

PROJEKTDOKUMENTATION

für

„Social Tagging, Folksonomien und Tag-Clouds in
ERP-Systemen“

Teil der Projektarbeit

„Social Tagging, Folksonomien und Tag-Clouds in ERP-Systemen“

im Rahmen des
Modules I 490 „Projektseminar“

an der

Hochschule für Technik und Wirtschaft Dresden



bei

Prof. Dr. rer. pol. Torsten Munkelt

Professur für Betriebliche Informationssysteme / Datenbanksysteme

Wintersemester 2016/2017

in den Studiengängen
Wirtschaftsinformatik (Bachelor/Diplom)

Studierende

Name	Bibl.-Nr.
Melanie Koester	s70275
Bastian Kieschnik	s70353
Kay Keller	s70362
Martin Siegfried Daebritz	s70384
Peter Blaskoda	s70406
Wolf	s72785
Jennifer Anders	s72806
Stephanie Scheffler	s72829

Lizenzbestimmungen



Creative Commons: Namensnennung und Weitergabe unter gleichen Bedingungen 4.0

<https://creativecommons.org/licenses/by-sa/4.0/>

Inhaltsverzeichnis

1 Projektauftrag	2
2 Stand der Technik	3
2.1 Die Filter im ERP-System	3
2.2 Entwicklungsvoraussetzungen	3
2.3 Gewählte Arbeitsmittel	4
3 Anforderungsanalyse	5
3.1 Lizenzen	5
3.2 Ermittlung der Anforderungen	5
3.2.1 Prinzipskizzen	6
3.2.2 Anforderungen	6
3.3 Entwurf der GUIs zur Eingabe von Tags	9
3.4 Folksonomie, Social Tagging und Tag-Clouds	13
3.4.1 Begriffsklärung	13
3.4.2 Social Tagging	14
3.4.3 Folksonomie	14
3.4.4 Tag-Clouds	17
3.5 Algorithmus Pseudocode	17
3.6 Fallbeispiel	18
3.7 Herangehensweisen der Tag-Cloud-Erstellung	21
3.7.1 Anforderungen / Grundlagen vor der Tag-Cloud-Erstellung	21
3.7.2 Spiralform-Verfahren	21
3.7.3 Hierarchical-Bounding-Boxes -Verfahren	21
4 Entwurf	23
4.1 Aufbau	23
4.1.1 Funktionen	23
4.1.2 Darstellung	24
4.1.3 Komponenten	26
4.1.4 Benutzeroberfläche	29
4.2 Konstruktion eines nachvollziehbaren Beispiels zur Präsentation	30
4.2.1 Ausgangslage	30
4.2.2 Schritt 1: Benutzer klickt auf das Tag „stark“	30
4.2.3 Schritt 2: Benutzer klickt auf das Tag „mutig“	32

4.2.4	Schritt 3: Benutzer klickt auf das Tag „clever“	33
4.2.5	Schritt 4: Benutzer sucht nach dem Begriff „Munkelt“.	34
4.2.6	Ergebnis	35
5	Implementierung	36
5.1	WordCloud-Komponente	36
5.1.1	Beschreibung	36
5.1.2	Aufbau	36
5.1.3	Verteilung	38
5.2	Implementierung im PPSn System	39
5.2.1	Beschreibung	39
5.2.2	Layout	39
5.2.3	Integration der WordCloud Komponente	40
6	Performance-Tests	45
6.1	Veränderte Parameter der spiralförmigen Tag-Cloud-Erstellung	45
6.2	Aussehen einer optimalen Tag-Cloud	46
6.3	Umsetzung im Algorithmus	46
6.4	Erweiterung des Spiralform-Verfahrens mit dem Halbierungsverfahren	48
6.5	Performance-Tests und Auswertung	49
6.6	Entscheidung zur Performance	50
7	Text Mining	51
7.1	Aufgabe	51
7.2	Vorgehen	51
7.3	Mögliche Lösungsansätze	52
7.3.1	Stammformreduktion	52
7.3.2	Lemmatisierung	53
7.3.3	TreeTagger	55
7.3.4	Wortschatzprojekt Uni Leipzig	56
8	Fazit	61
8.1	Text Mining	61
Quellen nachweise		64

1 Projektauftrag

Im Rahmen eines Projektseminars wurde die Aufgabe (Munkelt 2016) gestellt, eine Tag-Cloud innerhalb eines sich in Entwicklung befindenden ERP-Systems zu implementieren. Die Umsetzung erfolgt im Verlauf des Wintersemesters 2016/2017, in Zusammenarbeit mit dem ERP-Hersteller TecWare. Unterstützt werden die Studenten der HTW Dresden vom Softwareentwickler *neolithos*¹, der Firma TecWare und Herrn Professor Dr. Torsten Munkelt².

¹ Er bevorzugt auf Nachfrage in der Dokumentation beim "Handle" genannt zu werden
<https://github.com/neolithos>

² <https://www.htw-dresden.de/fakultaet-informatikmathematik/personal/professuren/prof-dr-torsten-munkelt.html>

2 Stand der Technik

2.1 Die Filter im ERP-System

Das noch in Entwicklung befindliche ERP-System¹, der Firma TecWare, besitzt aktuell drei Filter:

1. *Kategorienfilter*: Dieser Filter ist verantwortlich für die Auswahl nach sogenannten Kategorien, wie z.B. Auftrag und Artikel.
2. *Subfilter*: Mit Hilfe dieses Filters, können innerhalb der Kategorien sogenannte Unterkategorien ausgewählt werden. Ein Beispiel hierfür wäre das Filtern nach der Unterkategorie Kunde oder Lieferant, innerhalb der Kategorie Kontakt.
3. *Volltextfilter*: Anhand dessen kann der Nutzer ein Textfragment eingeben.

2.2 Entwicklungsvoraussetzungen

Um an dem ERP-System arbeiten zu können, sind folgende Voraussetzungen für die Entwicklungsumgebung gegeben:

- Microsoft Windows 8.1 oder neuer
- Microsoft Visual Studio 2015
- Microsoft SQL Server 2016 – LocalDB
- Microsoft SQL Server Managementstudio 2016
- TortoiseGit (oder ein Git-Client der Wahl, muss External/Submodules unterstützen)
- SQLite Datenbank Viewer (z.B. **DB Browser for SQLite**)

¹ <https://github.com/twdes/>

2.3 Gewählte Arbeitsmittel

Zur Verwaltung des Quellcodes wird aufgrund des bestehenden Projektes der Firma TecWare *Git* und als Hoster für die Repositories *GitHub*² verwendet.

Zur Verwaltung der Aufgaben wurde am ersten Termin *Trello*³ festgelegt. Damit sind Notizen für alle Teilnehmer auch mobil erreichbar und auch für Ungeübte im Umgang mit Git zugänglich.

Im Raum A111 der HTW stand uns ein Rechner mit Möglichkeit zur Installation des ERP neben der Entwicklungsumgebung bereit. Dieser wurde insb. durch die Aufgaben zum Textmining auch via Remote Desktop (RDP) genutzt, die Entwicklung in C# auf lokalen Maschinen u.a. im Laborbereich oder dem eigenen Notebook leichter zu handhaben ist.

² Projekt-Repository: <https://github.com/SemAhto/Projektdokumentation>

³ <https://trello.com/>

3 Anforderungsanalyse

3.1 Lizenzen

Das ERP-System der Firma TecWare wird unter der EUPL¹ angeboten. Dabei handelt es sich um eine Software-Lizenz, die die Kriterien der OSI² und der FSF³ erfüllt.

Im Gegensatz zur aktuellen Version 3 der GPL⁴ unterstützt die EUPL bereits seit der ersten Version über Abdeckung zu den Problemfelden Softwarepatente, Webservices und Gerichtsstand. Auf Nachfrage war insbesondere die eigene Festlegung des Gerichtsstandes für die Firma TecWare ausschlaggebend die von der Europäische Union herausgegebene Lizenz zu wählen statt einer der bisher viel weiter verbreiteten Alternativen wie der GPL, Apache.

Für unser Praxisprojekt ist insbesondere interessant, dass wir ggf. auf vorhandene Implementationen zurückgreifen könnten. Hierzu ist festzustellen, dass die EUPL kompatibel zur GPL ist⁵. Wir können damit im viralen Lizenzumfeld auch auf Software unter der GPL zugreifen und diese in ein Produkt unter EUPL einfließen lassen. Juristisch wäre man ansonst gezwungen erneut unter der ursprünglichen Lizenz zu veröffentlichen sofern die maßgeblichen Urheber nicht im dualen Verfahren auch unter der EUPL lizenziert würden. Die Übernahme von Code aus nicht-viralen Lizzenzen, wie der häufig anzutreffenden MIT Lizenz⁶ und zahlreichen ähnlichen, steht dahingegen keine Bedenken im Wege.

Weitere nicht zu unterschätzende Vorteile der EUPL sind die Übersetzung und rechtliche Gültigkeit in 28 Sprachen der EU sowie, dass der Textumfang – mit etwa der Hälfte der Länge im Vergleich zur GPLv3 – eher geeignet ist gelesen zu werden.

3.2 Ermittlung der Anforderungen

Während den ersten zwei Gruppenmeetings wurden der Entwurf von Anwendungsfällen für Folksonomien, Social Tagging und Tag-Clouds im ERP-System ausgearbeitet. Dabei wurde sich an der Satzschablone von Chris Rupp (vgl. Kluge 2013) orientiert. Anhand dessen wurden folgende Prinzipskizzen erstellt, wie auch Anforderungen schriftlich formuliert. Folgene Ergebnisse wurden dabei erzielt:

¹ <https://joinup.ec.europa.eu/community/eupl/description>

² Open Source Initiative: <https://opensource.org/>

³ Free Software Foundation: <https://www.fsf.org/>

⁴ GNU Geneal Public License: <http://www.gnu.de/documents/gpl.de.html>

⁵ <http://www.ifross.org/was-lizenzkompatibilitaet>

⁶ The MIT License <https://opensource.org/licenses/MIT>

3.2.1 Prinzipskizzen

Die Oberfläche von PPsn wurde im Groben nachempfunden ([Abbildung 3.1](#)).



Abb. 3.1: 1. Prinzipskizze

Durch Auswahl von Kathegoien soll die Tag-Cloud beeinflusst werden ([Abbildung 3.2](#)) können.

Durch die Auswahl von Tags soll die Tag-Cloud neu berechnet werden ([Abbildung 3.3](#)).

3.2.2 Anforderungen

Wir unterscheiden die folgenden Begriffe bei der Festlegung der Anforderungen.

Systemtags Tags, die bei Erstellung eines Objekts vom System generiert werden.

Usertags Tags, die vom User selbst erstellt und an das Objekt angehangen werden.

Tags Menge aller System- und Usertags.

Klassifikation:

0 = Text

1 = Datum

2 = Nummer

System:

1. Das System muss in der Lage sein, eine Vielzahl von Tags anzuzeigen.
2. Das System muss in der Lage sein, nach bestimmten Tags zu filtern.



Abb. 3.2: 2. Prinzipskizze

3. Das System muss in der Lage sein, eine Liste von Objekten (Bestellnummer, Artikel, Artikelnummer, Kundennummer) zu verkleinern, nachdem nach einem bestimmten Tag gefiltert wurde.
4. Das System muss in der Lage sein, einem Objekt Systemtags anzuhängen.
5. Das System muss in der Lage sein, aus den in der Liste angezeigten Tags, eine WordCloud zu generieren und anzuzeigen.
6. Das System muss in der Lage sein, Systemtags zu klassifizieren.
7. Das System muss in der Lage sein, zu einem einzelnen Objekt eine WordCloud zu generieren.
8. Das System muss in der Lage sein, in der Eingabemaske eine alphabetische Sortierung vorzunehmen.
9. Das System muss in der Lage sein, eine Filterliste in der Suchleiste zu generieren.
10. Das System muss in der Lage sein einen Suchverlauf anzuzeigen.

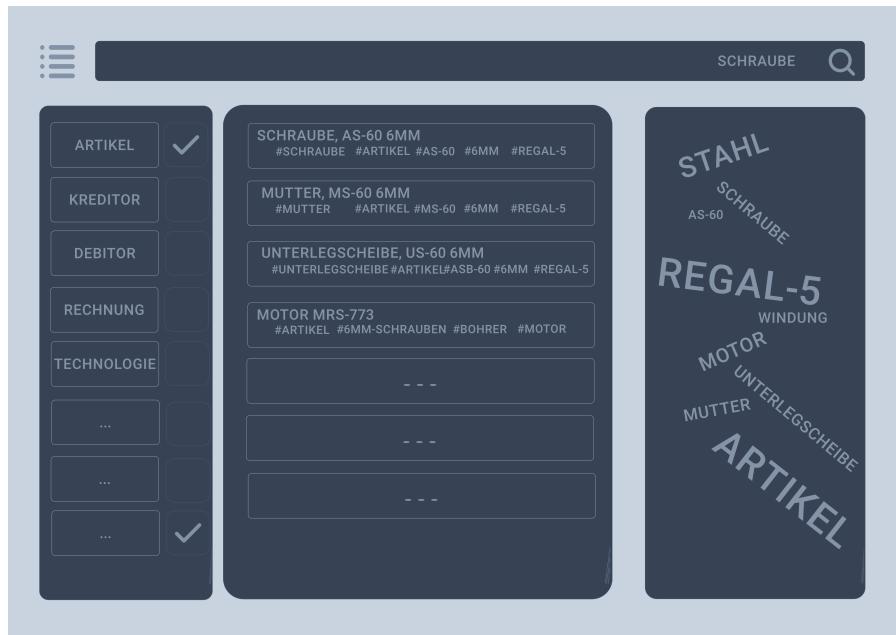


Abb. 3.3: 3. Prinzipskizze

Nutzer:

1. Der Nutzer muss in der Lage sein, nach Schlagworten in einer Eingabemaske zu suchen.
2. Der Nutzer muss in der Lage sein, eigene Tags an ein Objekt hinzuzufügen.
3. Der Nutzer muss in der Lage sein, Tags in der WordCloud ausblenden zu können.
4. Der Nutzer muss in der Lage sein, eigene Usertags zu ändern/löschen.
5. Der Nutzer muss in der Lage sein, Usertags zu klassifizieren.
6. Der Nutzer muss in der Lage sein, durch eine Interaktion einen Tag als Filter zu setzen.
7. Der Nutzer muss in der Lage sein, einen Tag auszuwählen.
8. Der Nutzer muss in der Lage sein, durch eine Interaktion einen Tag auszublenden.

Administrator:

1. Der Administrator muss in der Lage sein, Tags zu ändern/löschen.

3.3 Entwurf der GUIs zur Eingabe von Tags

Um eine Vorstellung von der zukünftigen Eingabe von Tags im ERP-System zu bekommen, wurden von den Studenten grafisch fünf GUIs entworfen. Diese sollten der Eingabe von Tags für sogenannte Datenobjekte dienen. Es wurde außerdem in Abbildung 3.4 ein Inline-Editor gegenüber einem Popup-Editor gestellt und eine [Beispielmaske mit Popup – Tag-Eingabe](#) vorgestellt.

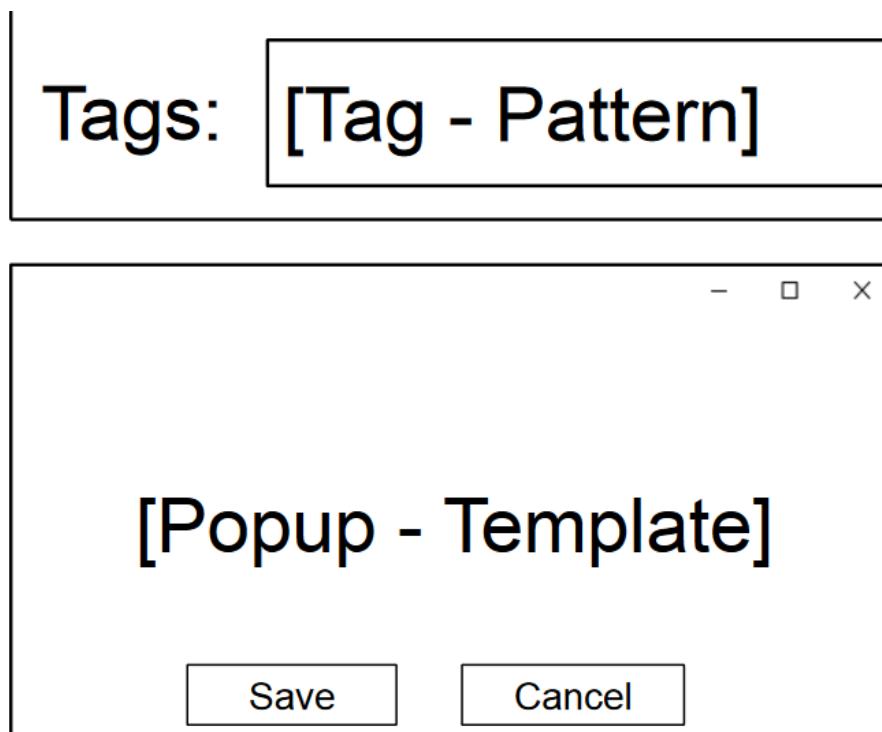


Abb. 3.4: Inline vs. Popup-Editor

Die Aufgabe der Studenten bestand darin, sich eine Form der Eingabemöglichkeit für Tags zu überlegen, wobei auch neue Dialoge entstehen können. Dabei wurden sowohl eine **Beispielmaske ohne Tag Eingabe** als auch eine **Beispielmaske mit Inline-Tag-Eingabe** entworfen.

The screenshot shows the 'Auftragsbearbeitung' application interface. The main window title is 'Auftragsbearbeitung'. The top menu bar includes 'Auftrag sichern', 'Auftrag neu', 'Folgемeldung neu', 'Auftrag drucken', 'Auftragsliste', 'Meldungsliste', and 'Selektion'. Below the menu, there are two tabs: 'Auftrag' (selected) and 'Abw. Kontierung'. The 'Auftrag' tab contains several input fields: 'Meldung' (empty), 'Meldender' (empty), 'Gew. Beginn' (13.05.2014), 'Auftrag' (empty), 'Auftragsart' (A6), 'Sonstige IH-Leistun.' (checkbox checked), 'Prio.' (empty), 'Techn. Platz' (SAA1-K), 'Kostenstellen-Abrechnung' (checkbox checked), 'Equipment' (empty), 'Verantw. ArbPl.' (ELT), '2002', 'IHLstArt' (027), 'Produktionsunterstützung' (checkbox checked), and 'Kurztext' (empty). On the right side of the 'Auftrag' tab, there are checkboxes for 'Dauerauftr.', 'Mat.Stat.', and 'Garantie'. Below the 'Auftrag' tab is a 'Rückmeldung' section with fields: 'PersNr' (1000), 'ArbPlatz' (ELT), 'IstStart' (13.05.2014), 'Uhrzeit' (11:42:10), 'Istarbeit' (checkbox checked), 'STD' (checkbox checked), and 'Fertig' (checkbox checked).

