanz zur String in `$eingabeNormal` berechnet. Ist sie kleiner als der aktuelle Wert in `$levenshteinVergleiche[0]`, wird sie an dessen Stelle gespeichert.

Der so ermittelte minimale Abstand wird in Relation zur Länge der betreffenden richtigen Bedeutung gesetzt. Er darf nicht grösser sein als  $\frac{1}{3}$  dieser Länge, aufgerundet auf eine ganze

Zahl. Sind die Abstände zu allen gespeicherten Bedeutungen grösser, so gilt die Eingabe als falsch. Der Wert  $\frac{1}{3}$  wurde in einer Testreihe mit der Klasse Pruefer als einigermaßen optimal ermittelt.

Ist der minimale Abstand im genannten Bereich, so werden aus der Datenbank alle deutschen Wörter geladen, die den gleichen Anfangsbuchstaben wie die Benutzereingabe haben, und deren Länge von derjenigen der Benutzereingabe um höchstens 1 abweicht. – Auch dieser Wert ist in Tests als relativ günstig ermittelt worden: Wählt man grössere Abweichungen, so muss das Programm unter Umständen Tausende von Levenshtein-Distanzen messen, wodurch es spürbar langsamer reagiert. Wählt man nur diejenigen Wörter aus, die gleich lang sind wie die Eingabe, so wird die Eingabe eines falschen Wortes mit zusätzlichen Tippfehlern unter Umständen als richtig akzeptiert.

Die so gewonnene Auswahl deutscher Wörter wird nun so lange durchlaufen, bis ein Wort gefunden wird, dessen Levenshtein-Distanz zur Benutzereingabe kleiner ist als das vorhin gespeicherte Minimum. Dies geschieht mit dem Code in Codesequenz 4.6.

Codesequenz 4.6: PHP-Code zum Suchen näher gelegener falscher Wörter

```

1 private function minLevenshteinSuchen(&$bedeutung ,
2                                     &$levenshteinVergleiche)
3 {
4     $return = false;
5     $minLaenge = strlen($bedeutung) - 1;
6     $maxLaenge = strlen($bedeutung) + 1;
7     $deutschSQL = "
8         SELECT deutsch
9         FROM deutsch
10        WHERE SUBSTRING(deutsch,1,1) LIKE ?
11        AND LENGTH(deutsch) >= $minLaenge
12        AND LENGTH(deutsch) <= $maxLaenge";
13    $stmt = $this->mysqli->prepare($deutschSQL);
14    $stmt->bind_param("s", substr($bedeutung,0,1));
15    $stmt->execute() or die("<h3>$stmt->error</h3>");
16    $stmt->bind_result($deutsch);
17    $eingabeNormal = $this->alpha->normalize($bedeutung);
18    while($stmt->fetch()) {
19        $lev = levenshtein($eingabeNormal,
20                          $this->alpha->normalize($deutsch));
21        if(($lev < $levenshteinVergleiche[0])) {
22            $levenshteinVergleiche[0] = $lev;
23            $levenshteinVergleiche[1] = $deutsch;
24            $return = true;
25            break;
26        }
27    }
28    $stmt->close();

```

```

29     return $return;
30 } // Ende der Methode minLevenshteinSuchen

```

Die in Codesequenz 4.6 gezeigte Methode empfängt als Parameter (`levenshteinVergleiche`) das vorhin generierte Array (siehe Codesequenz 4.5); es wird *by reference* übergeben, damit ein Schreibvorgang den ursprünglich gespeicherten Wert ändert. Die Methode funktioniert analog zum Code in Sequenz 4.5, mit dem einen Unterschied, dass die Schleife abgebrochen wird, sobald eine kleinere Levenshtein-Distanz gefunden wird als das bisher gespeicherte Minimum. – Es geht ja diesmal nicht darum, das absolute Minimum zu finden, sondern nur darum, zu prüfen, ob ein falsches Wort gefunden wird, das der eingabe näher liegt als ein richtiges.

Gibt diese Methode `true` zurück, so wird die Benutzereingabe als falsch taxiert.

