Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 27

# Entwurf und Realisierung einer App zur Verwaltung von Taxonomien

Niklas Schnabel

| | |
|---|---|
| **Studiengang:** | Softwaretechnik |
| **Prüfer:** | Prof. Dr.-Ing. habil. Bernhard Mitschang |
| **Betreuerin:** | **M.A. Laura Kassner** |
| **Beginn am:** | 1. April 2015 |
| **Beendet am:** | 1. Oktober 2015 |
| **CR-Nummer:** | H.3.4, H.5.2, I.2.7 |

## Kurzfassung

Die Wartung von auf spezielle Themen zugeschnittene Taxonomien und Ontologien wird im Allgemeinen manuell durchgeführt. Dieser wiederkehrende Prozess wird oft von Experten der entsprechenden Domäne durchgeführt, benötigt viel Zeit und verursacht dadurch hohe Kosten. Um eine Kostensenkung zu erreichen, soll die benötigte Zeit gesenkt werden. Demzufolge soll im Rahmen dieser Arbeit ein Werkzeug entwickelt werden, das einen Experten bestmöglich bei der Wartung einer Taxonomie unterstützt. Hierzu wird ein Client-Server-System präsentiert, welches Taxonomien zentral vorhält und über einen Web-Service den Clients zur Verfügung stellt. Entwickelt werden sowohl der Server, als auch der Client.

Serverseitig wird eine Applikation vorgestellt, die in einer relationalen Datenbank enthaltene Taxonomien und Funktionen zu deren Verwaltung über eine REST-Schnittstelle zur Verfügung stellt.

Clientseitig wird eine auf Tablets ausgelegte Web-Applikation präsentiert, die in nahezu jeder Situation einen Zugriff auf eine Taxonomie erlaubt. Zusätzlich kann diese durch eine halbautomatische Erweiterung vervollständigt werden. Halbautomatisch bedeutet in diesem Kontext, dass Vorschläge für neue Synonyme für Konzepte außerhalb des Systems generiert und nach der Bewertung eines Experten in die Taxonomie übernommen werden können.

# Inhaltsverzeichnis


**1. Einleitung** **11**
- 1.1. Hintergrund und Motivation . . . 11
- 1.2. Aufbau der Arbeit . . . 12

**2. Wissenschaftliche Grundlagen** **13**
- 2.1. Taxonomien und Ontologien . . . 13
  - 2.1.1. Taxonomien . . . 13
  - 2.1.2. Ontologien . . . 14
- 2.2. Visualisierung von Baumstrukturen auf Tablets . . . 16
- 2.3. Hierarchische Strukturen in relationalen Datenbanken . . . 17

**3. Technologische Grundlagen** **19**
- 3.1. Representational State Transfer (REST) . . . 19
  - 3.1.1. Eindeutige Identifikation von Ressourcen . . . 19
  - 3.1.2. Hypermedia . . . 20
  - 3.1.3. HTTP-Verben . . . 20
  - 3.1.4. Repräsentationen . . . 22
  - 3.1.5. Zustandslos . . . 22
- 3.2. AngularJS . . . 22
  - 3.2.1. Single-Page Webanwendung . . . 22
  - 3.2.2. Model/View/ViewModel . . . 23
  - 3.2.3. Besonderheiten des Frameworks . . . 24

**4. Analyse** **29**
- 4.1. Anforderungen . . . 29
- 4.2. Vorhandene Taxonomie . . . 30
  - 4.2.1. `*_data.tax` . . . 31
  - 4.2.2. `*_forschung.tax` . . . 32
- 4.3. Vorhandener Taxonomieeditor . . . 33
  - 4.3.1. Eigenschaften des Editors . . . 33
  - 4.3.2. Schwächen des Editors . . . 35
- 4.4. Technologieanalyse . . . 36
  - 4.4.1. Native App . . . 36
  - 4.4.2. Web-App . . . 37
  - 4.4.3. Hybride App . . . 38
  - 4.4.4. Bewertung der verschiedenen Arten von Apps . . . 38
  - 4.4.5. Fazit . . . 40

4.5. Visualisierung einer Baumstruktur auf einem Tablet . . . . . 40
4.5.1. Einfache Liste . . . . . 41
4.5.2. Side-by-Side Liste . . . . . 42
4.5.3. Eingerückte Liste . . . . . 43
4.5.4. Tablorer . . . . . 44
4.5.5. Treemap . . . . . 45
4.5.6. Radial Edgeless Tree (RELT) . . . . . 45
4.5.7. Fazit . . . . . 46