Abb. 3.5: Beispielmaske ohne Tag Eingabe

This screenshot is identical to Abb. 3.5, showing the 'Auftragsbearbeitung' application. The 'Auftrag' tab is selected. The 'Kurztext' field is highlighted with a red border, indicating it is the active field for entering inline tags. All other fields and sections appear the same as in Abb. 3.5.

Abb. 3.6: Beispielmaske mit Inline-Tag-Eingabe

Abb. 3.7: Beispielmaske mit Popup – Tag-Eingabe

Des Weiteren kümmerten sich die Studenten um Benutzereingabemöglichkeiten, wie eine Twitter-like Textbox (Inline), einen IDE-like Editor (Inline) und eine Evernote-like Tagbox (Inline).

#Foo #Bar #

Abb. 3.8: Twitter-like Textbox (Inline)

#Foo #Bar #Baz #Bar #

Blubb
Bums
Dings

Abb. 3.9: IDE-like Editor (Inline)



Abb. 3.10: Evernote-like Tagbox (Inline)

Eine Klassische Ansicht, wie in Abbildung 3.11 wurde ebenfalls entworfen.

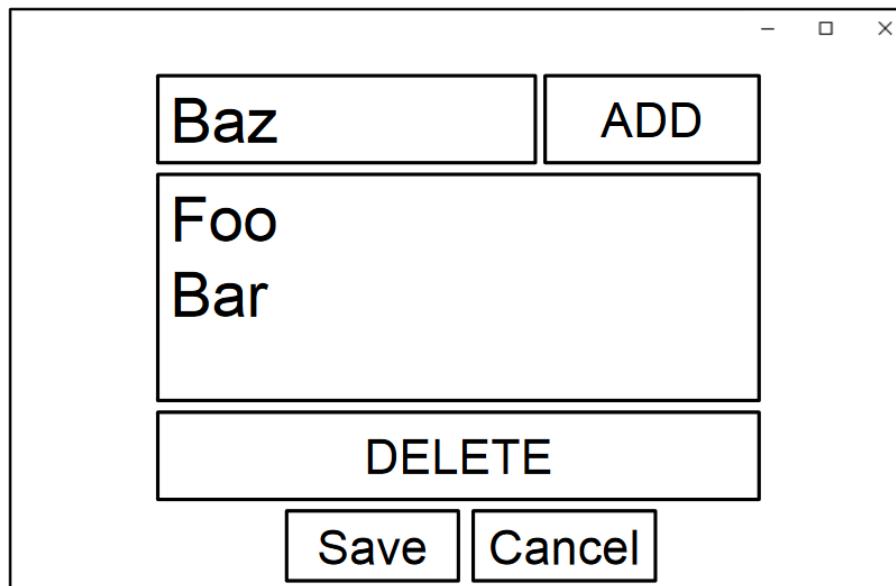


Abb. 3.11: Classic Listbox (Popup)

Die Ergebnisse wurden im Gruppenmeeting präsentiert und deren Verwendung, im Austausch mit neolithos diskutiert.

3.4 Folksonomie, Social Tagging und Tag-Clouds

3.4.1 Begriffsklärung

Bevor auf die Thematik der Folksonomie (Lohmann 2012) näher eingegangen wird, sollten ein paar Begriffe zum besseren Verständnis geklärt werden.

Social Network

Ein Social Network (GruenderszeneLexikon 2010) oder zu deutsch ein soziales Netzwerk wird in diesem Zusammenhang als eine Zusammenkunft verschiedener Nutzer im Internet bezeichnet. Diese Nutzer können miteinander interagieren und die Aktionen der anderen Nutzer nachvollziehen. Diese Community (eng. Gemeinschaft, Vereinigung) gibt es zum Teil mit sehr allgemeinen Interessen (z.B. Facebook) oder aber auch mit einem speziellen Ziel (z.B. XING zu Jobsuche und zum herstellen von Geschäftskontakten).

User

Ein User oder auch Nutzer ist ein Mitglied eines sozialen Netzwerkes. Dieser kann mit anderen Nutzern in dem social Network interagieren und verschiedene Aktionen ausführen. Nutzer können in verschiedene Nutzergruppen mit unterschiedlichen Berechtigungen unterteilt werden. So kann es zum Beispiel die normale Nutzer geben, Moderatoren und Administratoren. Dabei wären die Moderatoren verantwortlich für die Inhalte der Nutzer auf ihre Konformität mit den Regeln des Netzwerks zu prüfen und zum Beispiel Spam zu verhindern. Der Administrator hat in einem solchen Netzwerk meist eine eher technische Aufgabe. Ein User verfügt immer über ein bestimmtes Vokabular, welches durch sein Umfeld, sein intellektuelles Niveau und seine Kultur etc., geprägt ist.

Ressource

In Social Networks werden verschiedene Objekte/Ressourcen miteinander geteilt und gemeinsam bearbeitet oder verwendet. Solche Ressourcen können zum Beispiel Bilder, Musikdateien, aber auch Baupläne oder Lebensläufe sein. Meist sind die Inhalte auf bestimmte Ressourcen, welche mit dem Ziel des Social Networks zusammenhängt, beschränkt.

Tag

Ein Tag oder auch Schlagwort, Deskriptor oder Etikett ist in diesem Zusammenhang eine Form der unscharfen Klassifizierung beziehungsweise der Beschreibung von Ressourcen. Diese Tags sind meist willkürlich und unterliegen keinen strengen oder kontrollierten Regeln. Zum Beispiel bei der Plattform / dem Social Network *Instagram* werden Bilder von Nutzern geteilt und mit *Hashtags* beschrieben. Über diese *Hashtags* können andere Bilder mit den gleichen Bezeichnungen gefunden werden.

Unscharfe Suche

Eine unscharfe Suche ist die Möglichkeit eine bestimmte Ressource innerhalb eines Social Networks zu suchen, von welcher noch keine genaue Vorstellung besteht. Dabei werden die Suchergebnisse nur immer weiter eingegrenzt und es wird sich Stück für Stück an das gewünschte Ergebnis heran gearbeitet.

3.4.2 Social Tagging

Als Tagging oder auch taggen wird der Prozess des Tag an eine Ressource anhängen beschrieben (Indexieren). Beim Social Tagging (Wikipedia 2015) geschieht dies innerhalb eines Social Networks durch die User. Dadurch können User wiederum andere Ressourcen finden und die Ressourcen mit weiteren Tags versehen (Abbildung 3.12). Damit ist die Grundlage für eine unscharfe Suche geschaffen.

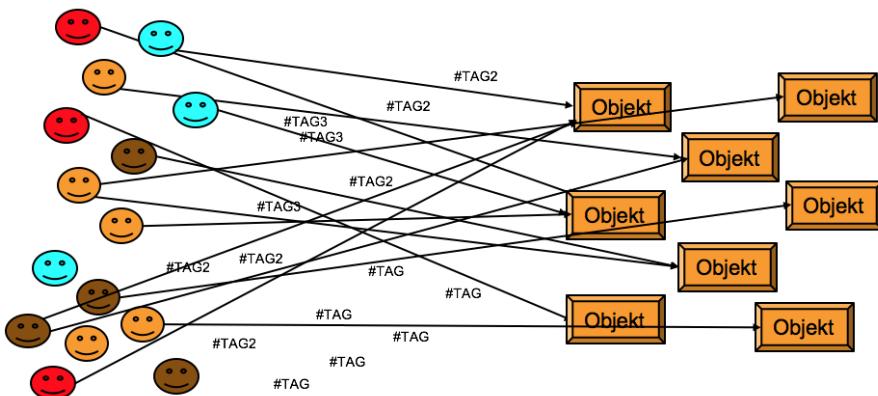


Abb. 3.12: Social Tagging

Beispiele für Social Networks welche das Social Tagging unterstützen gibt es viele, wie z.B. *Instagram*, *Youtube*, *del.icio.us*, etc..

3.4.3 Folksonomie

Das Kofferwort Folksonomie (Abbildung 3.13, Wal 2007) ist aus folgenden Bestandteilen zusammengesetzt: Folk - das Volk oder auch die Laien; Taxis - die Klassifikation; Nomos - das Management. Es bedeutet also soviel wie das Klassifikationsmanagement durch oder vom Volk aus. Der Begriff wurde erstmalig 2003 durch Van der Wal in einem Artikel erwähnt und maßgeblich durch ihn geprägt. Fast alle Artikel oder Quellen beziehen sich auf seine Theorie. Bei der Folksonomie (Mathes 2004) handelt es sich also, um die Beziehung von Tags, Ressourcen und Nutzern. Diese Verbindungen werden innerhalb eines Social Networks oder mittels einer sozialen Software erstellt und unterliegen keinen Regeln. Folksonomien entstehen durch den Prozess des Social Taggings und werden in immer mehr Systemen verwendet um den Nutzern eine zusätzliche Möglichkeit der Inhaltserschließung zu ermöglichen.

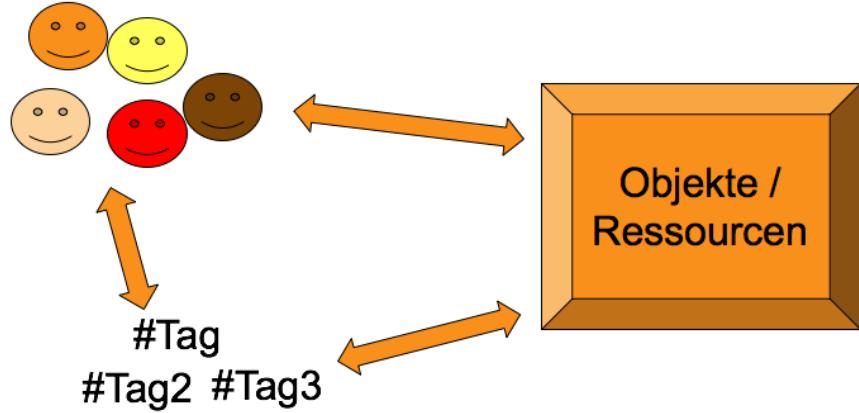


Abb. 3.13: Zusammenhang zw. User, Tags & Ressourcen

Die einzelnen Beziehungen zwischen Objekten, Nutzern und Tags können auf verschiedene Arten entstehen. Dabei unterscheidet man zwischen den Formen der Engen (Narrow) und der Breiten (Broad) Folksonomie.

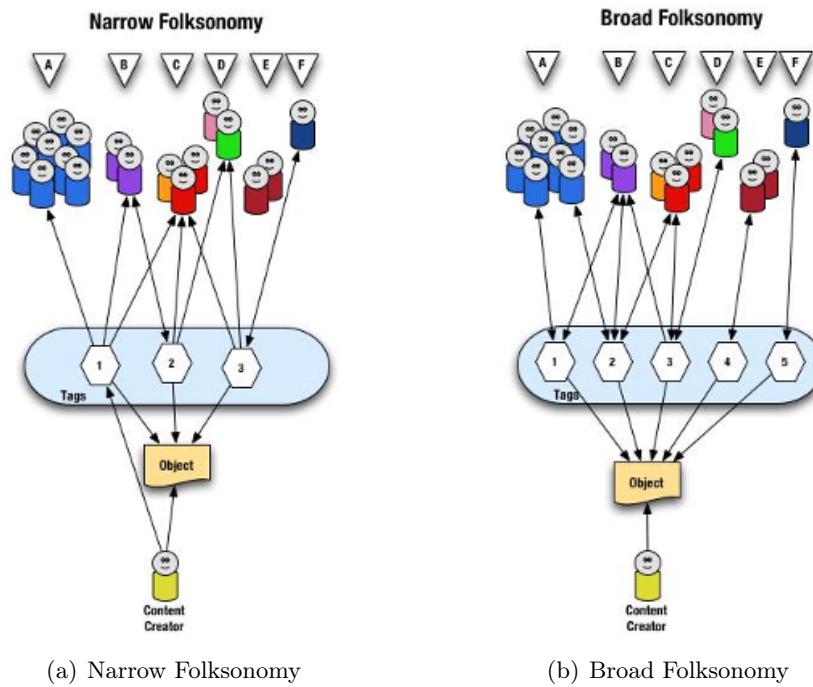


Abb. 3.14: Vergleich Folksonomien

Narrow Folksonomy / Enge Folksonomie

Die enge Folksonomie (3.14(a), Mantler und Gomer 2006) wird heute am häufigsten verwendet, zum Beispiel bei Social Networks wie *Twitter* und *Instagram*. Hier wird ein Objekt von einem Nutzer erstellt und in das Netzwerk gestellt. Er versieht das Objekt mit initialen Tags über welche es von anderen Nutzern mit verschiedenen Vokabularen gefunden werden kann. Diese können nun weitere Tags an das Objekt anhängen und

ermöglichen es nun eventuell weiteren Nutzern dieses Objekt zufinden und zu *betaggen*. Dabei ist das besondere, dass Tags nur einmalig an ein Objekt vergeben werden können. In der nachfolgenden Abbildung wird dies noch einmal verdeutlicht.

Broad Folksonomy / Breite Folksonomie

Anders als bei der engen Folksonomie besteht bei der breiten Folksonomie (3.14(b)) die Möglichkeit, dass die User Tags mehrfach an ein Objekt anhängen. Dadurch besteht die Möglichkeit die Popularität (die *Beliebtheit*) eines Tags auf einem Objekt zu bestimmten und auszuwerten. Ein Beispiel für die breite Folksonomie ist die Website del.icio.us. In der nachstehenden Abbildung wird auch die Breite Folksonomie noch einmal verdeutlicht.

Vorteile und Nachteile von Folksonomien

Es wurden die Vor- und Nachteile gegenübergestellt.

Vorteile	Nachteile
Stöbern / unscharfe Suche wird ermöglicht	Fehlende Kontrolle / Regeln
Authentisch	Spam / Troll-Tags
Masseninformation können gesammelt werden	Datenschutz / Erfassung von Nutzerverhalten
Sensibilisieren Nutzer für Inhaltserschließung	Keine Eindeutigkeit / fehlendes Vokabular

Das Suchen von Objekten und Ressourcen ist kein triviales Phänomen in der Informatik oder in Social Networks. Das Problem dabei ist, dass der Suchende meist nicht ganz exakt weiß wonach er eigentlich sucht. Entweder fehlt dem Suchenden die genaue Bezeichnung oder er hat nur eine weite Richtung in welcher die Ressource vermutet wird. Deshalb gibt es in vielen Systemen Klassifikationen oder ähnliches um den Nutzer bei seiner Suche zu unterstützen. Mittels Folksonomien und dem Social Tagging ist es möglich dem Nutzer eine unscharfe Suche zu ermöglichen. Dies bietet den Vorteil dass der Nutzer sich *stöbernd* durch die Objekte bewegen kann und seine Auswahl immer mehr eingrenzt, bis er nur noch eine überschaubare Menge an Ressourcen übrig hat und in dieser sich sein gesuchtes Objekt befindet.

Dabei wird er auch sensibilisiert auf den Inhalt oder die thematischen Verbindungen der Ressource, da er die Tags liest und mit dem Objekt verbindet. Je mehr User die Tags *betaggen*, desto authentischer wird die Folksonomie. Doch dadurch entstehen auch die Probleme mit Folksonomien denn durch die fehlende Kontrolle und durch fehlende Regeln können schlechte oder irritierende Tags das Ergebnis verfälschen. Zum Beispiel können Tags mit der gleichen Bedeutung in verschiedenen Zeitformen oder Wortbeugungen an die Ressource angehangen werden oder es können Spam- oder auch *Troll-Tags* angehangen werden. Und das natürlich auch in allen erdenklichen Sprachen. Das alles ist auf ein fehlendes Vokabular und die fehlenden Standards zurückzuführen. Möglichkeiten zur

Ausbesserung dieser Nachteile wäre zum Beispiel eine automatische Suchvervollständigung, welche schon vorhandene Tags vorschlägt. Eine weitere Verbesserung wäre die Wortstammfindung, welche in Kapitel ?? näher beschrieben wird.

3.4.4 Tag-Clouds

Die Darstellung der zueinander gehörigen Tags in Form einer Wolke (z.B. Abbildung 3.15) ist vielfältig denkbar. Für die Umsetzung wurde uns zunächst vorgegeben keine Rotation, Farben oder Schriftarten einzusetzen. Stattdessen sollen wir uns auf einen Gewichtungsfaktor der auf die Positionierung Einfluss nimmt konzentrieren.



Abb. 3.15: Tag-Cloud

3.5 Algorithmus Pseudocode

Zur visuellen Anordnung der unterschiedlich großen Tags/Worte in Tag-Clouds, recherchierten die Studenten nach verschiedenen Algorithmen. Ein Algorithmus wurde anhand eines **Pseudocodes** präsentiert. Anhand eines kleinen Beispiels wurde die Arbeitsweise des Algorithmus erläutert.

```

TagCollection tags: User-Tag-Ressource-Tupel aggregiert auf die Taganzahl

Function GenerateFolksomonyTagCloud(
    TagCollection tags,
    FontSize minSize,
    FontSize maxSize,
    Integer minCount)

{foreach(Tag tag in tags){

    tag.Size = (maxSize - minSize) ·  $\frac{tag.Count - minCount}{tags.MaxCount - minCount}$  + minSize
}

return tags;
}

```

Abb. 3.16: Pseudocode

3.6 Fallbeispiel

Auf der Abbildung 3.17 ist folgendes Szenario dargestellt: Drei verschiedene Personen, namens Karl, Max und Sabine geben für drei unterschiedliche Obstsorten Tags ihrer Wahl ein.

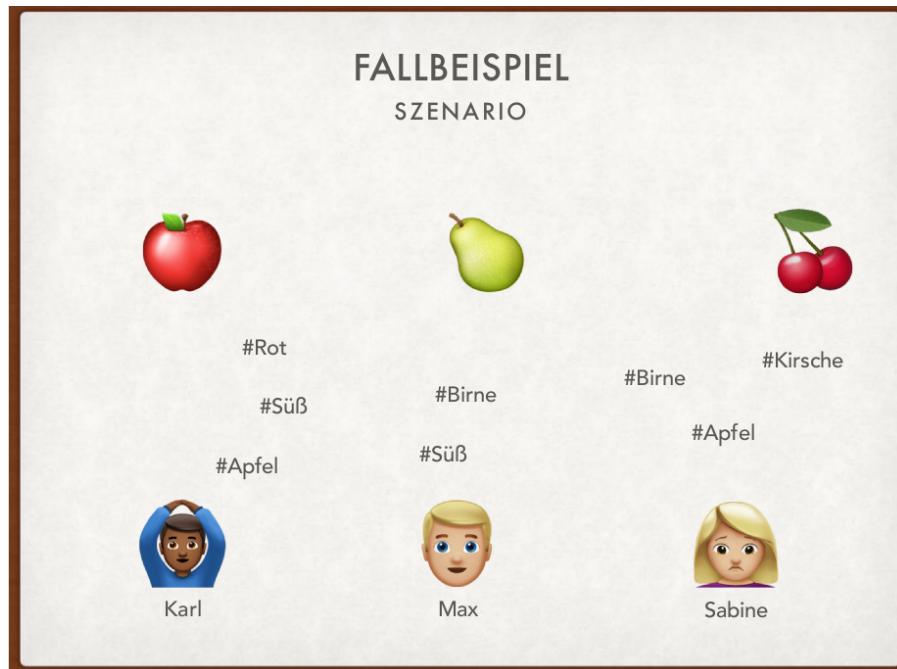


Abb. 3.17: Fallbeispiel Szenario

- Für die Apfelsorte: #Rot, #Süß, #Apfel
- Für die Birnensorte: #Birne, #Süß
- Für die Kirschsorte: #Kirsche, #Apfel

Diese eingegebenen Tags werden in einer Datentabelle (Abbildung 3.18) erfasst.