In Codesequenz 4.6 sieht man übrigens auch ein Beispiel einer Datenbankabfrage mit dem `Mysqli`-Objekt von PHP5: In den Zeilen 8 – 12 wird die Querystring aufgebaut, wobei in Zeile 10 anstelle der Variablen, die für die Suche ausgewertet werden soll, der sogenannte *Parameter-Marker* `?` steht. Dieser wird in der Methode `bind_param` des `Mysqli-Statement`-Objekts ersetzt: Diese Methode verlangt zwei Parameter: eine Kennung für den gewünschten Datentyp und eine Variable. Sie prüft zuerst, ob die Variable dem gewünschten Datentyp (hier steht `s` für String) entspricht, und ersetzt dann den Parameter-Marker durch deren Inhalt.

In der Methode `bind_results` des `Mysqli-Statement`-Objekts (Zeile 16) wird festgelegt, an welche Variablen die Feld-Inhalte jeder gefundenen Tabellen-Zeile «gebunden» werden sollen. In der `while`-Schleife schliesslich wird mit `fetch` jeweils eine Zeile der Resultat-Tabelle ausgelesen.

## 4.10. Synonyme suchen: Die Klasse `Synonymfinder`

Die Klasse `Synonymfinder` bietet eine einzige öffentliche Methode an: `sucheSynonyme(string, string)`. Als ersten Parameter übergibt man der Methode ein Wort, zu dem Synonyme gesucht werden sollen; der zweite Parameter wird im hier vorgestellten Prototypen noch nicht verwendet. Bei einer Weiterentwicklung der Software (siehe Kap. 3.2.5 – 3.2.7) könnte damit der Dienst ausgewählt werden, über den die Synonyme gesucht werden sollen.

### 4.10.1. Synonym-Abfrage bei `OpenThesaurus.de`

Für die vorliegende Version des Vokabeltrainers wurde als Synonymquelle aus den in Kapitel 3.2.4 genannten Gründen der Webservice von `OpenThesaurus.de` genutzt. `OpenThesaurus` bietet einen REST-Dienst mit einfachem API an: Das gesuchte Wort wird direkt mit der http-Querystring übergeben, z.B: <http://www.openthesaurus.de/synonyme/search?q=gehen&format=text/xml>

OpenThesaurus liefert die Synonyme als XML-Daten zurück. Ein Beispiel für deren Struktur zeigt die Codesequenz 4.7.

#### Codesequenz 4.7: XML-Output von OpenThesaurus.de

```
1 <matches>
2   <metaData>
3
4   </metaData>
5   <synset id='2857'>
6     <categories></categories>
7     <term term='funktionieren' />
8     <term term='funzen' level='umgangssprachlich' />
9     <term term='gehen' />
10    <term term='klappen' level='umgangssprachlich' />
11    <term term='laufen' level='umgangssprachlich' />
12    <term term='tun' level='umgangssprachlich' />
13  </synset>
14  <synset id='9481'>
15    <categories></categories>
16    <term term='gehen' />
17    <term term='herumschlendern' />
18    <term term='herumwandern' />
19    <term term='latschen' level='umgangssprachlich' />
20    (...)
21  </synset>
22  <synset id='9627'>
23    <categories></categories>
24    <term term='gehen' />
25    <term term='umziehen' />
26  </synset>
27  <synset id='12009'>
28    <categories></categories>
29    <term term='gehen' level='umgangssprachlich' />
30    <term term='in Betracht kommen' />
31  </synset>
32 </matches>
```

Diese XML-Daten werden in PHP in ein Objekt vom Typ DOMDocument übernommen und können dann recht bequem geparkt und in ein zweidimensionales Array geschrieben werden. Die Codesequenz 4.8 zeigt, wie ein http-Request an den Server von OpenThesaurus generiert wird und wie die zurückgegebenen Daten geparkt werden.