**5. Entwurf und Realisierung** **49**
5.1. Aufbau des Gesamtsystems . . . . . 49
5.2. Serverseite . . . . . 50
5.2.1. Verwendete Technologien und Frameworks . . . . . 50
5.2.2. Allgemeiner Aufbau des Servers . . . . . 51
5.2.3. Benutzer- und Rechteverwaltung . . . . . 53
5.2.4. REST Schnittstelle für Taxonomien . . . . . 56
5.2.5. Persistierung der Daten . . . . . 59
5.2.6. Importieren vorhandener XML-Daten . . . . . 64
5.2.7. Exportieren der Daten . . . . . 68
5.2.8. Behandlung von Fehlern . . . . . 69
5.3. Clientseite . . . . . 72
5.3.1. Verwendete Technologien . . . . . 72
5.3.2. Allgemeiner Aufbau der App . . . . . 73
5.3.3. Importieren/Exportieren von Taxonomien . . . . . 75
5.3.4. Visualisierung der Taxonomie . . . . . 78
5.3.5. Visualisierung der Synonymvorschläge . . . . . 80
5.3.6. Suchfunktion innerhalb einer Taxonomie . . . . . 81
5.3.7. Kopieren und verschieben von Konzepten . . . . . 83
5.3.8. Behandlung von Fehlern . . . . . 85

**6. Evaluation** **87**
6.1. Vergleich zwischen neuem und altem Taxonomieeditor . . . . . 87
6.1.1. Funktionsvergleich . . . . . 87
6.1.2. Verarbeitbare Taxonomiegröße . . . . . 89
6.1.3. Visualisierung einer Taxonomie . . . . . 89
6.1.4. Behobener Fehler . . . . . 90
6.1.5. Benutzererfahrung . . . . . 90
6.2. Performance-Evaluation . . . . . 90
6.2.1. Verhalten bei einer großen Menge Taxonomien . . . . . 91
6.2.2. Verhalten bei einer in der Größe anwachsenden Taxonomie . . . . . 93
6.2.3. Bewertung der Ergebnisse . . . . . 95

**7. Zusammenfassung** **97**

**8. Ausblick** **99**
8.1. Server . . . . . . . . . . 99
8.2. Client . . . . . . . . . . 99
8.3. Systemübergreifend . . . . . . . . . . 100

**A. Anhang** **101**
A.1. Komplettes ER-Diagramm . . . . . . . . . . 102
A.2. Schritt zwei und drei des Importierens einer Taxonomie . . . . . . . . . . 103

**Literaturverzeichnis** **105**

# Abbildungsverzeichnis


2.1. Aufbau einfacher Taxonomien . . . 14
2.2. Aufbau einer einfachen Ontologie . . . 16

4.1. Der strukturelle Aufbau der gegebenen Taxonomie . . . 31
4.2. Mockup des vorhandenen Taxonomie-Editors . . . 34
4.3. Visualisierung des Elternkonzepts-Problems des vorhandenen Taxonomieeditors . . . 36
4.4. Visualisierungen einer Baumstruktur mit einer einfachen und einer Side-by-Side Liste 42
4.5. Visualisierungen einer Baumstruktur mit einer eingerückten List und Tablorer . . . 44
4.6. Visualisierungen einer Baumstruktur mit einer Treemap und RELT . . . 46

5.1. Aufbau des Gesamtsystems . . . 50
5.2. Paketdiagramm der obersten Ebene des Servers . . . 52
5.3. Realisierung der Autorisierung von Benutzern . . . 54
5.4. Vereinfachtes Klassendiagramm der aus [SFB+15] übernommenen Benutzer- und Rechteverwaltung . . . 55
5.5. Vereinfachtes Klassendiagramm des Paketes `taxonomy` . . . 58
5.6. Vereinfachtes Entity-Relationship-Diagramm der Datenbankstruktur . . . 61
5.7. Klassendiagramm der Datenbankschicht . . . 62
5.8. Gerichteter azyklischer Graph mit und ohne einer transitiven Hülle . . . 65
5.9. Vorgehensweise beim Importieren einer Taxonomie . . . 66
5.10. Klassendiagramm der Klassen, die den Importvorgang von Taxonomien handhaben . 67
5.11. Vorgehensweise zur Generierung eines Downloadlinks für den Export einer Taxonomie 69
5.12. Klassendiagramm der Klassen, die den Exportvorgang von Taxonomien handhaben . 70
5.13. Klassendiagramm der ExceptionMapper . . . 72
5.14. Logischer Zusammenhang zwischen den einzelnen Views des Clients . . . 74
5.15. Screenshot der View `Import` . . . 75
5.16. Screenshot der View zum Exportieren einer Taxonomie . . . 77
5.17. Screenshot der Hauptview . . . 78
5.18. Screenshot der View `Suggestions` . . . 81
5.19. Screenshot der View `Search` . . . 82
5.20. Screenshotausschnitt des Kopier/Verschiebe-Kontextmenüs . . . 83
5.21. Screenshot eines Alerts . . . 85