In einer weiteren Tabelle, werden die eingegebenen Tags sowohl aufgelistet als auch festgehalten, wie oft ein Tag eingegeben wurde. Mit der in Abbildung 3.19 abgebildeten Formel, wird außerdem seine Größe, mit der er letztendlich in der Tag-Cloud abgebildet werden soll bestimmt.

In Abbildung 3.20 wurde der Tag #Süß am häufigsten eingegeben, weshalb er auch größer dargestellt wird.

FALLBEISPIEL		
DATEN		
User	Tag	Ressourcen
Karl Schneider	Apfel	Apfelsorte
Karl Schneider	Rot	Apfelsorte
Karl Schneider	Süß	Apfelsorte
Max Pichler	Birne	Birnensorte
Max Pichler	Süß	Birnensorte
Sabine Müller	Birne	Birnensorte
Sabine Müller	Kirsche	Kirschsorte
Sabine Müller	Apfel	Apfelsorte



Abb. 3.18: Fallbeispiel Szenario

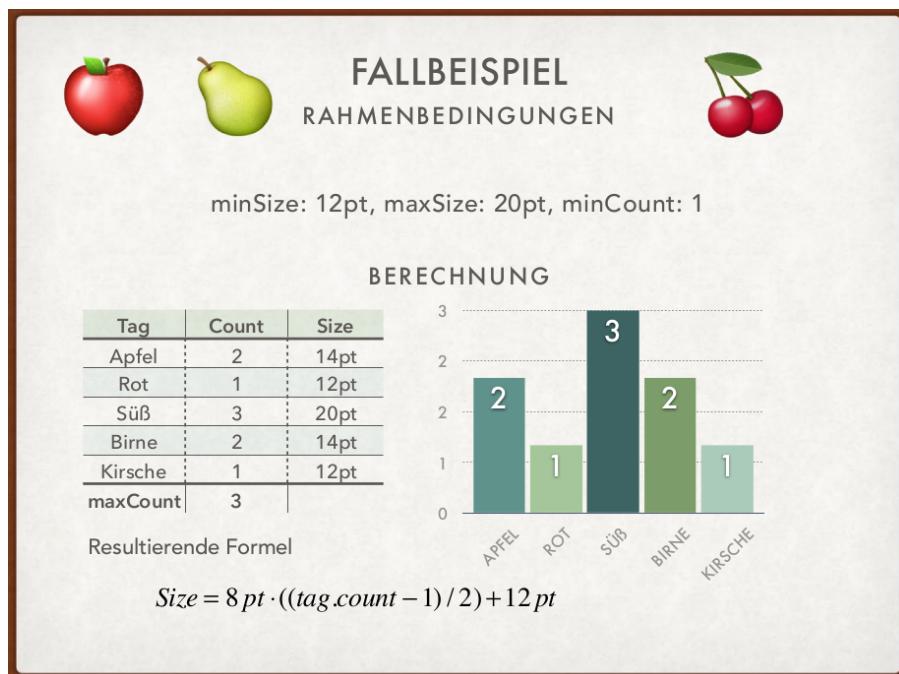


Abb. 3.19: Fallbeispiel Szenario

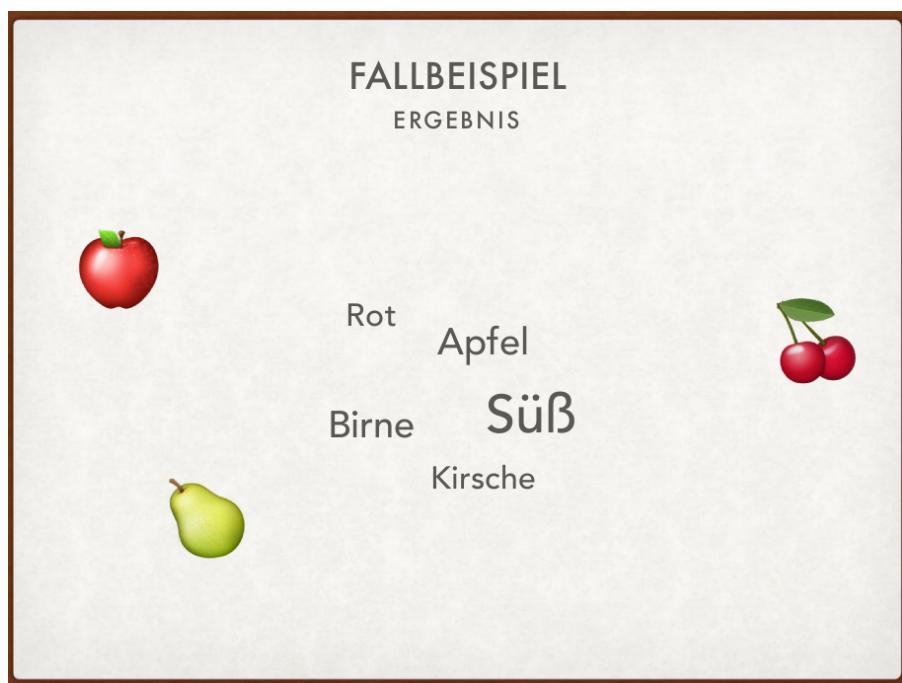


Abb. 3.20: Fallbeispiel Szenario

3.7 Herangehensweisen der Tag-Cloud-Erstellung

3.7.1 Anforderungen / Grundlagen vor der Tag-Cloud-Erstellung

In Vorbereitung auf die Tag-Cloud-Erstellung sollte eine bereits gefilterte Liste mit Angaben zu den jeweiligen Worthäufigkeiten vorliegen. Das Teilprojekt „Wortstammfindung“ sollte als Grundlage diese gefilterte Liste erstellen. Da das Wortstammfindungs-Projekt parallel zur Implementierung startete, mussten die Studenten sich für Testzwecke eine eigene Liste schreiben. Um der Realität einer echten Tag-Cloud näherzukommen, wurde folgender Quellcode zufällige Wortausgaben generiert. Das Tag, beziehungsweise das Wort mit der größten Häufigkeit, ist zuerst zu platzieren. Eine absteigende Sortierung nach Häufigkeit der Tags muss somit zwingend erfolgen. Die virtuelle Erstellung der vorgegebenen Positionierungsfläche für die Tag-Cloud folgt als nächster Schritt. Die anschließenden Schritte unterscheiden sich bei Spiralform-verfahren und „Hierarchical Bounding Boxes“-Verfahren.

3.7.2 Spiralform-Verfahren

Das erste Wort wird aus der Liste entnommen und in ein Rechteck umgewandelt. Für die Berechnung der Rechteckgröße spielen Wortlänge, Häufigkeit und maximale Schriftgröße eine zentrale Rolle. Passt bereits das erste Wort nicht in die Liste wird es bzw. das Rechteck solange verkleinert, bis es in das Fenster passt. Die Position des ersten Wortes ist zwingend im Zentrum des Positionierungsfensters. Jedes weitere Wort wird dann auf einer gedachten Spirale vom Zentrum des Positionierungsfensters aus abgelegt und auf Kollision geprüft. Liegt eine Kollision mit einem anderen gedachten Rechteck vor, so muss das Wort ein Stück weiter auf der gedachten Spirale verschoben werden. Ist die neue Position des Rechtecks nicht belegt, so bekommt das Rechteck diesen Platz zugewiesen. Als nächster Schritt kommt das nächste neue Wort bzw. umgewandelte Rechteck auf den nächsten neuen Punkt der Spirale. Der Positionierungsvorgang startet wieder. Dieser Schritt wird so oft wiederholt, wie Platz im Positionierungsfenster vorhanden oder eine maximale Wortanzahl vorhanden ist. Die Studenten haben sich für dieses Verfahren entschieden, da es einfacher zu programmieren ist als das „Hierarchical-Bounding-Boxes-Verfahren“ und es gab ein Beispiel-Code - der allerdings auf Pixel und Bildern basierte - an dem sie sich orientieren konnten.

3.7.3 Hierarchical-Bounding-Boxes -Verfahren

Das erste Wort ist in diesem Verfahren nicht unbedingt mittig im Zentrum des Positionierungsfensters zu platzieren. Nachdem das erste Wort abgelegt ist, wird das Positionierungsfenster in immer kleiner werdende Rechtecke unterteilt.

Wie in Abb. 3.21 zu sehen, werden Linienzüge (rote Linien in Abbildung) für die Umrandung der Rechtecke als Vektoren berechnet. Im Hintergrund entsteht dadurch eine Baumstruktur. Das nächste Wort wird in das kleinstmögliche passende und vorhandene

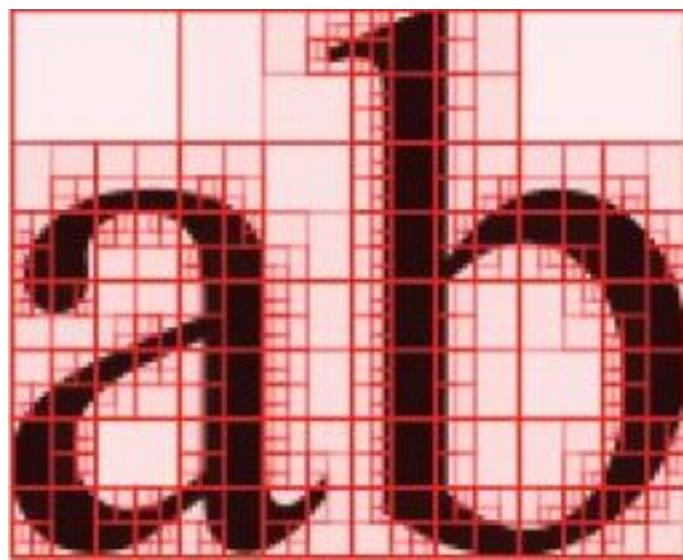


Abb. 3.21: Hierarchical-Bounding-Boxes: Aufteilung der Positionierungsfläche in Rechtecke

Rechteck abgelegt. Die Berechnung der neuen Linienzüge muss erneut vor dem Einfügen des nächsten Wortes stattfinden. Ein neues Wort kann eingefügt werden. Für jedes weitere Wort wiederholen sich die letzten zwei Schritte bis entweder eine maximale Wortanzahl erreicht ist, oder kein Platz mehr im Rechteck vorhanden ist. Die Berechnung der Linienzüge erfordert sehr viel Rechenaufwand und die Laufzeit des Verfahrens steigt quadratisch mit der Wortanzahl an. Aus dem erst genannten Grund fällt dieses Verfahren für die Tag-Cloud-Erstellung aus.

4 Entwurf

Aufgrund des sehr frühen Entwicklungsstandes von der echten Laufzeitumgebung für die WordCloud wurde beschlossen zunächst eine ähnlichen Prototyp zu erstellen mit dem Ziel, die komplexen Abläufe des realen Systems einfacher darzustellen, um ungewollte und für das Projektteam nicht nachvollziehbare Nebeneffekte während der Entwicklungsphase stark zu reduzieren. Anhand des Prototyp wurde die Berechnung von Folksonomien und die Möglichkeiten der Tageingabe erforscht.

Um eine einfache Datenbasis zum Testen zu erstellen wurde die freie Datenbank Terra¹ ausgewählt. Sie enthält geografische Daten wie Städte, Flüsse und Berge mit Relation untereinander, was sich als sehr nützlich erwies, da sich diese einfach in Tags übertragen ließen.

4.1 Aufbau

In diesen Abschnitt wird der Aufbau des Prototyp beschrieben.

4.1.1 Funktionen

In diesem Teilabschnitt werden die implementierten Funktionen kurz erläutert.

Datenbank Die folgenden Funktionen dienen dem Anlegen und Modifizieren der Datenbankdateien zur persistenten Datenhaltung.

Datenbank anlegen oder laden Über den Reiter „Datenbank“ → „Laden“ kann eine leere Datenbank geladen und neu befüllt oder eine vorhandene geladen und bearbeitet werden.

Datenbank leeren Über den Reiter „Datenbank“ → „Leeren“ kann man in einer zuvor geladenen Datenbank alle Datensätze löschen.

¹ Dürr und Radermacher 1990 via <https://www.sachsen.schule/terra2014/index.php>

4.1.2 Darstellung

Die ausführlichste Erläuterung zum Thema Visualisierung von WordClouds war Wordle (Feinberg 2010). Das Prinzip der Gewichtung von Worten und spiralförmigen Darstellung beginnend mit dem höchstgewichteten in der Bildmitte wurde aus den Beispielen entlehnt. Die Funktionsweise ist nachvollziehbar im Vortrag zur Visualisierung von Tag-Clouds (Lemire 2008) dargelegt. Zusätzlich half uns der Code des Projektes WordCloud (Talbot 2011) nicht den Fehler zu begehen eine Positionierung über Bitmaps zu versuchen. Die Performance mit mehreren Sekunden zur Generierung eines einzelnen Bildes ist hier ausreichend abschrecken. Deshalb werden die Elemente unserer Tag-Cloud ohne Bitmap in Rechtecken mit Positionsangabe sowie den Dimensionen Höhe und Breite versehen. Wir haben unsere Worte (Tags) weiterhin mit einem Gewicht versehen definiert. Dieses kann u.a. aus der Häufigkeit oder später einmal aus der Verknüpfungshäufigkeit in Wortnetzen (s.a. Unterabschnitt 7.3.4) ermittelt werden. Für die Testdaten wird dies schlicht inkrementell oder zufällig auf die Tags verteilt. Die Bildmitte kann aus den vorgegebenen Dimensionen des Bildes ermittelt werden. Weitere Parameter der Spirale sollen experimentell in Verbindung mit Tests zur Performance (Abschnitt 6.2) ermittelt werden. Zu vermuten ist hier als Abstand zwischen zwei Linienzügen die Höhe der kleinsten Schriftdarstellung. Wie in den Anforderungen bereits vorgegeben werden in der Zieldarstellung keine farblichen

Daten

Die folgenden Funktionen dienen Import und Export von gespeicherten Daten von bzw. in andere Systeme und deren manuellen Modifikation.

TerraDB XML in SQLite Datenbank konvertieren Im Prototyp ist es möglich die XML-Export-Datei der Terra Datenbank in eine SQLite Datenbank umzuwandeln, damit weitere Transformationen einfacher mit SQL umgesetzt werden können. Dafür wird zunächst die XML Datei ausgewählt und anschließend der Speicherort der SQLite-Datenbank. Danach wird für jeden Tag-Typen eine Tabelle mit allen Attributen erstellt und anschließend die XML Datei eingelesen und die Datensätze entsprechend eingefügt. Fremdschlüssel werden nicht beachtet. Am Ende entsteht eine SQLite Datenbank die das gleiche Schema wie die XML Datei ausweist mit dem Vorteil, dass dieses mit SQL abgefragt werden kann.

TerraDB SQLite Datenbank importieren Die bei Abschnitt 4.1.2 erstellte SQLite Datenbank muss jetzt wieder geladen werden und mit Hilfe von SQL Abfragen werden die BusinessObjects und Tags erstellt. Dabei wurde statische Tags, d.h. diejenigen die bei allen BusinessObject des ElementTyps gleich sind, und Dynamische Tags, d.h. diejenigen die aus Attributen oder Verknüpfungstabellen des Elements gewonnen wurden und für jedes Business Object unterschiedlich sein können.

Element		
Typ		Statistische Tags Dynamische Tags
Berg	berg	Name, Land, Landteil, Gebirge
Ebene	eben	Name, Land, Landteil
Fluss	fluss	Name, Land, Landteil, MündetInElementName
Insel	insel	Name, Land, Landteil, Inselgruppe
Land	land	Name, Land, Landteil
Landteil	landteil	Name, Land, Landteil
Meer	meer	Name, Land, Landteil
See	see	Name, Land, Landteil
Stadt	stadt	Name, Land, Landteil, LiegtAnElementTyp, LiegtAnElementName
Wüste	wueste	Name, Land, Landteil, Wüstenart

BusinessObject hinzufügen Um ein neues BusinessObject hinzuzufügen, kann über die Menüleiste unter dem Reiter „Daten“ → „Neuer Datensatz“ ein neues BusinessObject erstellt werden. Hierfür müssen nur alle benötigten Daten des hinzuzufügenden Objekts in das Formular eingetragen werden. Das neue Objekt wird nach dem Bestätigen direkt in der Datenbank gespeichert.

Datenbank als Cypher Graph Definition exportieren Im Prototyp existiert ein Service, welcher Code für das Anlegen einer Neo4J² Datenbank erstellt. Dieser Code erstellt die benötigten Datensätze über BusinessObjects und Tags sowie deren Beziehung zueinander.

BusinessObject anzeigen und editieren Um ein konkretes BusinessObject anzuzeigen, muss lediglich auf dieses geklickt werden. Es öffnet sich ein Fenster welches alle Daten zum angeklickten Objekt in einem Formular anzeigt, worin diese auch direkt geändert werden können.

WordCloud

Die folgenden Funktionen dienen zur Modifikation des Filters bezogen auf die Tags zum Filter über die WordCloud.

Hinzufügen Um ein Tag zum Filter hinzuzufügen, ist lediglich ein Klick auf das gewünschte Tag in der WordCloud nötig. Dieses wird dann sofort zum Filter hinzugefügt. Sofort darauffolgend aktualisiert sich auch die WordCloud und die Anzeige der gefundenen BusinessObjects, da diese direkt von dem Filter abhängen.

Entfernen Einen Tag vom Filter zu entfernen ist ähnlich simpel, wie einen Tag zum Filter hinzuzufügen. Zum Entfernen ist ebenfalls nur ein Klick auf das zu entfernende Tag

² <https://neo4j.com/>

im Filter nötig. Darauffolgend wird, analog zum Tag hinzufügen, die WordCloud sowie die Anzeige der BusinessObjects aktualisiert.