#### Codesequenz 4.8: Teil der Methode sucheSynonyme der Klasse Synonymfinder

```
1 $wort = $this->checker->cleanString($wort);
2 $wort = rawurlencode($wort);
3 $url = sprintf("http://www.openthesaurus.de/
4     synonyme/search?q=%s&format=text/xml", $wort);
5 $c = curl_init($url);
6 curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);
7 curl_setopt($c, CURLOPT_USERAGENT, "Mozilla/5.0");
8 $page = curl_exec($c);
9 curl_close($c);
10
11 $dom = DOMDocument::loadXML($page);
12
13 $root = $dom->documentElement;
14 $synSetArray = $root->getElementsByTagName("synset");
15
16 foreach($synSetArray as $synset) {
17     $setelements = array();
18     if($synset->hasChildNodes()) {
19         $synnodes = $synset->getElementsByTagName("term");
20         foreach($synnodes as $term) {
21             $setelements[] = $term->getAttribute("term");
22         }
23         $synonymgruppen[] = $setelements;
24     }
25 }
```

In Zeile 1 von Codesequenz 4.8 wird das übernommene Wort sicherheitshalber durch den postDataChecker von gefährlichen Zeichen befreit, in Zeile 2 wird es so codiert, dass es den Transfer zum OpenThesaurus und zurück übersteht (siehe Kap. 4.10.2). In den Zeile 3 und 4 wird die http-Querystring aufgebaut, in den Zeilen 5 – 7 werden die Optionen für den http-Request gesetzt. (Dieser Teil des Codes stammt aus [ST05, 349 f].) In den Zeilen 16 – 24 wird der Inhalt der DOM-Knoten in einer Doppelschleife durchlaufen und in ein zweidimensionales Array geschrieben.

### 4.10.2. Eine Schwierigkeit: Konsistente Zeichensatzcodierung

Eine Schwierigkeit, die leider immer noch jedes Web-Projekt begleitet, ist, dass stets auf konsistente Zeichensatzcodierungen geachtet werden muss – zumindest solange nicht nur ASCII-Zeichen darzustellen sind. Hier sollen ein paar Punkte aufgelistet werden, deren Beachtung einem eine Menge Ärger ersparen kann:

- *Codierung von Querystrings in http-Requests*

Wird ein http-Request durch einen Webbrowser erzeugt, übernimmt dieser automatisch das «URL-Encoding» der mitgelieferten Parameter. Baut man ihn hingegen in

einem PHP-Programm auf, muss man darauf achten, dass die GET-Parameter oder die POST-Daten nicht tel quel gesandt, sondern zuvor mit der Funktion `urlencode` codiert werden. Enthalten die Daten, die mitgeschickt werden sollen, Sonderzeichen, so soll man die Funktion `rawurlencode` ([29]) verwenden. Der Grund dafür ist, dass das http-Protokoll in Querystrings bis vor sehr kurzer Zeit nur Buchstaben, Zahlen und einige wenige Interpunktionszeichen wie etwa den Underscore akzeptierte, und dass die meisten Server diese Beschränkung bis heute haben.

- *Konsistenz der Zeichencodierung zwischen Datenbank und Webseiten*

Es ist äusserst wichtig, dass man für jede Webseite, die man schreibt, innerhalb des `<head>`-Bereichs die Zeichencodierung angibt, und dass diese mit der Codierung aller Tabellen der Datenbank übereinstimmt. In HTML 5 ist die Syntax dafür sehr einfach geworden – siehe Codesequenz 4.9.

Codesequenz 4.9: HTML-5-Header

```
1 <!DOCTYPE html>
2 <html lang="de">
3 <head>
4   <meta charset="utf-8" />
5   <title>...</title>
6 </head>
```

Vergisst man, in Datenbank und Webseiten dieselbe Zeichencodierung zu erzwingen, hängt es von den Browser-Einstellungen der Benutzerin ab, ob z.B. griechische Schrift (oder auch nur deutsche Umlaute!) richtig angezeigt werden.

- Beim *Aufbau der Datenbank* muss man unbedingt darauf achten, dass man von Anfang an die richtige Codierung wählt, sonst muss man unter Umständen für jede Spalte jeder Tabelle nachträglich die richtige Codierung einstellen.
- Schliesslich ist leider PHP bis zur Version 5.3 nicht ganz UTF-8-kompatibel. Die normalen Stringfunktionen wie `strtolower` oder `substr` ergeben unerwartete Resultate, wenn man sie mit UTF-8-Strings füttert. Glücklicherweise stehen hierfür Ersatzfunktionen wie `mb strtolower` zur Verfügung ([28]).