4.1.3 Komponenten

In diesem Teilabschnitt werden die Komponenten zur Funktionsbereitstellung technisch beschrieben und deren Modellierung erläutert. Es wurde darauf geachtet die Funktionalität, die Daten und die Benutzer

Model

Das Modell für die Datenhaltung ([Abbildung 4.1](#)) ist dem des realen Systems nachempfunden, beschränkt aber sich auf die wesentlichen Elemente zur Generierung einer Folksonomie. Zur Visualisierung für die Präsentation wurde zusätzlich noch ein Bild hinzugefügt.

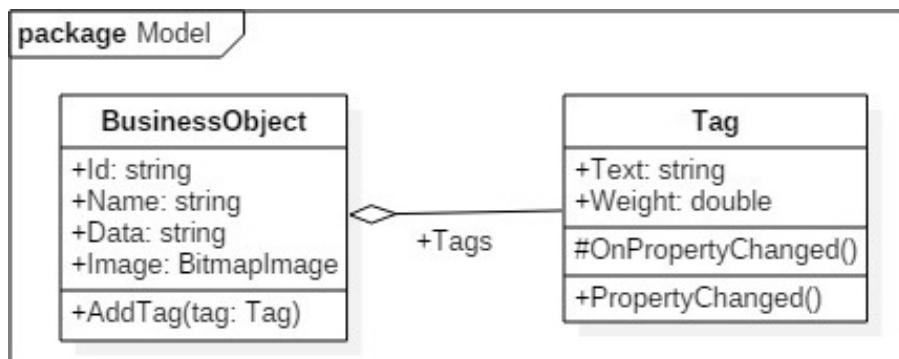


Abb. 4.1: Klassendiagramm des Datenmodells

Das Bild wird base64-codiert als Text in dem Attribut `data` des Business Objects gespeichert.

Die [Abbildung 4.2](#) verdeutlicht die Beziehungen in der Datenbank.

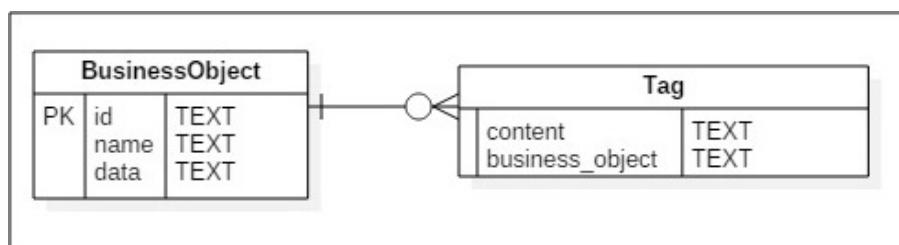


Abb. 4.2: Entity Relationship Diagram

DataAccess

Die Klasse `SqliteDatabaseConnection` abstrahiert die systemeigene Schnittstelle und erstellt Abfragen verschiedener Rückgabegranularität (Nur Ausführen, Skalarer Wert, Alle Werte). Außerdem wird die Funktionalität um weitere Hilfsmethoden wie Tabellenexistenz

prüfen erweitert. Die Klasse Repository stellt die Datenobjektzugriffsmethoden ([Abbildung 4.3](#)) bereit und bietet so einen abstrakten und objektorientierten Zugriff auf die Persistenzschicht.

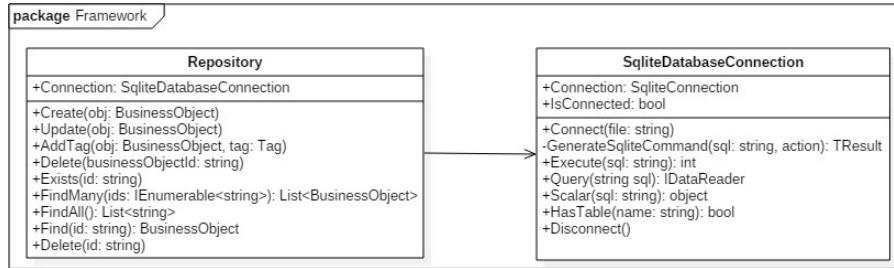


Abb. 4.3: Klassendiagramm von Datenbankzugriff

Service

In dieser Komponente ([Abbildung 4.4](#)) ist die gesamte Logik implementiert.

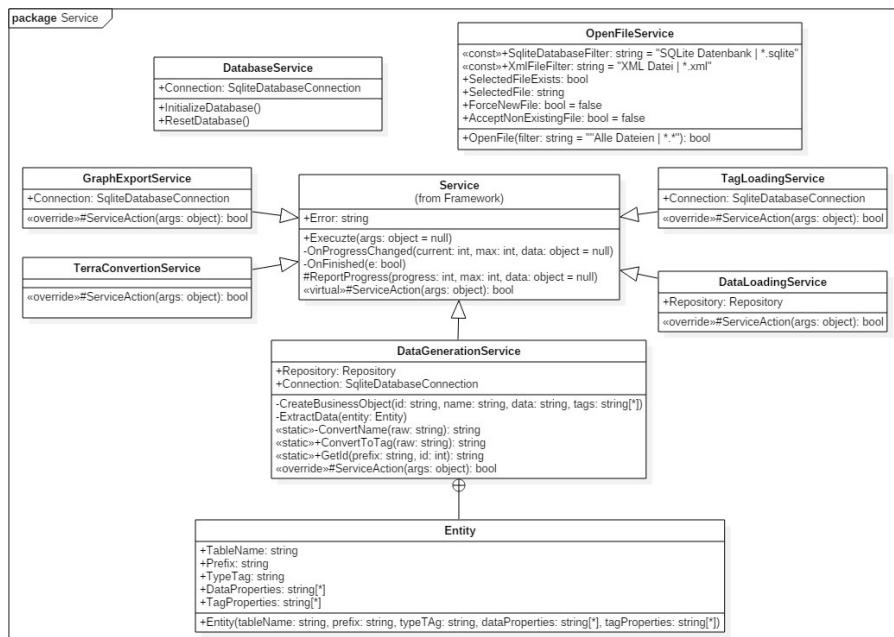


Abb. 4.4: Klassendiagramm von Services

DatabaseService initialisiert und leert die Datenbank.

OpenFileService sorgt für eine geordnete Auswahl einer Datei, indem er Eigenschaften anbietet, die die Dateien an Endungen filtert und prüft ob die Datei wirklich existiert und so entweder nichtexistierende Dateien verbietet, erlaubt oder erfordert.

TerraConversionService lädt die Terra Xml Datei und wandelt diese in eine SQLite Datenbank um.

DataGenerationService lädt diese Datenbank wieder und wandelt deren Elemente und Beziehungen in BusinessObjects und Tags um und speichert diese in der Persistenzschicht. Die im Service enthaltene Entity Klasse enthält dabei die Eigenschaften eines Elementtyps, um die Daten aus der Datenbank zu extrahieren und zu transformieren.

GraphExportService lädt alle Daten der Persistenzschicht und wandelt diese in Cypher kommandos um, die ein Neo4j Graphen Datenbankschema erzeugen. Das Ergebnis wird in eine zuvor ausgewählte Textdatei gepeichert. Dabei stellen die BusinessObjects und die Tags jeweils Knoten dar und deren Beziehungen die Verbindungen. Damit kann sich ein Überblick über das Netz in der Datenbank verschafft werden.

TagLoadingService generiert und führt die SQL Abfrage für das Laden der in Beziehung stehenden Tags anhand der Suchanfrage aus, welche als Parameter übergeben wird. Dabei werden auch deren Gewichte anhand der Häufigkeit des Vorkommens an den im Suchergebnis gelisteten BusinessObjects berechnet.

DataLoadingService generiert und führt die SQL Anfrage aus, welche anhand der Suchanfrage die entsprechenden BusinessObjects aus der Persistenzschicht lädt.

Klassen

TerraDbWordCloudAppearanceArguments (WordCloud Klasse) definiert die für die Word-Clouddarstellung benötigten Parameter.

Logger dient zu Demonstrationszwecken und bieten dem Nutzer einen Einblick, wie die Berechnungsalgorithmen für die Folksonomien funktionieren.

MVVM

Das Model-View-ViewModel (MVVM) Pattern ist zentraler Bestandteil der Windows Presentaion Foundation (WPF), dem aktuellen Applicationframework für die Windowsprogrammierung. Es dient der strikten Trennung der Oberfläche (View) von der Logik (ViewModel) und den Daten (Model).

ViewModel enthalten, wie bereits erwähnt, die Logik der Benutzeroberfläche der Applikation. Das heißt sie stellt Daten zur Darstellung und Kommandos zur Interaktion bereit. Das **MainViewModel** ist für das Hauptfenster, das **BusinessObjectEditorViewModel** für das Popup, das angezeigt wird wenn ein gefundenes **BusinessObject** per Klick geöffnet wird und das **TagCollectionViewModel** für das Popup, das angezeigt wird, wenn Tags eines **BusinessObject** per **OldSchool** Methode bearbeitet werden zuständig. Diese wurden an die Jeweiligen Views als **DataContext** gebunden.

RelayCommand ist eine Helperklasse, die das ICommand Interface implementiert und die Methoden CanExecute und Execute als Delegat-Typen bereitstellt, sodass diese Logik direkt im ViewModel implementiert werden kann ohne dass eine Extra Klasse

für Kommandos angelegt werden muss und gegebenenfalls Kontexte umständlich übergeben werden müssen.

Converter

Converter dienen als Übersetzer der Daten in ein angemessenes Format für die View.

Sie werden an eine Datenbindung angehangen über konvertieren die Daten bei jeder Änderung. So kann diese Konvertierungslogik aus dem ViewModel entfernt werden und ist wiederverwendbar

`BooleanToVisibilityContainer` überträgt einen Wahrheitswert in die `VisibilityEnumeration`.

`TagCollectionToStringConverter` serialisiert die TagCollection in eine Zeichenkette und parst diese nach Änderung wieder in eine TagCollection zurück. Somit ist eine sehr einfache Tageingabe möglich, da dieser an jede beliebige Texteingabe gebunden werden kann.

4.1.4 Benutzeroberfläche

Die Benutzeroberfläche ist wie für einen Prototyp ([Abbildung 4.5](#)) typisch schlicht gehalten. Zur besseren Benutzbarkeit sind die Grenzen rechts und unterhalb der Gefundenen BusinessObjects frei verschiebbar.

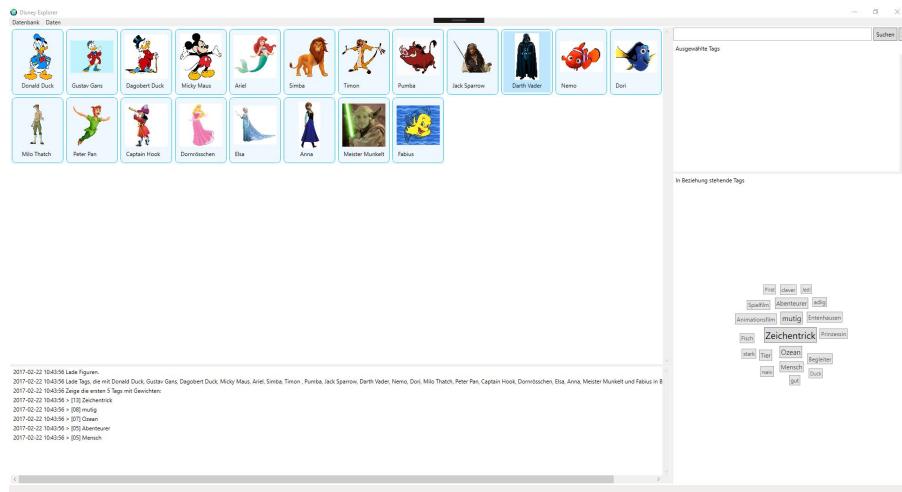


Abb. 4.5: Initialzustand des Prototyps (Datenbank geladen)

4.2 Konstruktion eines nachvollziehbaren Beispiels zur Präsentation

Da geographische Daten für die Präsentation für das Publikum einer Präsentation eher uninteressant und nicht unbedingt nachvollziehbar sind wurde beschlossen, unser erworbenes Verständnis einer guten Folksonomie zu anzuwenden und ein eingängigeres Beispiel zu erstellen, sodass die Zuhörer das Interesse während der Präsentation nicht verliert. Dabei wurde sich auf den Themenbereich Disney geeinigt, da davon ausgegangen wurde, dass jeder zumindest einen Teil dieser Figuren kennt und so die Beziehungen deutlicher und somit verständlicher werden. Durch die emotionale Wirkung der Marke Disney ist auch mit anhaltender Aufmerksamkeit zu rechnen.

Beispielhafter Filter-/Suchprozess

Um ein Verständnis für den Ablauf einer Such- bzw. Filteraktion durch den Nutzer und den Zusammenhang zwischen WordCloud, Ergebnisliste und Datenbankabfrage zu erlangen, wird der Prozess exemplarisch anhand der selbst erstellten Datenbank beschrieben.

4.2.1 Ausgangslage

Ein Benutzer sucht nach einem Objekt, was er nicht genau kennt. Ihm sind lediglich einige Eigenschaften bekannt. Die zugehörige Datenbank wurde bereits geladen.

Tags	Auswahl (mit Ziel „Munkelt“)
1	stark
2	mutig
3	clever

4.2.2 Schritt 1: Benutzer klickt auf das Tag „stark“

Dadurch wird im Hintergrund das SQL Query gem. [Abbildung 4.6](#) generiert und ausgeführt.

```
SELECT
    BusinessObject.id, BusinessObject.name
FROM
    BusinessObject
JOIN Tag AS tag1
    ON BusinessObject.id = tag1.business_object
    AND tag1.content = 'stark';
```

Abb. 4.6: 1. Abfrage

Pro angeklickten Tag wird jeweils ein zusätzlicher JOIN über die Tabelle der Businessobjekte und der Tags durchgeführt um die Filterung anhand des Tags zu ermöglichen.

Es werden nun alle Objekte mit dem entsprechenden Tag „stark“ herausgefiltert und angezeigt ([Abbildung 4.7](#)).



Abb. 4.7: Filterergebnis Tag „stark“

Als Resultat wird ebenfalls die WordCloud entsprechend des neuen Filterergebnis neu erstellt. Im Vergleich zur Ausgangs-WordCloud wird sie kleiner, da die Filtermenge eingeschränkt wurde ([Abbildung 4.8](#)).



Abb. 4.8: WordCloud nach 1. Filterung

4.2.3 Schritt 2: Benutzer klickt auf das Tag „mutig“

Dadurch wird im Hintergrund das SQL Query gem. [Abbildung 4.9](#) ausgeführt.

```
SELECT
    BusinessObject.id, BusinessObject.name
FROM
    BusinessObject
JOIN Tag AS tag1
    ON BusinessObject.id = tag1.business_object
    AND tag1.content = 'stark'
JOIN Tag AS tag2
    ON BusinessObject.id = tag2.business_object
    AND tag2.content = 'mutig';
```

Abb. 4.9: 2. Abfrage

Es werden nun alle Objekte mit den entsprechenden Tags „stark“ und „mutig“ herausgefiltert und angezeigt ([Abbildung 4.10](#)).



Abb. 4.10: Filterergebnis Tags „stark“ & „mutig“

Als Resultat wird die WordCloud analog zum Schritt 1 entsprechend des neuen Filterergebnis neu erstellt ([Abbildung 4.11](#)).



Abb. 4.11: WordCloud nach 2. Filterung

4.2.4 Schritt 3: Benutzer klickt auf das Tag „clever“

Dadurch wird im Hintergrund das SQL Query gem [Abbildung 4.12](#) ausgeführt.

```

SELECT
    BusinessObject.id, BusinessObject.name
FROM
    BusinessObject
JOIN Tag AS tag1
    ON BusinessObject.id = tag1.business_object
    AND tag1.content = 'stark'
JOIN Tag AS tag2
    ON BusinessObject.id = tag2.business_object
    AND tag2.content = 'mutig'
JOIN Tag AS tag3
    ON BusinessObject.id = tag3.business_object
    AND tag3.content = 'clever';
    
```

Abb. 4.12: 3. Abfrage

Es werden nun alle Objekte mit den entsprechenden Tags „stark“, „mutig“ und „clever“ herausgefiltert und angezeigt. Die Ergebnisse in den resultierenden Objekten bleiben in diesem Fall allerdings im Vergleich zu Schritt zwei gleich.

Die WordCloud wird analog zum Schritt 1 entsprechend des neuen Filterergebnis neu erstellt ([Abbildung 4.13](#)).



Abb. 4.13: WordCloud nach 3. Filterung

4.2.5 Schritt 4: Benutzer sucht nach dem Begriff „Munkelt“.

Dadurch wird im Hintergrund das SQL Query gem. [Abbildung 4.14](#) ausgeführt.

```

SELECT
    BusinessObject.id, BusinessObject.name
FROM
    BusinessObject
JOIN Tag AS tag1
    ON BusinessObject.id = tag1.business_object
    AND tag1.content = 'stark'
JOIN Tag AS tag2
    ON BusinessObject.id = tag2.business_object
    AND tag2.content = 'mutig'
JOIN Tag AS tag3
    ON BusinessObject.id = tag3.business_object
    AND tag3.content = 'clever'
WHERE name LIKE '%Munkelt%';
  
```

Abb. 4.14: 4. Abfrage

Durch eine zusätzliche WHERE-Klausel werden nun alle Objekte mit den entsprechenden Tags „stark“, „mutig“ & „clever“ sowie den Namen, welche „Munkelt“ enthalten, herausgefiltert und angezeigt ([Abbildung 4.15](#)).

Als Resultat wird die WordCloud analog zum Schritt 1 entsprechend des neuen Filterergebnis neu erstellt ([Abbildung 4.16](#)).



Abb. 4.15: Einzelnes Objekt nach 4. Filterung

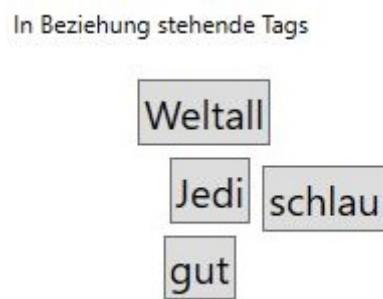


Abb. 4.16: WordCloud nach 4. Filterung

4.2.6 Ergebnis

Die Auswahl führte im 4. Schritt mit [Abbildung 4.16](#) zu einem eindeutigen Treffer. Eine weitere Selektion von Tags kann die Genauigkeit nicht mehr erhöhen.

5 Implementierung

5.1 WordCloud-Komponente

5.1.1 Beschreibung

Ein Ziel des Projektes war es, die berechneten Folksonomien mit Hilfe einer WordCloud zu visualisieren. Dieser Mechanismus sollte anschließend in das bestehende PPSn Softwaresystem der Firma Tecware integriert werden. Um eine hohe Flexibilität und Wiederverwendbarkeit zu erreichen wurde die Berechnung und Visualierung der WordCloud in einem Paket gekapselt.

5.1.2 Aufbau

Contract

In diesem Teil wird die Schnittstelle für die Verwendung in anderen Projekten definiert. Das `IWord`-Interface stellt die kleinste Einheit der WordCloud dar. Die implementierende Klasse muss eine Text-Property anbieten die das anzuseigende Wort als String beinhaltet. Das `IWeightedWord`-Interface erweitert `IWord` um einen beliebig Gewichtswert. Es stellt eine Schnittstelle für die Eingabedaten dar. Die `VisualizedWord`-Klasse ([Abbildung 5.1](#)) implementiert `IWord` und stellt das Ergebnis, welches aus der Berechnung der Visualierung entsteht, dar. Es enthält alle wichtigen Parameter, die für eine Darstellung in dem Panel wichtig sind. Die `Range` Klasse stellt einen Zahlenbereich dar und kann relative Werte zu einem anderen Zahlenbereich berechnen.

WordCloudCalculator

Der `WordCloudCalculator` wandelt die übergebenen, gewichteten Wörter des Typs `IWeightedWord` ([Abbildung 5.1](#)) in anzeigbare Wörter des Typs `VisualizedWord` anhand von Rahmenbedingungen, die durch die Bereiche der Freiheitsgrade (Schriftgröße und Transparenz), der Größe des Anzeigefensters und einer Methode zur Berechnung der Anzeigegröße eines Wortes, dargestellt werden. Die Schnittstelle für die Rahmenbedingungen ist in `IWordCloudAppearanceArguments` festgelegt. Die Schnittstelle für die Implementierung eines WordCloudCalculators ist in `IWordCloudCalculator` festgelegt.

Eine Möglichkeit wurde in `ExtractingWordCloudCalculator` implementiert. Dieser besitzt eine weitere Methode vom Typ `IWordAppearanceCalculationMethod`, die ein gewichtetes Wort als Eingabe bekommt und dieses in eine visualisierte Wort umwandelt.

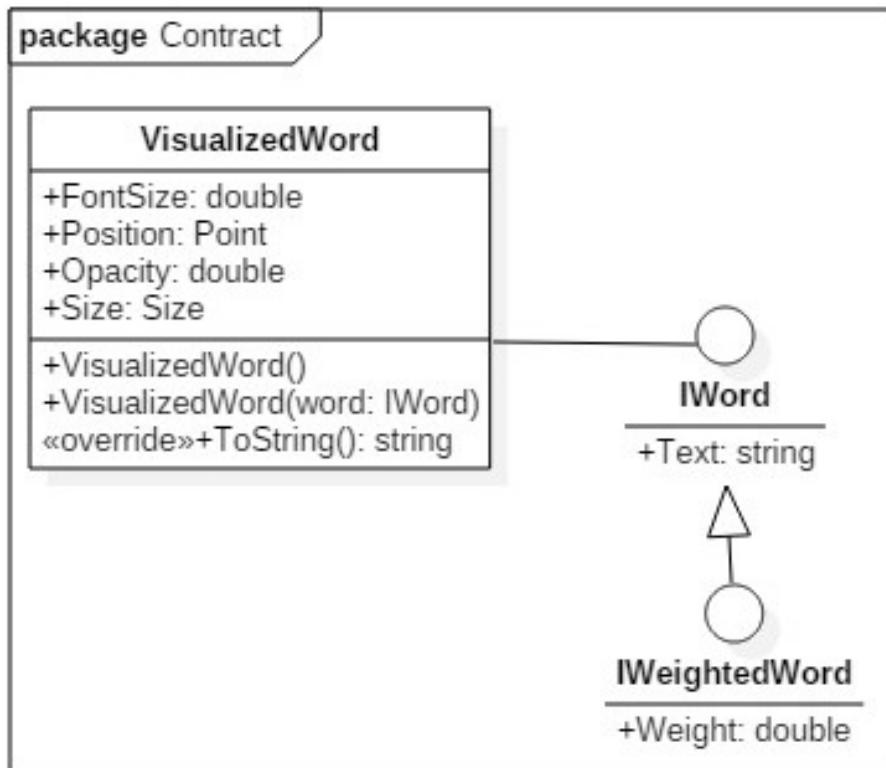


Abb. 5.1: Klassen im Contract der WordCloud

Der Calculator an sich sortiert die Gewichte der übergebenen gewichteten Wörter zuerst absteigend und übergibt dann jedes einzelne Wort solange der Umwandlungsmethode bis diese signalisiert, dass kein Wort mehr auf den anzeigenbaren Bereich passt. Zum Schluss werden die anzeigenbaren visualisierten Wörter zurückgegeben.

Mit Hilfe dieser Klasse war es möglich verschiedene Methoden der Anordnung auszuprobieren. Die `SimpleAppearenceCalculationMethod` ordnet die ersten 5 Wörter diagonal untereinander an und dient als Einstieg für die Erforschung neuer Methoden. Die `SpiralAppearenceCalculationMethod` ordnet die Wörter in der bereits beschriebenen Spiralform an.

WordCloudControl

Das WordCloudControl hat die Aufgabe, die von dem WordCloudCalculator berechneten Wortvisualisierungen in einem Benutzersteuerelement auf dem Bildschirm darzustellen. Die `WordCloud`-Klasse ist dabei das zentrale Element der Komponente. Sie wird im Layout der Software an der gewünschten Stelle platziert. Die Klasse enthält Definitionen für die Übergabe der gewichteten Wortliste, die Rahmenbedingungen für die Visualisierung, eines Kommandos, das Ausgeführt wird wenn ein Wort ausgewählt wird und den Positionierungsmethodentyp für den ExtractingWordCloudCalculator. In der `WordCloudGui.xaml` befindet sich die Viewdefinition der WordCloud. Die WordCloud besteht aus einem ItemsControl, welches die Möglichkeit bietet eine Liste mit Datenobjekten zu binden

und anzuzeigen. Dabei wurde ein sogenanntes MultiBinding eingesetzt, um auch bei Veränderung der Größe der Anzeigefläche die WordCloud neu zu berechnen. Durch das Überschreiben des ItemTemplates wurde die visuelle Darstellung eines Wortes als Button-Steuerelement definiert, woran die berechneten Freiheitsgradparameter und das Klickkommando angebunden wurden. Des Weiteren wurde das ItemsPanel mit dem neu erstellten WordCloudPanel überschrieben, welches die Größe und Position der einzelnen Wörter auf der Anzeigefläche festlegt.

Zur Unterstützung wurden die Helferkasse `CommandProxy` implementiert, welche die Übergabe einer Kommandoinstanz von den Datenkontext des Muttercontrols an den Datenkontext eines Kindcontrols ermöglicht.

Des Weiteren wurde in der `WordSizeCalculatorFactory` eine statische Methode entwickelt, welche anhand von Parametern (Schriftart, Rahmenbreite, Außenabstand, Innenabstand) eine Methode erzeugt, die anhand eines Strings und der Schriftgröße dessen benötigte Anzeigefläche berechnet. Diese Helferkasse ist für die Definition der Darstellungsrahmenbedingungen gedacht.

5.1.3 Verteilung

Nuget

Da diese Komponente wie bereits erwähnt wiederverwendbar ist, wurde für den für C# entwickelten Paketmanager ein Paket erstellt und auf nuget¹ veröffentlicht.

Verwendung

Um diese Komponente in einem neuen Projekt zu verwenden, musst das Paket per nuget hinzugefügt werden. Um das mit der Package Manager Console zu erledigen muss folgendes eingegeben werden:

```
Install-Package WordCloudCalculator
```

Nachdem das Paket erfolgreich installiert ist, folgt die Implementation des Interfaces `IWordCloudAppearenceArguments`, um die Rahmenbedingungen für die Darstellung festzulegen und als Statische Ressource instanziert werden.

```
<Window.Resources>
    <wordCloud:SomeWordCloudAppearenceArguments
        x:Key="AppearenceArguments" />
</Window.Resources>
```

Außerdem muss in der `App.xaml` folgendes RessourceDictionary zu `Application.Resources` hinzugefügt werden, um die Styles der WordCloud der neuen Applikation bekannt zu machen:

¹ ermöglicht einfaches Einbinden in Visual Studio von <https://www.nuget.org/packages/WordCloudCalculator>

```
<ResourceDictionary
    Source="/WordCloudCalculator;component/WPF/WordCloudGui.xaml"/>
```

Nun kann das WordCloudControl in das Layout eingefügt werden

```
<wpf:WordCloud
    AppearanceArguments="{StaticResource AppearanceArguments}"
    Words="{Binding Tags}"
    WordAppearenceCalculationMethodType=
        "extractingWordCloudCalculator:SpiralAppearenceCalculationMethod"
    WordSelectedCommand="{Binding SelectTag}"
    />
```

5.2 Implementierung im PPSn System

5.2.1 Beschreibung

Das PPSn System der Firma TechWare ist eine Client-Server ERP-Lösung für kleine und mittelständige Unternehmen. Es ist eine OpenSource Neuentwicklung der schon bestehenden ERP Lösung.

Im Hintergrund arbeitet eine Lua Interpreter Engine, mit der sich, alternativ zu C#, das System erweitern lässt.

Eine Neuerung im System ist die Nutzung einer Tag-Cloud ([Abbildung 5.2](#), unten rechts) zur unscharfen Suche in der Datenbank.

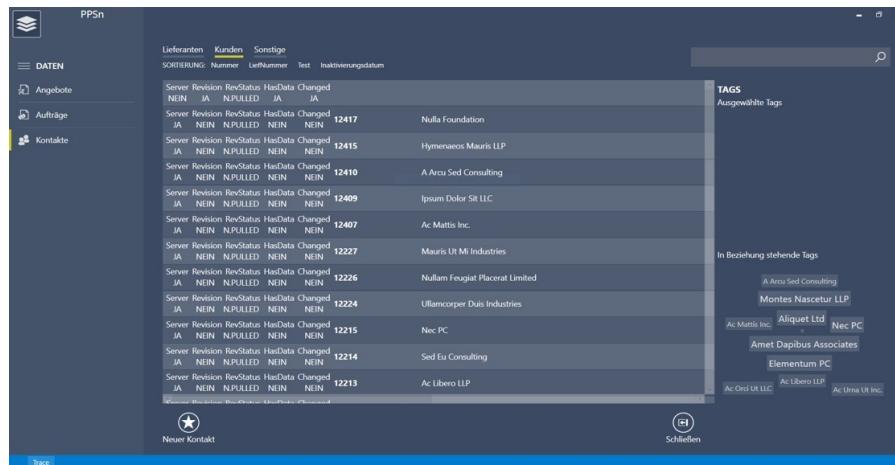


Abb. 5.2: PPSn Softwaresystem

5.2.2 Layout

Die Benutzeroberfläche ([Abbildung 5.3](#), schematisch) besteht aus einer Objektliste, in der alle Datensätze der Datenbank angezeigt werden. Diese können mit Filtern, welche

um dieser List platziert sind verkleinert werden. Anhand dieser Liste wird die Tag-Cloud berechnet. Durch die Auswahl eines Tags wird eine neue Filterbedingung hinzugefügt und sodass die Objektliste aktualisiert wird und so die Tag-Cloud neu berechnet. Durch diesen iterativen Prozess ist eine Suche anhand von Schlagworten möglich.

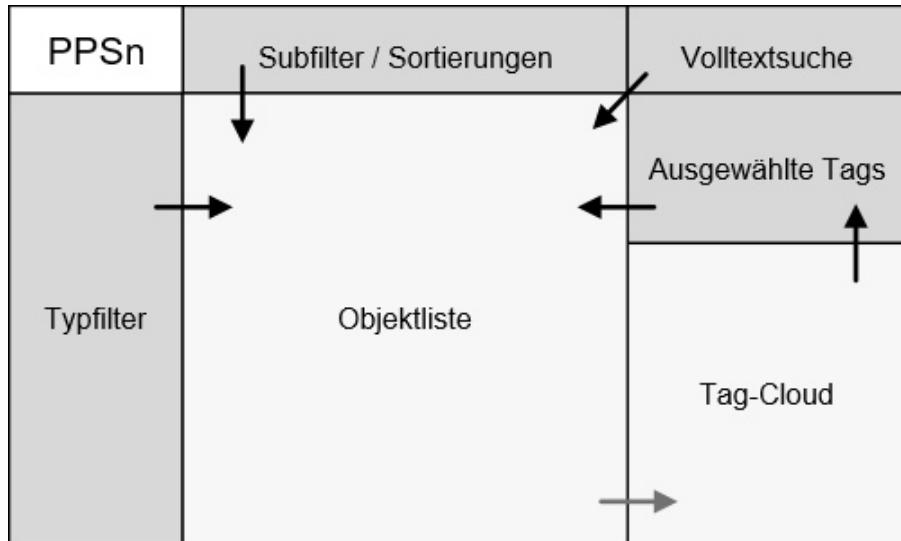


Abb. 5.3: Layout von PPSn: Beeinflussungsübersicht. Dunkelgrau: Filterelemente; Hellgrau: Datenpräsentation

5.2.3 Integration der WordCloud Komponente

Um die WordCloud mit den Tags zu befüllen musste das PPSn System erweitert werden.

Aufgrund der sehr komplexen Systemarchitektur, wird im Folgenden nur auf die Teile des Systems eingegangen in denen Änderungen vorgenommen wurden.

Komponenten

Das `PPsEnvironment` (Abbildung 5.4) enthält den `PpsObjectGenerator`, welcher in der Lage ist SQL zu erzeugen und das Ergebnis als `IEnumerable` bereitzustellen, sodass es mit LINQ weiter verarbeitet werden kann. Da allerdings der bestehende Generator nicht in der Lage war Tags abzufragen und deren Gewichte zu berechnen, musste ein weiterer entwickelt werden. Um die Basisfunktionalität aber beizubehalten, wurde dieser zu `AbstractPpsObjectGenerator` abstrahiert und ohne Änderung wieder von `PpsObjectGenerator` abgeleitet um das Interface zu abhängigen Komponenten nicht zu ändern. Die `AbstractPpsObjectGenerator` Klasse wurde um zwei virtuelle Methoden erweitert. `ExtendCommand` dient dazu, dass abgeleitete Klassen das Kommando über die Factory erweitern können. Durch Implementierung von `GenerateCommandSql` hingegen muss das gesamte Kommando als String neu implementiert werden.

Für die Tag-Cloud wurde dann der neue `WordCloudObjectGenerator` eingeführt, welcher die in Relastion stehenden Tags und deren Gewichte wie im Prototypen berechnet

und abfragt.

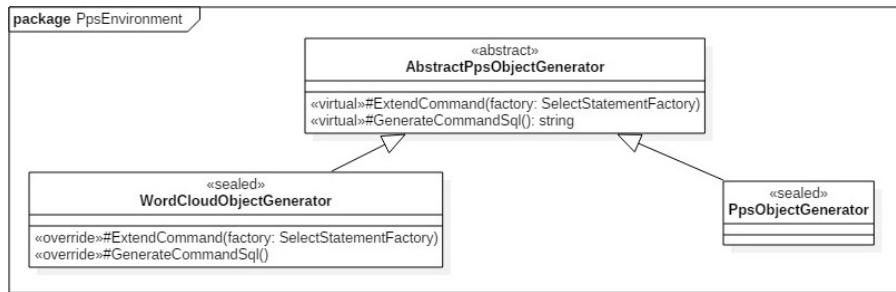


Abb. 5.4: PpsObjectGenerator

Um das generieren des SQL Strings zu vereinfachen und ein nachträgliches Erweitern zu ermöglichen wurde die **SelectStatementFactory** ([Abbildung 5.5](#)) entwickelt. Sie enthält Methoden, die die einzelnen Teile einer Select Abfrage unabhängig von einander aufnimmt und zum Schluss zusammenführt.

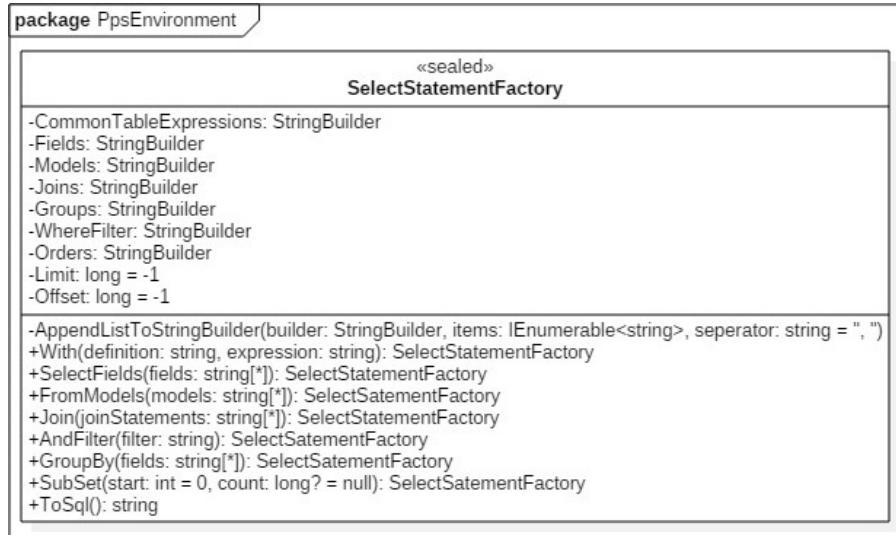


Abb. 5.5: SelectStatementFactory

Das ViewModel ist das **PpsNavigatorModel** ([Abbildung 5.7](#)). Es sind die Komponenten gem. [Abbildung 5.6](#) hinzugekommen.

Name der Komponente	Funktion
Tags	Beinhaltet alle in Relation stehenden Tags
SelectedTags	Beinhaltet alle Tags nach denen gefiltert werden soll
SelectTagCommand	Wird ausgeführt, wenn in der Tag-Cloud auf ein Tag geklickt wird. Fügt dieses der SelectTags Liste hinzu.
UnselectTagCommand	Wird ausgeführt, wenn auf ein Tag in der Liste der ausgewählten Tags geklickt wird. Entfernt dieses aus der SelectedTags Liste
TagFilter	Generiert ein ExpressionObjekt, welches vom System in einen logischen Ausdruck umgewandelt wird und so zur Filterung der Objektliste beiträgt

Abb. 5.6: Hinzugefügte Komponenten

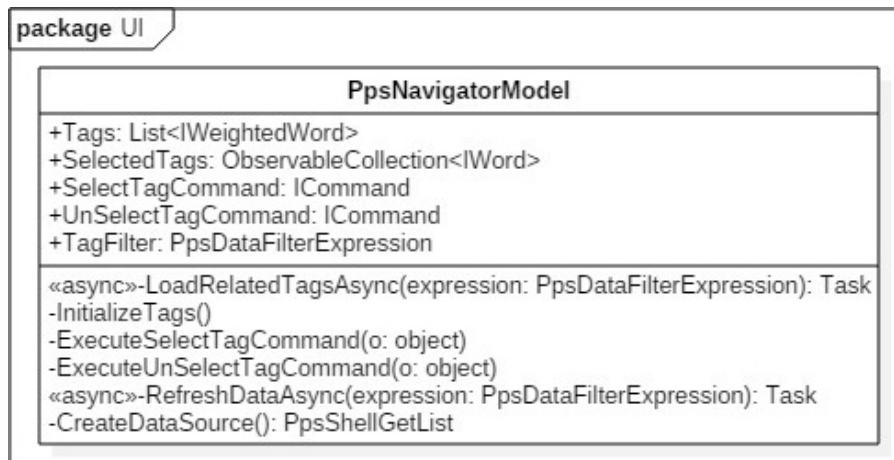


Abb. 5.7: PpsNavigatorModel

Abläufe

Im Diagramm ([Abbildung 5.8](#)) wird beschrieben wie die Daten für die WordCloud geladen werden.