Was leider noch gar nicht funktioniert, ist die Umwandlung von Mehr-Byte-Charakteren in Zahlen. Die Funktion `ord` ergibt nur für Buchstaben in der Codierung ISO-8859-1 die erwarteten Resultate. Die hier verwendete Methode `alphabetize` der Klasse `Alphabetizer` könnte also keine griechischen Wörter «alphabetisieren».

## 4.11. Web Interface

### 4.11.1. Startseite, Login, Navigation

Auf der Startseite wählt der Benutzer die Sprache aus, mit der er arbeiten will. Wenn die Sprache noch nicht ausgewählt worden ist, wird er von jeder anderen Seite aus auf die Startseite umgeleitet.

Zum Anmelden steht eine Login-Seite zur Verfügung. Ist eine Benutzerin angemeldet, erscheint in der Haupt-Navigation anstelle des Links zur Login-Seite ein Logout-Button.

Die Navigation zeigt jeweils diejenigen Bereiche der Anwendung, zu denen der Benutzer momentan Zugang hat: Ist noch keine Sprache gewählt, erscheinen nur die Links zur Start- zur Login- und zur Informationsseite.

Auf einem normalgrossen Bildschirm erscheint die oberste Ebene der Navigation als Balken oben im jeweiligen Browserfenster, Untermenues werden eingeblendet, sobald die Maus sich über dem entsprechenden Hauptmenue-Punkt befindet. Auf Kleingeräten erscheint das Menue momentan oben auf jeder Seite als Liste – mit Ausnahme des iPhone von Apple: Hier ist das Menue normalerweise ausgeblendet, es erscheint beim Berühren der Menue-Schaltfläche. Der CSS- und JavaScript-Code zu dieser Funktion wird in [Sta10, 13 ff] zur Verfügung gestellt.

### 4.11.2. Wörter suchen, anzeigen, exportieren

Nachdem der Benutzer auf der Startseite die Sprache, mit der er arbeiten will, ausgewählt hat, kann er auf zwei Wegen Vokabeln suchen:

- Ein oder mehrere Lernpensen aus einer Liste auswählen und die entsprechenden Wörter auf Knopfdruck anzeigen oder als \*.csv-Datei exportieren
- Den Anfang einer Vokabel eingeben. Mit einem AJAX-Request wird eine Liste aller Vokabeln erzeugt, die mit den schon eingegebenen Buchstaben beginnen. Ein Klick auf eine Vokabel zeigt diese mit allen Angaben.

Diese beiden Interface-Seiten waren schon vor Beginn dieser Arbeit implementiert, mussten aber, um mit den neu geschriebenen und überarbeiteten Klassen zu funktionieren, grundsätzlich überarbeitet werden.

### 4.11.3. Wörter trainieren

Der hier vorliegende Prototyp erlaubt zwei Arten von Training:

- Die Vokabeln des neusten Pensums memorieren und trainieren: Die Wörter werden dann in Dreiergruppen jeweils zuerst mit allen Angaben gezeigt und danach abgefragt. Zu jeder eingegebenen Wortbedeutung erscheint ein Feedback, nachdem eine Taste zum Prüfen der Eingabe geklickt worden ist.

- Diejenigen Vokabeln repetieren, die gemäss Trainingsplan zur Repetition fällig sind: Sie erscheinen nicht mit allen Angaben, sondern werden sofort abgefragt. Auch hier erscheint nach dem Klick auf die Prüftaste ein Feedback zu den eingegebenen Lösungen. Zeigt sich, dass der Benutzer ein Wort nicht mehr kennt, so wird der Lernfortschritt für dieses Wort auf Stufe 2 zurückgesetzt. Ist er auf Stufe 2 oder tiefer, so wird es auf Stufe 0 gesetzt, d.h. das Wort erscheint beim nächsten Training sofort wieder.

#### **4.11.4. Datenbank-Administration**

Dieser Teil des Benutzerinnen-Interface ist erst skizziert: Im jetzigen Zustand enthält die Software Webseiten für folgende Aufgaben:

- Zuteilung der Benutzer zu den Lerngruppen (Rolle: Lehrer)
- Bearbeitung einzelner Vokabeln (Rolle: Lehrer)

Für die Seite mit den Scripts zur Bearbeitung von Vokabeln gilt dasselbe wie für die Scripts zum Suchen und Anzeigen von Vokabeln: Sie waren schon vorhanden, mussten aber stark überarbeitet werden.