Durch den Aufruf der zentralen Datenladefunktion `RefreshDataAsync` werden die Daten in die Objektliste geladen. Dabei wird der Tagfilter anhand der ausgewählten Tags generiert (3) und mit den bereits existierenden Filtern mit `und` verknüpft. Die daraus entstehenden `DataSource` wird genutzt, um die in Relation stehenden Tags zu laden. Dafür wird im `PpsEnvironment` die `CreateTagCloudFilter` Methode gerufen, welche ein `WordCloudObjectGenerator`-Objekt erstellt und das daraus resultierende `IEnumerable` zurückgibt (5-8). Anschließend werden mit LINQ die benötigten Tagdaten aus dem `IEnumerable` Ergebnis extrahiert und der `Tags`-Property zugewiesen (9). Dadurch wird die WordCloud neu erstellt.

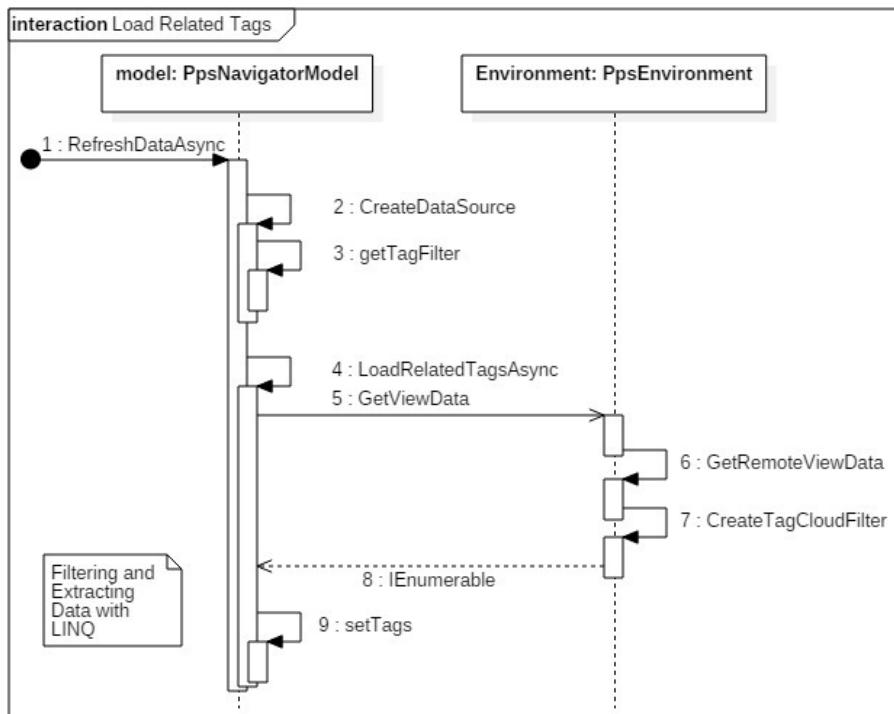


Abb. 5.8: Daten laden

Zusätzlich zu den bereits vorhandenen Aufrufen von RefreshDataAsync, wird diese Methoden nun auch ausgeführt, wenn ein Tag dem Filter hinzugefügt ([Abbildung 5.9](#)) oder vom Filter entfernt ([Abbildung 5.10](#)) wird.

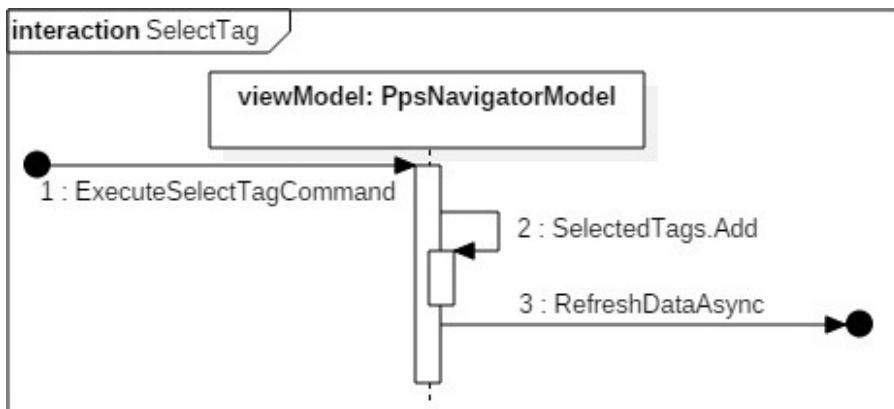


Abb. 5.9: Ablauf: Tag auswählen

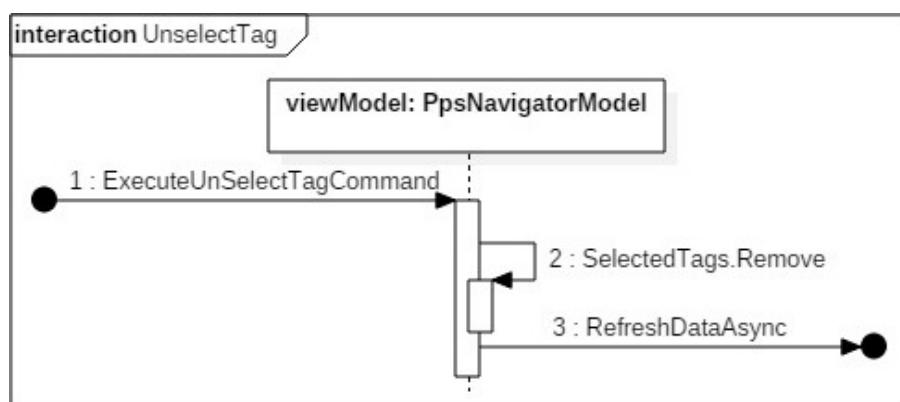


Abb. 5.10: Ablauf: Tag aus Auswahl entfernen

6 Performance-Tests

Die Tag-Cloud ist ein Teil des Filtersystems des ERP-Systems. Daher spielt bei der Erstellung der Tag-Cloud die Erstellungszeit eine wichtige Rolle. Um diese Erstellungszeit optimieren zu können, musste als erstes ein besonders gutes Tag-Cloud-Aussehen hergestellt werden, um einen Vergleich zu ermöglichen.

6.1 Veränderte Parameter der spiralförmigen Tag-Cloud-Erstellung

Bei der Erstellung einer Spirale sind zwei Parameter sehr wichtig, Wachstumsfaktor der Spirale und die Größe des Einfügesektors. Die [Abbildung 6.1](#) zeigt die Parameter in Zusammenhang mit einer Spirale.

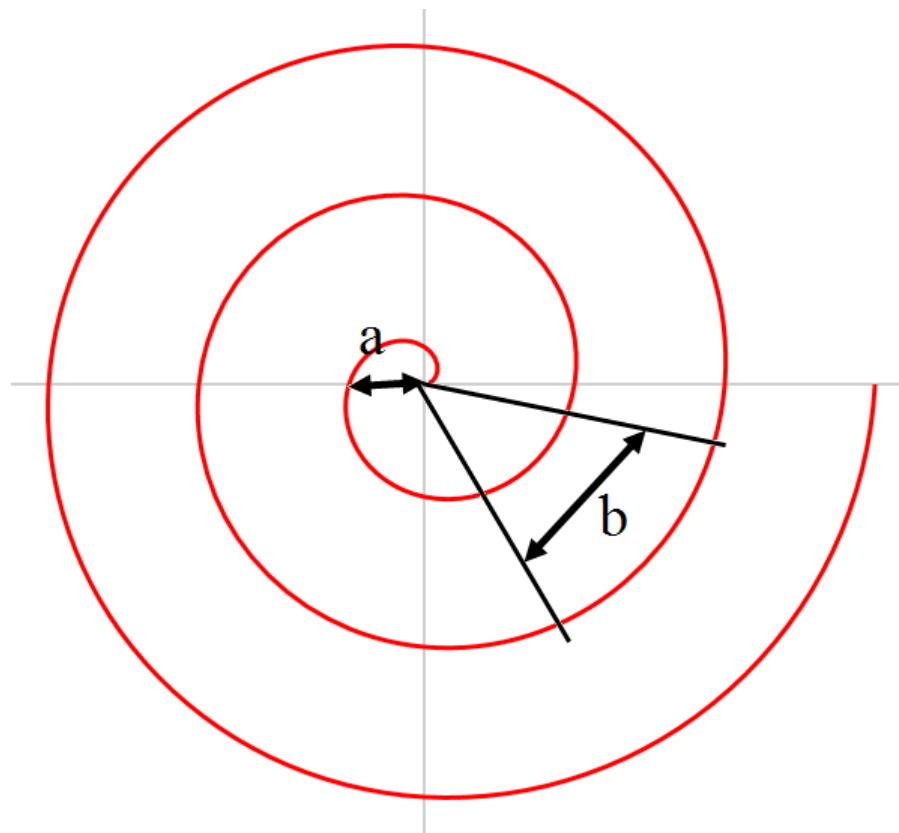


Abb. 6.1: Spirale, a: Wachstumsfaktor, b: Größe des Einfügesektors

6.2 Aussehen einer optimalen Tag-Cloud

Ein optimales Aussehen der Tag-Cloud ([Abbildung 6.2](#)) zeichnet sich durch eine gute Platzausnutzung bei möglichst wenig Löchern aus. Der Abstand zwischen den einzelnen Tags sollte minimal, jedoch noch deutlich erkennbar sein. Ein gutes Beispiel zeigt die folgende Abbildung.

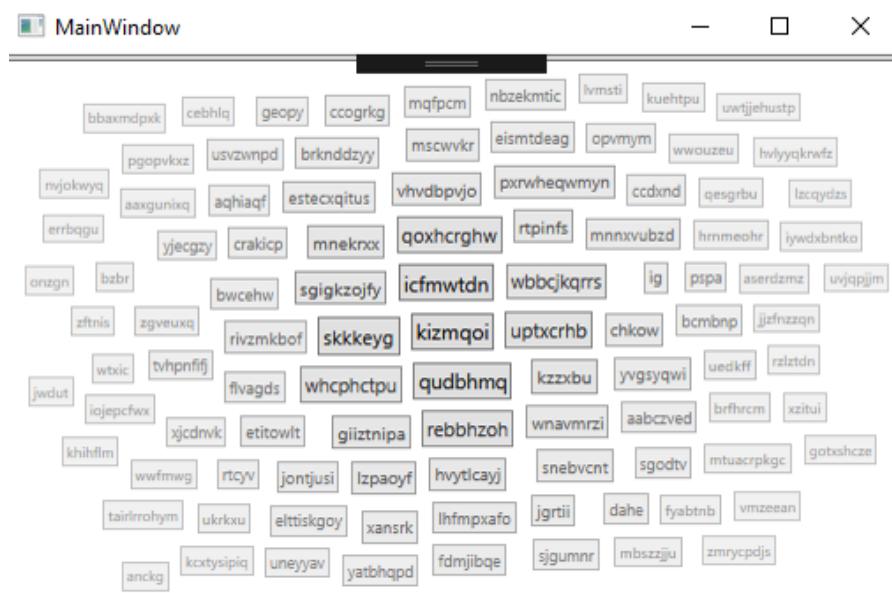


Abb. 6.2: Aussehen einer Optimalen Tag-Cloud

In der oben gezeigten Abbildung sind der Wachstumsfaktor mit 0,1 und die Größe der Einfügesektoren mit 120 Grad angegeben. Bei den Tests des zeitlichen wir die Tag-Cloud mit den genannten Werten in 944 Millisekunden im Durchschnitt erstellt. Um das Bild darzustellen werden zudem 99.770 Positionierungsversuche im Durchschnitt unternommen. Eine Tag-Cloud-Erstellung mit diesen Durchschnittswerten ist nicht für den Einsatz in einem Filtersystem gedacht.

6.3 Umsetzung im Algorithmus

Damit die Anweisungen funktionieren müssen die Namespaces `System.Diagnostics` und `System.IO` in den Algorithmus eingebunden werden. Als Erstes wird die Stoppuhr ([Abbildung 6.3](#)) für die Zeitmessung deklariert.

Eine Funktion `Stopp` ([Abbildung 6.4](#)) mit den Aufrufparametern für die Stoppuhr `watch`, den zu messenden Wert `messwert` und eine Zählvariable `lauf` ist ebenfalls wichtig. In dieser Methode wird die Stoppuhr gestoppt. Die Übergabe der Zeit, der Anzahl der Durchläufe und des Messwertes an die Methode `Append` sind weitere Inhalte dieser Methode.

Mit der Funktion `Append` ([Abbildung 6.5](#)) werden die übergebenen Daten in eine Textdatei für die Auswertung exportiert.

```

public System.Diagnostics.Stopwatch Zeit()
{
    var watch = System.Diagnostics.Stopwatch.StartNew();
    return watch;
}

```

Abb. 6.3: Funktion Zeit

```

public long Stopp(System.Diagnostics.Stopwatch watch,
                  string messwert, int lauf)
{
    watch.Stop();
    var elapsedMs = watch.ElapsedMilliseconds;
    string dateiname = speicherort;
    string neueZeile = messwert + ":" + elapsedMs + ", "
                      + lauf + "\r\n";
    Append(dateiname, neueZeile);
    return elapsedMs;
}

```

Abb. 6.4: Funktion Stopp

Innerhalb der Methode `CalculateWordAppearence` muss als erste Anweisung die Stoppuhr gestartet werden. Danach wird die Zählvariable `dlauf` deklariert und mit dem Wert 0 initiiert ([Abbildung 6.6](#)).

In der Do-While-Schleife in der das Tag positioniert wird, muss die Zählvariable bei jedem Durchlauf um eins erhöht werden. Am Ende der `CalculateWordAppearence`-Methode muss noch die Methode `Stopp` mit `Stopp(stoppzeit, "CalcWordAppear", dlauf)`; aufgerufen werden, damit die Stoppuhr angehalten wird.

```

public void Append(string sFilename, string sLines)
{
    StreamWriter myFile = new StreamWriter(sFilename,
                                           true);
    myFile.WriteLine(sLines);
    myFile.Close();
}

```

Abb. 6.5: Funktion Append

```

var stoppzeit = Zeit();
var dlauf = 0;

```

Abb. 6.6: Deklaration für Messung

6.4 Erweiterung des Spiralform-Verfahrens mit dem Halbierungsverfahren

Für dieses Verfahren wurde ein Performance-Test durchgeführt. Es wurde ein ca. 3 Jahre alter, handelsüblicher Desktop PC im Labor der HTW eingesetzt um Vergleichbarkeit zu einem Bülorechner herzustellen.

Mit den Ausgangswerten 0,1 für den Wachstumsfaktor und 120 Grad für die Größe der Einfügesektoren (Optisches Ergebnis s. [Abbildung 6.7](#)) ist bei Messung auf einem Notebook eine Erstellungszeit von 998 Millisekunden bei 71.321 Positionierungsversuchen im Durchschnitt herausgekommen. Diese Werte sind für eine Tag-Cloud-Erstellung in einem ERP-System ungeeignet.

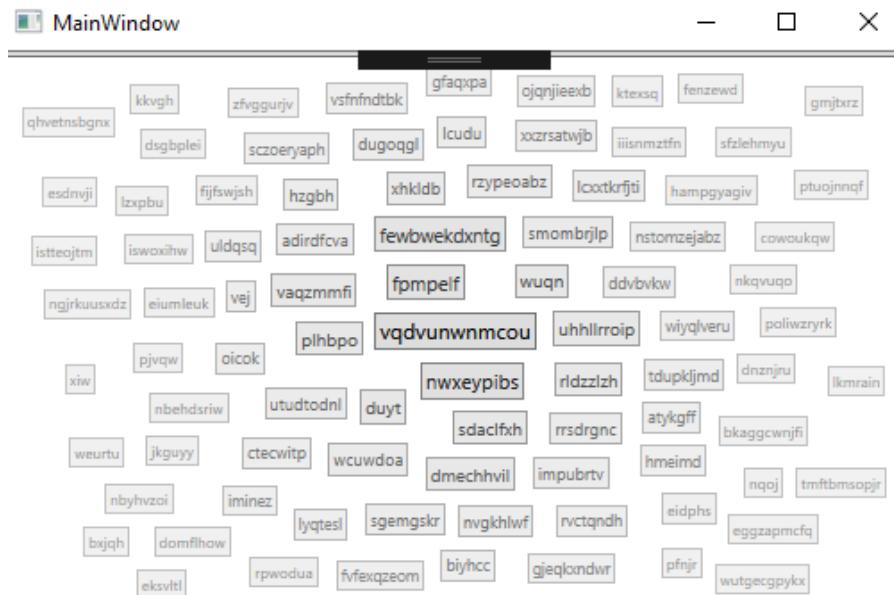


Abb. 6.7: Tag-Cloud-Aussehen mit Startwerten unter zusätzlicher Nutzung des Halbierungsverfahrens

Der Wachstumsfaktor von 0,9 und Einfügesektor von 120 Grad liefern sowohl ein ansprechendes Tag-Cloud-Aussehen ([Abbildung 6.8](#)) als auch im Durchschnitt Performancewerte von 75 Millisekunden bei 11.516 Positionierungsversuchen.