### **4.12. Schnittstellen zu anderen Datenbanken**

Die Informationen zur Datenbankverbindung, zum einzelnen Benutzer und zur Benutzergruppe werden in je einer PHP-Klasse verarbeitet. Deshalb kann die Software ohne allzu grosse Änderungen an eine bestehende Schul-Datenbank angebunden werden. Bei einer Installation sind folgende Anpassungen nötig:

- In der Klasse DBVerbindung müssen die Login-Daten und der Datenbankname angepasst werden.
- In den Klassen Benutzer und Benutzergruppe müssen die Querystrings den Gegebenheiten der Schuldatenbank angepasst werden. In den meisten Fällen wird es so sein, dass das Login-Prozedere vom Schul-Intranet bereits angeboten wird und dass die Software ein vorhandenes Session-Array nutzen kann.
- Bei den Zugriffen auf das Session-Array müssen die Bezeichnungen der Elemente, die auf Benutzerinnen und Gruppen verweisen, angepasst werden (sofern die Schul-Website über ein Intranet verfügt und die Information über den gerade eingeloggten Anwender in einer Session vorhält). Es wurde allerdings darauf geachtet, dass der Vokabeltrainer vorwiegend auf selbst generierte Elemente der Session zurückgreift. Alle notwendigen Änderungen könnten in den PHP-Scripts der Index-Seite vorgenommen werden.

- In der Klasse Trainingssteuerung muss die Methode neustesPensumLaden angepasst werden, da sie in der Datenbank auf die Beziehung zwischen Benutzerin und Benutzerinnengruppe zurückgreift.
- Die Seiten zur Benutzer- und Gruppenverwaltung können in der Regel gelöscht werden.

# 5. Zusammenfassung und Ausblick

## 5.1. Grundsätzliches

In dieser Arbeit konnte gezeigt werden, dass es möglich ist, einen Vokabeltrainer zu entwickeln, der nicht nur «Richtig» und «Falsch» kennt, sondern eine nicht genau so erwartete Benutzereingabe gleichsam im Feld aller möglichen Wörter lokalisiert und kontrolliert, ob sie genug nahe bei einer erwarteten Lösung ist: Wörter mit Tippfehlern werden, wenn sie nicht allzu viele Fehler enthalten, als vertippte richtige Wörter erkannt, und auch Synonyme erwarteter Lösungen werden identifiziert.

Andererseits wurden auch grundsätzliche Probleme bei der Identifikation von Synonymen gezeigt, namentlich das Problem, dass ein Vokabeltrainer einzelne Vokabeln, allenfalls kleine Beispielsätze, aber nie Texte enthält, in deren Zusammenhant Wortbedeutungen erst richtig deutlich werden.

Will man diesen Weg weiter gehen, so sollte man auf jeden Fall untersuchen, wie viel Text nötig ist, damit ein Programm prüfen kann, ob zwei deutsche Wörter in Bezug auf eine bestimmte fremdsprachliche Vokabel als Synonyme akzeptiert werden können. Möglicherweise wäre hierfür folgender Weg gangbar: Die Vokabel und die deutschen Bedeutungen werden in der Datenbank mit kurzen «Kollokationen» (Beispielsätzen) gespeichert. Für die automatische Suche nach Synonymen werden die Bedeutung der Vokabel und die deutschen Wörter, deren Synonymie vermutet wird, in Texten im World Wide Web gesucht. Diese Texte und die gespeicherte Kollokation müssten dann automatisch «getaggt» werden, wie dies [GHP10] zeigen. So liesse sich Information über Zusammenhänge gewinnen, in denen die Wörter stehen, und damit auch über deren genauere Bedeutung.