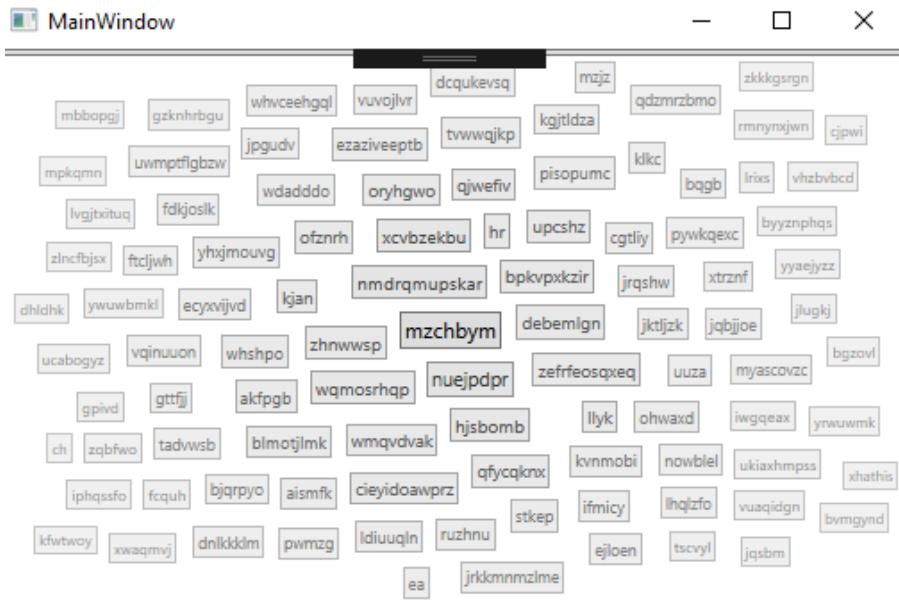


Abb. 6.8: Abbildung: Tag-Cloud-Aussehen mit optimierten Werten mit Halbierungsverfahrens

6.5 Performance-Tests und Auswertung

Beim Testen verschiedener Werte des Wachstumsfaktors fällt auf, dass sich die Erstellungszeit mit größer werdendem Wachstumsfaktor bei gleicher Größe der Einfügesektoren verringert. Es liegt allerdings kein lineares Verhalten vor. Ein ähnlicher Verlauf bei den Einfügesektoren und gleichbleibendem Wachstumsfaktor ist nicht zu erkennen. Das Minimum der Positionierungsversuche schwankt sehr stark und es ist kein einfaches mathematisch erklärabares Verhalten ableitbar.

Die besten Parameterwerte die zu einem guten Tag-Cloud-Aussehen ([Abbildung 6.9](#)) führen sind mit 122 Millisekunden bei 2.988 Positionierungsversuchen im Durchschnitt erreicht.

Hierbei beträgt die Einfügesektor-Größe 30 Grad und der Wachstumsfaktor 0,9. Bei anderen Parameterwerten für Größe des Einfügesektors und Wachstumsfaktor gibt es bessere Performance-Werte, jedoch ist entweder die Platzausnutzung innerhalb des Positionierungsfensters nicht gegeben oder es gibt viele großer Lücken innerhalb der Tag-Cloud.

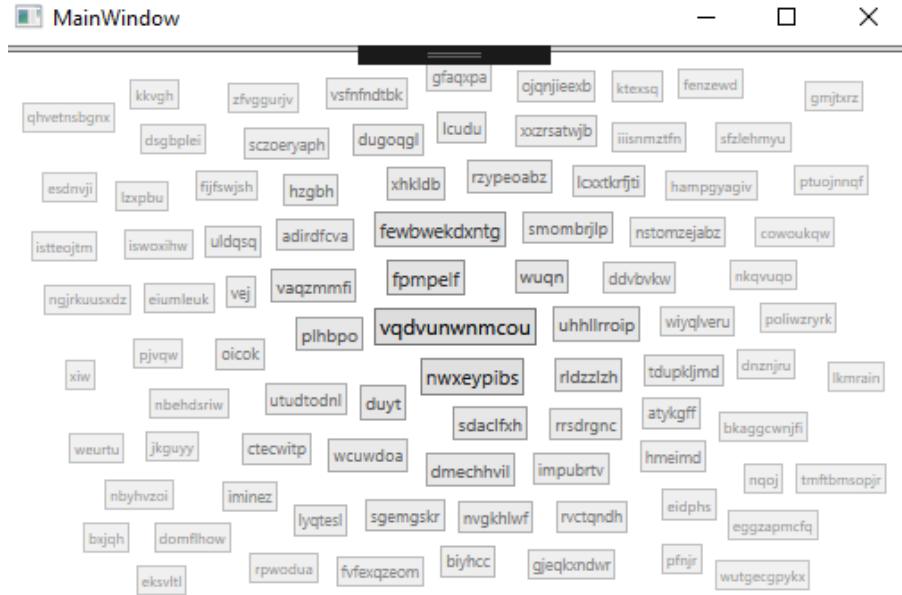


Abb. 6.9: Abbildung: optimales Tag-Cloud-Aussehen

6.6 Entscheidung zur Performance

Für die Erstellung einer Tag-Cloud mit Hilfe des Spiralform-Verfahrens wurde anhand der optischen Erscheinung und festgestellten Performance als optimale Parameter für die Einfügesektor-Größe 30 Grad und für den Wachstumsfaktor 0,9 ohne Nutzung des Halbierungsverfahrens ausgewählt. Das liegt zum einen an der noch nicht fertiggestellten Umsetzung des Halbierungsverfahrens. Dieses sollte zum einen bei einer sauberen Implementierung bessere Durchschnittserstellungszeiten liefern. Zum anderen benötigt das normale Spiralform-Verfahren wesentlich weniger Positionierungsversuche, auf die wir ein größeres Gewicht gelegt haben. Unserer Ansicht nach sind die ungefähren 50 Millisekunden Unterschied in der Erstellungszeit vernachlässigbar. 122 Millisekunden sind für die Erstellung einer Tag-Cloud annehmbar.

7 Text Mining

Unter Text Mining (Wikipedia 2017b) wird ein Bündel von Algorithmus-basierten Analyseverfahren zur Entdeckung von Bedeutungsstrukturen verstanden, die Kerninformationen schnell erfassen können. Gedacht sind diese Verfahren für die Verarbeitung und Durchsuchen großer Textmengen.

7.1 Aufgabe

Zu Beginn des Projektseminars kam schnell die Frage auf, wie die Tags angelegt werden sollen. Die Eingabefelder der Artikel könnten automatisch zu Tags transformiert werden. Aber wie wird dann das Beschreibungsfeld behandelt. Ein Beispiel hierzu wäre ein Nutzer, der einen neuen Artikel anlegt und das Beschreibungsfeld mit folgendem Text füllt: „Kotflügel für großes Auto“. Folgende Tags sollten automatisch gesetzt werden: Kotflügel, groß und Auto. Doch wie ist es möglich, das Wort „großes“ zu dem Wort „groß“ als Tag umzuwandeln. Durch diese Fragestellung sind die Studenten auf das Thema Text Mining aufmerksam geworden.

7.2 Vorgehen

Folgende drei Teilbereiche des Text Mining wurden behandelt. Die Stammformreduktion (Stemming), die Lemmatisierung und das Part-of-Speach Tagging.

Bei der Stammformreduktion wird das Wort auf eine Art Wortkern reduziert, der nicht unbedingt in der Sprache existiert. Bei der Lemmatisierung wird das Wort auf seiner lexikalischen Grundform reduziert. Als Part-of-Speach Tagging (Wikipedia 2013) bezeichnet man die Zuordnung von Wörtern und Satzzeichen eines Textes zu Wortarten. Hierzu werden sowohl die Definition des Wortes als auch der Kontext berücksichtigt.

7.3 Mögliche Lösungsansätze

7.3.1 Stammformreduktion

Im Bereich der Stammformreduktion haben die Studenten sich mit dem Porter-Stemmer-Algorithmus auseinandergesetzt. Der Porter-Stemmer-Algorithmus (Wikipedia 2016a) ist ein verbreiteter Algorithmus der Computerlinguistik zum automatischen Zurückführen von Wörtern auf ihren Wortstamm (Stemming). Der Algorithmus basiert auf einer Menge von Verkürzungsregeln, die so lange auf ein Wort angewandt werden, bis dieses eine Minimalanzahl von Silben aufweist. Der ursprünglich für Wörter der englischen Sprache entwickelte Algorithmus kann relativ leicht für andere Sprachen portiert werden.

Zuerst wird die Anzahl der Vokal-Konsonant-Sequenzen bestimmt. Jedes Wort lässt sich als Zeichenkette in der Form [C] (VC)^m [V] interpretieren, wobei C für eine Folge von einem oder mehreren Konsonanten und V für eine Folge von einem oder mehreren Vokalen steht. Gemessen wird hier die Anzahl m der Vokal-Konsonant-Sequenzen zwischen optional führenden Konsonanten und einer optionalen Folge von Vokalen am Ende. Beispiele:

- w[eb] (m=1)
- b-[et]-ew-[en] (m=2)
- tr-ee (m=0)

Nach der Bestimmung der Silben werden die vorher festgelegten Verkürzungsregeln abgearbeitet. Die Verkürzungsregeln bestehen aus Paaren von Bedingungen und Ableitungen für verschiedene Suffixe (Wortendungen). Die Verkürzungsregeln werden in Gruppen zusammengefasst. Aus jeder Gruppe darf nur eine Regel angewendet werden. Beispiel: Die erste Gruppe beinhaltet die Suffix-Verkürzungsregeln „sses“ → „s“, „ies“ → „i“ und „s“ „“, die beispielsweise zu den Ableitungen „libraries“ → „librari“ und „Wikis“ → „Wiki“ führen. Eine später folgende Gruppe besteht aus der Regel „y“ → „i“, so dass beispielsweise das Wort „library“ auf den gleichen Stamm („library“ → „librari“) zurückgeführt wird. Zum besseren Verständnis haben die Studenten den Porter-Stemmer-Algorithmus (s. Abb. 7.1) zusätzlich für die deutsche Sprache ausprobiert.

$[C](VC)^m[V]$
 - $m = 3$: Statistik St-at-is-t-ik
 - $m = 0$: Treue Tr-eue
 Bedingung z.B. ($m > 1$)
 Suffix (S1) → Suffix (S2) S1= „er“ , S2= „“

Beispiel:

Kleiner → Kl-e-in-er	Mauer → M-au-er
M=2	M = 1
Bedingung erfüllt	Bedingung nicht erfüllt
S1 wird zu S2	Ergebnis = Mauer
Ergebnis = klein	

Negativbeispiel:

Kater → K-at-er	$m = 2$
Ergebnis = Kat	

Abb. 7.1: Porter-Stemmer-Algorithmus

7.3.2 Lemmatisierung

Im Bereich Lemmatisierung wurde festgestellt, dass ein Wörterbuch im Hintergrund zum Abgleichen benötigt wird. Folgende Möglichkeiten wurden dabei näher betrachtet:

- Elasticsearch
- GermaNet
- Sketch Engine

Elasticsearch

Elasticsearch (Wikipedia 2017a, Elasticsearch 2010) ist eine Suchmaschine, die mit indizierten Dokumenten arbeitet. Die kleinste Einheit bilden dabei die Dokumente. Diese werden indiziert, um sie durchsuchen zu können. Mehrere Dokumente bilden einen Typ und mehrere Typen werden als Index bezeichnet. Vergleicht man diesen Aufbau mit einer SQL-Datenbank, wäre ein Dokument eine Zeile, der Typ die Tabelle und der Index die Datenbank.

Funktionsweise:

Zur Indexierung wird ein Dokument im JSON-Format an Elasticsearch gesendet. Dort wird daraufhin ein Analyseprozess gestartet. Währenddessen wird das Dokument aufbereitet, indem der Text in einzelne Worte und die Worte in einzelne Buchstaben zerlegt werden. Eigene Umwandlungsstufen sind einbaubar. Die Resultate des Analyseprozesses werden im Index gespeichert und können dann bei einer Suchanfrage durchsucht werden. Die ursprünglich übermittelten Dokumente werden zusätzlich an einem anderen Ort gespeichert.

Ergebnis:

Da die Studenten für die gestellte Aufgabe die Lemmata eines Wortes ermitteln wollen, wurde Elasticsearch nicht genauer untersucht.

GermaNet

GermaNet (Wikipedia 2014, UniTübingen 2009) ist eine elektronische, lexikographische Referenzdatenbank für den deutschen Wortsinn. Es stammt von der Universität Tübingen. Für GermaNet wurde eine Nutzungs Lizenz über die HTW Dresden erworben. Die Möglichkeit für ein eingegebenes Wort, das Lemma zurück zu erhalten, wurde von den Studenten getestet. Da GermaNet für den semantischen Zusammenhang von Worten gedacht ist, war es nicht möglich nur das Lemma für ein Wort zu erfragen.

Sketch Engine

Sketch Engine (Wikipedia 2016b, LexicalComputing 2016) ist eine seit 2004 von Lexical Computing Limited entwickelte Corpus Manager- und Analysesoftware. Mittels einer 30-Tage-Testversion haben wir die Möglichkeiten der Sketch Engine überprüft. Sketch Engine bietet u.a. folgende Funktionen:

- Wortlisten
- N-Gramm
- Verteilungs-Thesaurus
- Konkordanzsuche
- Korpusvergleich

Auch hier sind die Studenten nicht weitergekommen, da die Sketch Engine und nicht ein einzelnes Lemma zurückgeben werden kann.

7.3.3 TreeTagger

Der TreeTagger (Helmut Schmid 1995) der Universität München ist ein Werkzeug zur Annotation eines Textes mittels Part-of-speech Tagging und Lemmata Informationen. Entwickelt wurde er von Helmut Schmidt in dem TC Projekt am Institut für Computerlinguistik der Universität Stuttgart. Er kann erfolgreich Deutsch, Englisch, Französisch und viele weitere Sprachen taggen. Er besitzt einen handtrainierten Textkorpus, der individuell erweiterbar ist. Einsetzbar ist er unter Windows, Linux und Mac-OS. Der TreeTagger (Helmut Schmid 1994) und das Trainingsprogramm ist frei verfügbar für Forschung, Bildung und Evaluation. Zusätzlich ist unter Windows ein graphisches Interface verfügbar.

Funktionsweise:

Der TreeTagger benötigt seinen Textkorpus. Er berechnet zwei Wahrscheinlichkeitsarten und weist dann dem wahrscheinlichsten Tag, die Wortart und das Lemma, zu. Die erste Wahrscheinlichkeit ist lexikalisch, d.h. der TreeTagger durchsucht einen Textkorpus nach einem Wort, um damit die Wortart und das Lemma ausgeben zu können. Die zweite Wahrscheinlichkeit ist distributional und entsteht durch den Zusammenhang von Wörtern in einem Satz. Aus beiden Wahrscheinlichkeiten wird eine gesamte Wahrscheinlichkeit berechnet und der TreeTagger ordnet demnach die Wortart zu und gibt das Lemma an. Er rechnete mit zwei Wahrscheinlichkeiten, da gerade in der deutschen Sprache, je nach Verwendung des Wortes in einem Satz, ein Wort unterschiedlichen Wortarten angehören kann. Beispiel: Ich wasche meine Hose.

Ausgabe TreeTagger:

Wort	Wortart	Lemma
Ich	/PPER	Ich
wasche	/VFIN	waschen
meine	/PPOSAT	ich
Hose	/NN	Hose
.	/\$.

- PPER = Personalpronomen
- VFIN = Verb finite Form
- PPOSAT = Possessivpronomen
- NN = Nomen
- \$ = Satzzeichen

In diesem Fall muss der TreeTagger bei dem Wort „meine“ zwei Wahrscheinlichkeiten vergleichen. „meine“ kann sowohl als Possessivpronomen (eine Form des Wortes ich) oder als finites Verb (ein Verb des Wortes Meinung) zugeordnet werden. Jetzt findet

der TreeTagger in seinem Korpus zwei Mal eine Wortart für das Wort „meine“ aus dem Beispielsatz. Zu je einer Wahrscheinlichkeit ist das Wort „meine“ ein Verb oder ein Pronomen. Findet der TreeTagger ein Wort nicht in seinem Korpus, ersetzt er alle Großbuchstaben durch Kleinbuchstaben und startet die Suche von vorne, sollte er es danach nicht finden. Die zweite Wahrscheinlichkeit mit der der TreeTagger arbeitet ist distributional. Wie wahrscheinlich ist es, dass aus dem obigen genannten Beispielsatz auf ein Verb (wasche) ein weiteres folgt und wie wahrscheinlich ist es, dass auf ein Verb (wasche) ein Possessivpronomen folgt. Anhand der umliegenden Wörter kann der TreeTagger genauere Zuordnungen treffen.

Auswertung:

Das erste Problem ist, dass der TreeTagger besser mit ganzen Sätzen arbeitet. Der weiteren bedeutet Wahrscheinlichkeit nicht Sicherheit. Bei 20.000 Wörtern ordnet der TreeTagger zu 97,53% die richtige Wortart zum jeweiligen Wort zu. Die größte Schwachstelle, die die Studenten im TreeTagger sahen, war die Tatsache, dass er nicht hauptsächlich zur Findung der Lemmata dient, sondern Wortarten zuordnen soll. Das der TreeTagger zusätzlich anhand seines Korpus auch Lemmata ausgeben kann, ist nur ein schöner Zusatz. Würden die Studenten den TreeTagger für ihre o.g. Aufgabe einsetzen wollen, müssten sie den Textkorpus mittels des Trainingsprogrammes erweitern, d.h. händisch Wortarten und Lemmata zuordnen. Daraus wurde geschlossen, dass die Tagliste für das ERP-System händisch geführt werden kann. Der Aufwand wäre der Gleiche.

7.3.4 Wortschatzprojekt Uni Leipzig

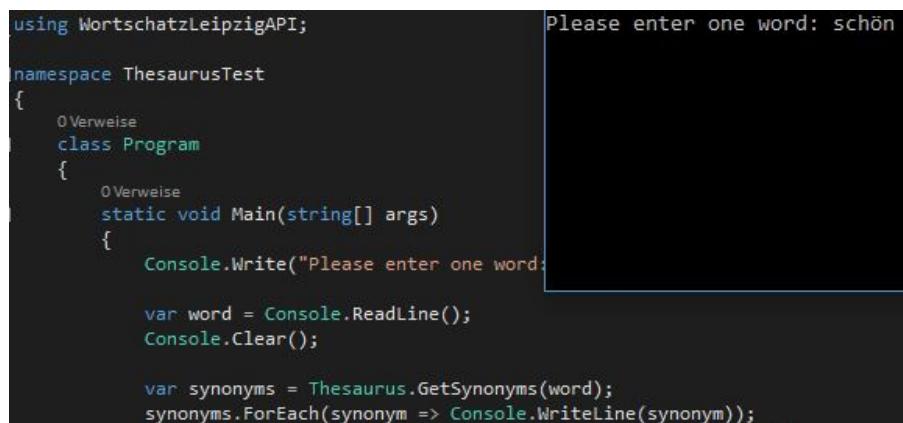
Das Wortschatzprojekt der Universität Leipzig (Herrmann 2013b) beinhaltet korpusbasierte monolinguale Wörterbücher für unter anderem die deutsche Sprache. Es wird seit 1998 von Gerhard Heyer und Uwe Quasthoff an der Universität Leipzig entwickelt. Es wurde ein umfassender Korpus des deutschen Wortschatzes als Vollformlexikon erstellt. Die über Jahre aufgebaute Lexikondatenbank wurde über das Internet zur Nutzung verfügbar gemacht und zur Bearbeitung und Ergänzung zur Verfügung gestellt. Seitdem wurde es beständig erweitert. Das Lexikon umfasst aktuell ca. 35 Millionen Beispielsätze mit 500 Millionen laufenden Wörtern. Der Zweck des Projektes soll vor allem der Sprachverarbeitung und Texttechnologie dienen. Die Annotation der Token umfasst das Sachgebiet Beispielsätze sowie die Grammatik. Da das Ziel aus der Ermittlung der Lemmata der Worte des Beschreibungsfeldes bestand, schien das Wortschatzprojekt eine geeignete Lösung darzustellen.