## 5.2. Weiterentwicklung des Prototypen

Will man den Prototypen zur Release-Reife weiterentwickeln, so wären folgende Schritte zu prüfen:

- Ein Wort ist ein Wort

Die Unterscheidung zwischen Vokabeln und deutschen Bedeutungen stammt aus dem Design der Datenbank, deren Entwicklung lange vor dem Verfassen dieser Arbeit begonnen worden war. Inhaltlich ist sie nach den aktuellen Weiterentwicklungen nicht mehr zu rechtfertigen. Man müsste die Datenbank so umgestalten, dass in einer Tabelle «Wort» sämtliche Angaben gehalten werden, die für Wörter aller Sprachen

(auch deutsche) relevant sind, und dass für jede Sprache eine spezialisierte Tabelle besteht, die die exklusiven Merkmale von Wörtern der jeweiligen Sprache enthält. Damit wäre auch eine Weiterentwicklung dahin möglich, dass z.B. eine Benutzerin französischer Muttersprache deutsche Vokabeln trainieren kann.

- Alternativ dazu könnte man die Datenbank auch umfassend flexibilisieren, sodass sie nur noch aus einer einzigen dreispaltigen Tabelle bestünde. Sämtliche Attribute von Entitäten wären dann Objekte einer Subjekt-Prädikat-Objekt-Beziehung: Über das Wort «dīcere» gäbe es dann z.B. eine Menge Aussagen in dieser Tabelle, wie: «dīcere» → «hatPerfektAktiv» → «dīxī». Dadurch käme sämtliche semantische Information aus den Metadaten in die Daten der Datenbank. Allerdings müssten dann die Abfragen wohl in SPARQL statt in SQL gemacht werden können, was wiederum die Auswahl des Hostings für einen Online-Vokabeltrainer stark einschränkt.
- Die Funktionalität des Programms könnte stärker zum Client hin verlegt werden. Dieser würde dann die Objekte, die länger benötigt werden, halten. Ist das Programm weiterhin als Web-Applikation konstruiert, so würde der Client mit dem Server hauptsächlich über AJAX kommunizieren, was auch den Datenverkehr nochmals reduziert. Seit HTML5 zumindest teilweise verfügbar ist, können Daten, die einmal geladen worden sind, auch lokal gespeichert werden.

Möglich wäre natürlich auch die Weiterentwicklung zu einer selbständigen Applikation. Diese müsste dann allerdings für verschiedene Gerätetypen je einzeln vorliegen.

- Natürlich müsste neben der technischen Dokumentation auch ein Benutzerhandbuch verfasst werden – dies besonders für diejenigen Personen, denen die Verwaltung der Datenbank erlaubt ist.
- Es müsste eine Einstellungsseite programmiert werden, die es erlaubt, sämtliche Anpassungen an eine bestimmte Installationsumgebung «von aussen» vorzunehmen, ohne dass im Code der Klassen und Scripts selbst etwas geändert wird. Dafür wären auch die betreffenden Klassen und Scripts anzupassen (Vgl. Kap. 4.12).

# A. Anhang

## **A.1. Beilagen**

Dieser Arbeit liegt eine CD-ROM bei, die folgendes Material enthält:

- Den Quellcode der PHP-Klassen und der PHP-Scripts
- Eine mit PHPDOC erzeugte HTML-Dokumentation aller PHP-Klassen
- Den SQL-Dump der Struktur der Datenbank

## **A.2. website**

Da die Software nur auf einem Server getestet werden kann, der auch die nötige Datenbank enthält, steht sie für Testzwecke zur Verfügung unter <http://vocabula.ch/trainer>.

# Literaturverzeichnis

- [Bar96] Bartels, Klaus: *Wie Berenike auf die Vernissage kam*. Darmstadt : Wissenschaftliche Buchgesellschaft, 1996
- [DR74] Duden-Redaktion (Hrsg.): *Das Fremdwörterbuch*. Mannheim : Bibliographisches Institut Mannheim, 1974
- [DS67] De Saussure, Ferdinand: *Grundfragen der allgemeinen Sprachwissenschaft*. 2. Auflage. Berlin : Walter De Gruyter, 1967
- [Ebn89] Ebner, G.: Wort-Arithmetik. Phoentische Ähnlichkeiten mit der Levenshtein-Distanz errechnet. In: *c't* 7/89 (1989), S. 192 – 208
- [End08] Endres, Wolfgang: *So macht Lernen Spass. Praktische Lerntipps für Schülerinnen und Schüler, Sek. I*. Weinheim und Basel : Beltz Verlag, 2008
- [GHP10] Gillmeier, Stephan ; Hengartner, Urs ; Pedrazzini, Sandro: Wie man mit der Wikipedia semantische Verfahren verbessern kann. In: *HMD – Praxis der Wirtschaftsinformatik* 47 (2010), Februar, Nr. 271, S. 70 – 80
- [GN05] Gurevych, Iryna ; Niederlich, Hendrik: Accessing GermaNet data and computing semantic relatedness. In: *Proceedings of the ACL 2005 on Interactive poster and demonstration sessions*. Morristown, NJ, USA : Association for Computational Linguistics, 2005, 5–8
- [Gri04] Grimm, Wilhelm Jacob und G. Jacob und Grimm: *Deutsches Wörterbuch. Elektronische Ausgabe der Erstbearbeitung*. Frankfurt/Main : Zweitausendeins, 2004
- [HKRY08] Hitzler, Pascal ; Krötzsch, Markus ; Rudolph, Sebastian ; York, Sure: *Semantic Web*. Berlin : Springer, 2008
- [KL07] Kunze, Claudia ; Lemnitzer, Lothar: *Computerlexikographie. Eine Einführung*. Tübingen : Narr, 2007
- [Klu02] Kluge, Friedrich: *Etymologisches Wörterbuch der Deutschen Sprache*. 24. Auflage. Berlin : de Gruyter, 2002
- [Lot72] Lotman, Ju. M.: *Die Struktur literarischer Texte*. 2. Auflage. München : Wilhelm Fink Verlag, 1972

- [Mei07] Meier, Andreas: *Relationale und Postrelationale Datenbanken*. Berlin : Springer, 2007
- [Mic94] Michael, Jörg: Joker im Spiel. Erweiterung der Levenshein-Funktion auf Wildcards. In: *c't* 3/94 (1994), Nr. 3, S. 230 ff
- [Mic07] Michael, Jörg: Von Hintz und Kunz. Ein Programmpaket zur fehlertoleranten Anschriftensuche. In: *c't* 20 (2007), S. 214 – 219
- [PK10] Portmann, Edy ; Kuhn, Adrian: Extraktion und kartografische Visualisierung von Informationen aus Weblogs. In: *HMD – Praxis der Wirtschaftsinformatik* 47 (2010), Februar, Nr. 271, S. 81 – 90
- [Pla90] *Kapitel* Kratylos. In: Platon: *Werke in 8 Bänden, griechisch und deutsch*. Bd. 3. Wissenschaftliche Buchgesellschaft, 1990, S. 397 – 575
- [SET09] Segaran, Toby ; Evans, Colin ; Taylor, Jamie ; Treseler, Mary E. (Hrsg.): *Programming the semantic web*. Sebastopol, CA : O'Reilly, 2009
- [ST05] Sklar, David ; Trachtenberg, Adam: *PHP 5 Kochbuch*. Köln : O'Reilly, 2005
- [Sta10] Stark, Jonathan ; Jepson, Brian (Hrsg.): *Building iPhone Apps with HTML, CSS, and JavaScript*. Sebastopol, CA : O'Reilly, 2010
- [Ves78] Vester, Frederic: *Denken, Lernen, Vergessen*. München : dtv, 1978
- [WSU06] Wirth, Theo ; Seidl, Christian ; Utzinger, Christian: *Sprache und Allgemeinbildung. Neue und alte Wege für den alt- und modernsprachlichen Unterricht am Gymnasium*. Zürich : Lehrmittelverlag des Kantons Zürich, 2006