Umsetzung

Die Universität Leipzig stellt u.a. einige SOAP-Webservices (UniLeipzig 2017) zur Verfügung, mit denen ein direkter Datenzugriff sowie gezielte Abfragen möglich sind. Zu jedem Webservice wird ein Beispielclient mit Quellen zur Verfügung gestellt. Dies ermöglicht die Kombination sowie das Einbauen der Webservices in eigene Programme. Die Zugriffsvoraussetzungen unterscheiden sich je nach Abfrageumfang. Für einfache Abfragen kann das login „anonymous“ mit dem Passwort „anonymous“ verwendet werden. Eine Registrierung ist für komplexere Abfragen allerdings erforderlich, um bei Problemen kontaktiert werden zu können. Massendatenabfragen sind lediglich Kooperationspartnern vorbehalten. Da der Verwendung des Wortschatzprojektes vorerst lediglich ein „Versuch“ durchgeführt werden sollte, wurde sich auf das allgemeine Login für einfache Abfragen beschränkt. Nach einigen Recherchen mit dem Ziel einen bereits bestehenden und benutzbaren Code zur Abfrage des Lemmata zu erhalten, sind die Studenten auf eine C#-API zur Abfrage des Thesaurus gestoßen. Diese API wurde von Raffael Herrmann im Juli 2013 entwickelt.

Die Funktionsweise und der Aufbau des Codes wurde in (Herrmann 2013a) ausführlich erklärt, so dass hier nicht weiter darauf eingegangen wird.

Nach Start der API öffnet sich ein Konsolenfenster (Abbildung 7.2). Nun ist es möglich ein Wort einzugeben von welchem man den zugehörigen Thesaurus als Rückgabewert erhalten möchte.



The screenshot shows a terminal window with the following text:

```
using WortschatzLeipzigAPI;
namespace ThesaurusTest
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please enter one word:");
            var word = Console.ReadLine();
            Console.Clear();

            var synonyms = Thesaurus.GetSynonyms(word);
            synonyms.ForEach(synonym => Console.WriteLine(synonym));
        }
    }
}
```

Please enter one word: schön

Abb. 7.2: Eingabe C# - API Thesaurus

Als Rückgabe (Abbildung 7.3) erhält man alle gefundenen Thesauri, sowie die Anzahl der Synonyme.

```

using WortschatzLeipzigAPI;

namespace ThesaurusTest
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please enter one word");

            var word = Console.ReadLine();
            Console.Clear();

            var synonyms = Thesaurus.GetSynonyms(word);
            synonyms.ForEach(synonym => Console.WriteLine(synonym));
            Console.WriteLine("Found " + synonyms.Count);
        }
    }
}

```

schmuck
gewinnend
fotogen
fesch
berückend
distinguiert
bildschön
kleidsam
auserlesen
wohlgeformt
bildhübsch
wonnevoll
wonniglich
wohlgestaltet
schoen
unbewölkt
malenswert
Found 81 synonyms.

Abb. 7.3: Rückgabe C# - API Thesaurus

Das Ziel bestand daraus, statt des Thesaurus die Lemmata des Eingabewortes zu erhalten. Darüber hinaus sollte es möglich sein mehrere Worte nacheinander abzufragen. Die Eingabe sollte außerdem über einlesen einer Liste aus einem Textdokument erfolgen. Folgende Codeabschnitte wurden daher geändert:

Änderungen der Funktion `GetSynonyms()` sind in Abbildung 7.4 erkennbar.

```

23 |     BasicHttpBinding basicAuthBinding = new BasicHttpBinding(BasicHttpSecurityMode.TransportCredentialOnly);
24 |     basicAuthBinding.Security.Transport.ClientCredentialType = HttpClientCredentialType.Basic;
25 |     EndpointAddress basicAuthEndpoint = new EndpointAddress("http://wortschatz.uni-leipzig.de:8100/axis/services/Baseform");
26 |
27 |     BaseformClient service = new BaseformClient(basicAuthBinding, basicAuthEndpoint);
28 |     var requestInterceptor = new InspectorBehavior();
29 |     service.Endpoint.Behaviors.Add(requestInterceptor);
30 |
31 |     service.ClientCredentials.UserName.UserName = "anonymous";
32 |     service.ClientCredentials.UserName.Password = "anonymous";
33 |
34 |     DataMatrix matrix = new DataMatrix();
35 |     List<DataVector> vector = new List<DataVector>();
36 |     DataVector dv = new DataVector();
37 |     dv.Add("Wort");
38 |     dv.Add(inputWord);
39 |     vector.Add(dv);
40 |
41 |     matrix.AddRange(vector);
42 |
43 |     RequestParameter request = new RequestParameter();
44 |     request.corpus = corpus;
45 |     request.parameters = matrix;

```

Abb. 7.4: C# - Funktion GetBaseform

Zeile 23 bis 25 Änderung der Servicerefenz zur Abfrage der Lemmata.

Zeile 27 bis 29 Erstellen des Webservice-Clients.

Zeile 31 bis 32 Konfiguration des Clients, da der Webservice Zugangsdaten erfordert.

Zeile 34 bis 45 Erstellung der Aufrufparameter. Es sind 2 Parameter erforderlich, zum Einen das Wort zu welchem das Lemmata zurück gegeben werden soll sowie das Wörterbuch in dem gesucht werden soll. Das Wörterbuch wurde bereits im Funktionskopf übergeben (`string corpus = "de"`). Die Parameter können nicht als „String“ übergeben werden, sondern erfordern obige Datenstruktur.

Die Funktion Main() mit Änderungen ist in Abbildung 7.5 zu finden.

```
16 int counter = 0;
17 string line;
18
19
20 System.IO.StreamReader file = new System.IO.StreamReader(@"C:\Users\test.txt");
21 using (System.IO.StreamWriter file2 = new System.IO.StreamWriter(@"C:\Users\ergebnis.txt"))
22
23     while ((line = file.ReadLine()) != null)
24 {
25
26
27     var lemmas = Baseform.GetBaseform(line);
28     lemmas.ForEach(lemma => Console.WriteLine(lemma));
29     lemmas.ForEach(lemma => file2.WriteLine(lemma));
30     counter++;
31 }
```

Abb. 7.5: C# - FunktionMain

Zeile 16 bis 17 Anlegen der Hilfsparameter. Counter um Anzahl der Eingabe Worte im Dokument zu zählen.

Zeile 20 bis 21 Einlesen des Dokuments welches Liste mit Worten enthält Erstellen des Dokuments in dem die zurückgegebenen Lemmata gespeichert werden sollen.

Zeile 23 bis 31 Hier erfolgt der Aufruf der Funktion GetBaseform(); . Jedes zurückgegebene Lemma wird in die Datei ergebnis.txt geschrieben sowie auf der Konsole ausgegeben. Die Schleife wird so lange erneut aufgerufen bis die Datei test.txt keine Zeile mehr enthält.

Bei der Eingabe der Worte „guter“, „Katzen“ und „schneller“ erfolgt nun folgende Rückgabe (Abbildung 7.6).

```
gut
A
Katze
N
schnell
A
There were 3 lines in the test document.
```

Abb. 7.6: Rückgabe C# - APIBaseform

Zu jedem Wort wird das Lemma sowie die Wortart zurückgegeben. Bei obigem Beispiel also:

Wort	Lemma	Wortart
Guter	Gut	A (Adjektiv)
Katzen	Katze	N (Nomen)
Schneller	Schnell	A (Adjektiv)

Auswertung:

Generell eignet sich das Wortschatzprojekt sehr gut zur Abfrage der Lemmata und auch der Wortart eines Wortes. Bei einer Eingabe von 263 Wörtern aus einem Buchabschnitt konnten die Studenten eine richtige Lemmatazuordnung von 92% feststellen.

Da es sich um sehr fachspezifische Daten handelt, welche als Eingabe erfolgen würden, ist das Wortschatzprojekt nicht geeignet. Bei unbekannten Wörtern erfolgt keine Rückgabe. Bei einer Eingabe von 199 Wörtern aus einem Abschnitt eines Zahnmedizinischen Katalogs lag die Genauigkeit nunmehr bei 62%. Darüber hinaus führt die Eingabe von Zahlen zu einer Überflutung an Rückgaben.

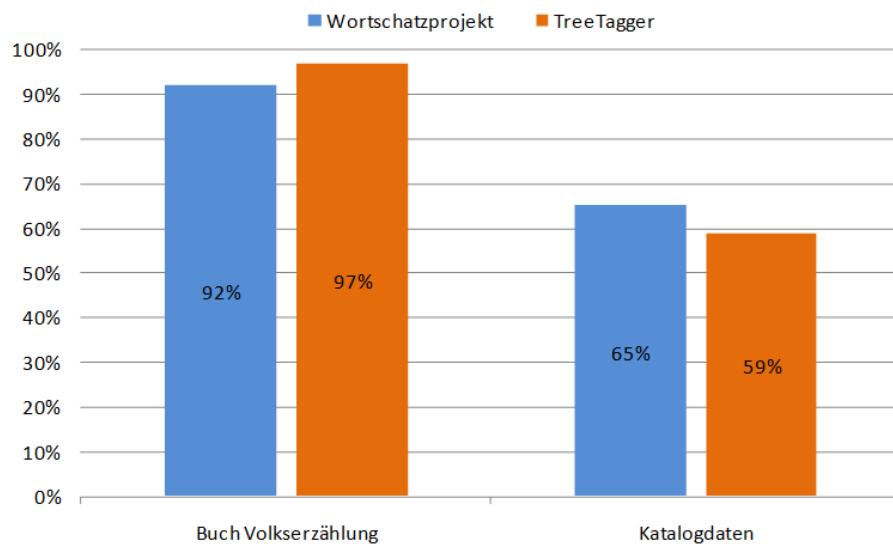


Abb. 7.7: TreeTagger vs. Wortschatzprojekt

8 Fazit

Die Einarbeitung in das PPSn System war anfangs auf Grund der hohen Komplexität der Softwarearchitektur und fehlenden Dokumentation sehr schwierig. Dank der Unterstützung von neolithos und der Erstellung eines einfachen Prototyps für initiale Testzwecke konnte dem sehr gut entgegengewirkt werden, sodass letztendlich die WordCloud erfolgreich in PPSn integriert wurde.

8.1 Text Mining

Der TreeTagger und das Wortschatzprojekt wurden miteinander verglichen. Dazu haben die Studenten das Buch „Volkserzählung“ (Tolstoj 1960) aus dem Gutenbergprojekt und Katalogdaten aus dem zahnärztlichen und kieferorthopädischen Bereich durch beide Programme laufen lassen. Aufgrund des begrenzten Abfragekontingentes des Wortschatzprojektes mussten die übergebenen Wörter stark begrenzt werden. Aus dem Buch „Volkserzählungen“ von Lew Tolstoi wurden 263 Wörter des ersten Kapitels übergeben. Hier lag der TreeTagger mit 97% (255) richtig zugeordneten Lemmata besser, als das Wortschatzprojekt mit 92% (241) richtigen Lemmata. Wie es auch im folgenden Diagramm veranschaulicht dargestellt wurde. Bei den Katalogdaten lag das Wortschatzprojekt mit 65% (130) richtig zugeordneten Lemmata vor dem TreeTagger mit nur 59% (117) richtiger Zuordnung. Es ist deutlich zu erkennen, dass der TreeTagger (Abbildung 7.7) mit umliegenden Wörtern in einem Satz arbeitet, was bei den Katalogdaten nicht gegeben war. Es handelt sich bei den Katalogdaten um einen Auszug auf einer Artikeldatenbank, der aus insgesamt 199 Wörtern bestand. Sollte Lemmatisierung in dem ERP-System umgesetzt werden, würde sich eine Mischung aus dem Wortschatzprojekt und dem TreeTagger empfehlen. Da die Daten im ERP-System eher wie die Katalogdaten aufgebaut sind, hat das Wortschatzprojekt mit seinem breiten Wörterstamm einen klaren Vorteil, was die Zuordnung bei einzelnen Worten vereinfacht.

Identifizierter Lösungsansatz

Alle anderen Felder bei der Anlage eines Artikels werden zu Tags, damit können die Wörter im Beschreibungsfeld erst mit den vorhandenen Tag abgeglichen werden. Mittels einer Füllwörterliste können nicht relevante Wörter wie (für, ein, und, usw.) herausgefiltert werden und nur bei den noch verbleibenden Wörtern, könnte mit Hilfe des Wortschatzprojektes oder auch des TreeTaggers das Lemma gefunden und als Tag hinzugefügt werden.

Quellen nachweise

- [1] Martin Dürr und Klaus Radermacher. *Einsatz von Datenbanksystemen – ein Leitfaden für die Praxis*. Berlin, Heidelberg [u.a.]: Springer, 1990. ISBN: 3540520805. URL: http://slubdd.de/katalog?TN_libero_mab2337570.
- [2] Elasticsearch. *Elasticsearch: RESTful, Distributed Search & Analytics*. Selfpublishing/Website. [Online; Stand 26. Februar 2017]. 2010. URL: <https://www.elastic.co/de/products/elasticsearch>.
- [3] Jonathan Feinberg. *Wordle*. 22. Mai 2010. URL: http://static.mrfeinberg.com/bv_ch03.pdf.
- [4] GruenderszeneLexikon. *Social Network*. [Online; Stand 27. Februar 2017]. 2010. URL: <http://www.gruenderszene.de/lexikon/begriffe/social-network>.
- [5] Raffael Herrmann. *CSharp-API für den Wortschatz Leipzig Thesaurus*. Selfpublishing. [Online, lastseen 25. Februar 2017]. 22. Juli 2013. URL: <https://code-bude.net/2013/07/22/csharp-api-fuer-den-wortschatz-leipzig-thesaurus-webservic/>.
- [6] Raffael Herrmann. *WortschatzLeipzigAPI*. [Online; Stand 26. Februar 2017]. 21. Juli 2013. URL: <https://github.com/codebude/WortschatzLeipzigAPI>.
- [7] R. Kluge. *Schablonen für alle Fälle*. SOPHIST GmbH. [Online; accessed 28. April 2016]. 2013. URL: https://www.sophist.de/fileadmin/SOPHIST/Publikationen/Broschueren/SOPHIST_Broschuere_MASTeR.pdf.
- [8] Daniel Lemire. „Tag-Cloud-Drawing – Algorithms for Cloud Visualization“. In: *Workshop on Tagging and Metadata for Social Information Organization*. Workshop on Tagging and Metadata for Social Information Organization. WWW2007, 11. Nov. 2008. URL: <https://de.slideshare.net/lemire/tagcloud-drawing-algorithms-for-cloud-visualization-presentation>.
- [9] LexicalComputing. *Sketch Engine – language corpus management and query system*. Selfpublishing Website. [Online; Stand 26. Februar 2017]. Apr. 2016. URL: <https://www.sketchengine.co.uk/>.
- [10] Steffen Lohmann. *Social Tagging and Folksonomies*. [Online; Stand 27. Februar 2017]. 19. Juli 2012. URL: <http://www.socialtagging.org>.
- [11] Olga Mantler und Viktor Gomer. *Folksonomie*. 24. Jan. 2006. URL: <https://www.techfak.uni-bielefeld.de/~swrede/xml-isy/talks/folksonomies.pdf>.

- [12] Adam Mathes. *Folksonomies - Cooperative Classification and Communication Through Shared Metadata*. [Online; Stand 28. Februar 2017]. Dez. 2004. URL: <http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html>.
- [13] Torsten Munkelt. *Social Tagging, Folksonomien und Tag-Clouds in ERP-Systemen*. OPAL HTW-Dresden. Juni 2016.
- [14] Helmut Schmid. *Improvements in Part-of-Speech Tagging with an Application to German*. Proceedings of the ACL SIGDAT-Workshop. Dublin, Ireland. [Online; Stand 26. Februar 2017]. 1995. URL: <http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger2.pdf>.
- [15] Herlmut Schmid. *Probabilistic Part-of-Speech Tagging Using Decision Trees*. Proceedings of International Conference on New Methods in Language Processing, Manchester, UK. [Online; Stand 26. Februar 2017]. 1994. URL: <http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger1.pdf>.
- [16] Mike Talbot. *WordCloud*. [Forked: Stand 26. Februar 2017; <https://github.com/SemAhto/WordCloud>]. 17. Aug. 2011. URL: <https://github.com/whydoidoit/WordCloud>.
- [17] Hermann Tolstoj Lev Nikolaevič Asemissen. *Volkserzählungen*. Republishing bei Projekt Gutenberg. [Online; Stand 26. Februar 2017]. Berlin : Rütten & Loening, 1960. URL: <http://gutenberg.spiegel.de/buch/volkserzahlungen-4048/2>.
- [18] Projekt Deutscher Wortschatz UniLeipzig. *Wortschatz Uni-Leipzig Webservices*. Selfpublishing Website. [Online; Stand 26. Februar 2017]. 2017. URL: <http://wortschatz.uni-leipzig.de/Webservices/>.
- [19] UniTübingen. *GermaNet Structure*. [Online; Stand 26. Februar 2017]. Dez. 2009. URL: http://www.sfs.uni-tuebingen.de/lst/germanet_structure.shtml.
- [20] Thomas Vander Wal. *Folksonomy*. [Online; Stand 27. Februar 2017]. 2. Feb. 2007. URL: <http://vanderwal.net/folksonomy.html>.
- [21] Wikipedia. *Elasticsearch — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 26. Februar 2017]. 2017. URL: <https://de.wikipedia.org/w/index.php?title=Elasticsearch&oldid=162905569>.
- [22] Wikipedia. *GermaNet — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 26. Februar 2017]. 2014. URL: <https://de.wikipedia.org/w/index.php?title=GermaNet&oldid=135334643>.
- [23] Wikipedia. *Part-of-speech Tagging — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 26. Februar 2017]. 2013. URL: https://de.wikipedia.org/w/index.php?title=Part-of-speech_Tagging&oldid=119249118.
- [24] Wikipedia. *Porter-Stemmer-Algorithmus — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 26. Februar 2017]. 2016. URL: <https://de.wikipedia.org/w/index.php?title=Porter-Stemmer-Algorithmus&oldid=152129261>.

- [25] Wikipedia. *Sketch Engine — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-February-2017]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Sketch_Engine&oldid=727453539.
- [26] Wikipedia. *Social Tagging — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 27. Februar 2017]. 2015. URL: https://de.wikipedia.org/w/index.php?title=Social_Tagging&oldid=144242080.
- [27] Wikipedia. *Text Mining — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 26. Februar 2017]. 2017. URL: https://de.wikipedia.org/w/index.php?title=Text_Mining&oldid=161693990.