# Benutzte Web-Ressourcen

- [Fie00] Fielding, Roy T.: *Representational State Transfer (REST)*. Website, 2000. – [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm) (accessed: 22. November 2010)
- [Med J] MediaWiki: *MediaWiki: API*. Website, o. J.. – <http://www.mediawiki.org/wiki/API> (accessed: 8. November 2010)
- [Nab Ja] Naber, Daniel: *FAQ zum OpenThesaurus*. Website, o. J.. – <http://www.openthesaurus.de/about/faq> (accessed: 13. November 2010)
- [Nab Jb] Naber, Daniel: *OpenThesaurus.de: API-Zugriff*. Website, o. J.. – <http://www.openthesaurus.de/about/api> (accessed: 11. November 2010)
- [NS08] Nastase, Vivi ; Strube, Michael: *Decoding Wikipedia Categories for Knowledge Acquisition*. Website, 2008. – <http://www.h-its.org/english/research/nlp/papers/nastase08b.pdf> (accessed: 8. November 2010)
- [28] (php.net): *PHP-Manual, Dokumentation der Funktion mb\_strtolower*. Website, 2010. – <http://ch2.php.net/manual/de/function.mb-strtolower.php> (accessed: 10. September 2010)
- [29] (php.net): *PHP-Manual, Dokumentation der Funktion rawurlencode*. 2010. – <http://ch.php.net/manual/de/function.rawurlencode.php> (accessed: 22. November 2010)
- [Tha06] Thalheimer, Will: *Spacing Learning Events Over Time: What the Research Says*. Website, 2006. – [http://www.phase-6.ch/opencms/system/galleries/download/lernsoftware/Spacing\\_Learning\\_Over\\_Time\\_\\_March2009v1\\_.pdf](http://www.phase-6.ch/opencms/system/galleries/download/lernsoftware/Spacing_Learning_Over_Time__March2009v1_.pdf) (accessed: 6. November 2010)
- [31] (W3C): *Semantic Web Development Tools*. Website, Januar 2010. – <http://www.w3.org/2001/sw/wiki/Tools> (accessed: 21. November 2010)
- [WDH<sup>+</sup>08] Wirth, Theo ; Diem, Robert ; Hartmann, Lucius ; Siegfried, Hanspeter ; Wangler, Clemens: *Computer-Einsatz im Altsprachlichen Unterricht*. Website, 2008. – [http://swisseduc.ch/altphilo/aktuell/docs/ICT\\_im\\_AU.pdf](http://swisseduc.ch/altphilo/aktuell/docs/ICT_im_AU.pdf) (accessed: 20. November 2010)

- [33] (Wikipedia): *Wikipedia s.v. Kölner Phonetik*. Website, o. J.. – [http://de.wikipedia.org/wiki/Kölner\\_Phonetik](http://de.wikipedia.org/wiki/Kölner_Phonetik) (accessed: 14. November 2010)
- [34] (Wikipedia): *Wikipedia s.v. Vergessenskurve*. Website, o. J.. – <http://de.wikipedia.org/wiki/Vergessenskurve> (accessed: 6. November 2010)
- [Zim08] Zimmer, Nicolas: *Forumsbeitrag zur soundex-Funktion in PHP*. Website, 2008. – <http://ch.php.net/manual/de/function.soundex.php> (accessed: 14. November 2010)

# Abbildungsverzeichnis

3.1.	Semiotisches Dreieck gemäss [WSU06, 52, fig. 2] . . . . .	15
3.2.	Bedeutungsspektren von lat. «portus» und dt. «Hafen» . . . . .	18
4.1.	ERM Vokabeldatenbank Teil 1: Vokabel, Angaben, Bedeutungen . . . . .	28
4.2.	Beispiel eines Graphen mit zwei Beziehungstypen . . . . .	29
4.3.	ERM Vokabeldatenbank Teil 2: Benutzer, Gruppen, Lernkontrolle . . . . .	33

# Codeverzeichnis

4.1. SQL-Statement zum Finden von Synonymen aus Tripel-Tabelle . . . . .	31
4.2. SQL-Statement zum Finden von Synonymen (traditionell) . . . . .	31
4.3. PHP-Code zum Erzeugen eines Wort-Objekts . . . . .	34
4.4. Zwei Methoden der Klasse Alphabetizer . . . . .	36
4.5. PHP-Code zum Messen der geringsten Levenshtein-Distanz . . . . .	37
4.6. PHP-Code zum Suchen näher gelegener falscher Wörter . . . . .	38
4.7. XML-Output von OpenThesaurus.de . . . . .	40
4.8. Teil der Methode sucheSynonyme der Klasse Synonymfinder . . . . .	41
4.9. HTML-5-Header . . . . .	42