6.1. Auswertung der Evaluation mit einer großen Menge Taxonomien . . . 92
6.2. Auswertung der Evaluation mit einer sich stetig vergrößernden Taxonomie . . . 95

A.1. Komplettes ER-Diagramm . . . 102

A.2. Screenshot der App, der den zweiten Schritt des Importvorgangs zeigt . . . . . . . . . . 103
A.3. Screenshot der App, der den dritten Schritt des Importvorgangs zeigt . . . . . . . . . . 103

# Verzeichnis der Listings


3.1. Antwort von `http://example.com/products` im JSON-Format mit Links . . . . . . 20
3.2. Definition des Modules *taxonomy* in einer JavaScript-Datei, das eine Abhängigkeit auf das Modul *ngTouch* hat . . . . . . 24
3.3. Verwendung des AngularJS Moduls *taxonomy* in einer HTML-Datei . . . . . . 24
3.4. Definition des Controllers `taxonomyCtrl` in einer JavaScript-Datei . . . . . . 24
3.5. Verwendung des Controllers `taxonomyCtrl` und der in `$scope` definierten Variable `name` innerhalb einer HTML-Datei . . . . . . 25
3.6. Definition der Stelle, bei welcher Templates eingefügt werden sollen. Diese wird durch das Attribut `ui-view` in einer HTML-Datei markiert . . . . . . 25
3.7. Definition der Routen des *ui-router* in einer JavaScript-Datei . . . . . . 25
3.8. Definition der Stelle, bei welcher Templates eingefügt werden sollen. Diese wird durch das Element `ui-view` in einer HTML-Datei markiert . . . . . . 26

4.1. Daten einer Beispieltaxonomie, die zwei verschiedene Konzepte enthält. Beim ersten Konzept existiert nur ein deutsches Synonym und beim zweiten Konzept nur ein englisches . . . . . . 32
4.2. Relationen zwischen den in der `*_data.tax`-Datei definierten Konzepte . . . . . . 33

5.1. Verwendung von Promisses zur Fehlerbehandlung . . . . . . 86

# 1. Einleitung

In diesem Kapitel wird eine Einleitung in das Thema gegeben und die dahinterstehende Motivation erläutert. Zudem wird eine Übersicht über den Aufbau dieser Arbeit gegeben.

## 1.1. Hintergrund und Motivation

Semantische Ressourcen für die automatische Sprachanalyse können in verschiedenen Formen repräsentiert werden. Zu den am häufigsten verwendeten, dürften Taxonomien und Ontologien zählen. Allgemein sind diese Art von Daten aufwändig zu erstellen und deren Pflege ist mit viel Arbeit verbunden, was durch das enthaltene Wissen zu begründen ist. Dieses ist hauptsächlich bei Experten vorhanden, die im Allgemeinen hohe Kosten verursachen, und kann dementsprechend nur von diesen eingebracht werden. Das Ziel dieser Arbeit ist es, diesen Experten eine Applikation (folgend als App bezeichnet) für ein Tablet zu bieten, welches das Pflegen und Erweitern einer Taxonomie vereinfacht.

Um die Erweiterung einer Taxonomie zu beschleunigen sollen Vorschläge für neue Synonyme von Konzepten, die im Rahmen der Arbeit von Marcel Estel [Est15] generiert werden, innerhalb der App einem Experten zu deren Bewertung präsentiert werden. Hierdurch soll dieser entscheiden können, ob ein Vorschlag inhaltlich für die entsprechende Taxonomie geeignet ist. Bei Bejahung soll dieser aufgenommen, ansonsten im weiteren Verlauf ignoriert werden. Durch die Kombination dieser Arbeit mit [Est15] soll ein System geschaffen werden, das sowohl eine manuelle als auch semiautomatische Population der Taxonomie ermöglicht.

Als Grundlage wird eine manuell erstellte Taxonomie aus der Automobilindustrie verwendet [ST10], die in einem proprietären auf XML basierenden Format vorliegt. Zur Verwaltung dieses Taxonomieformats existiert ein in Java implementierter Editor. Dieser bietet keine Möglichkeit gegebene Vorschläge zu bewerten und kann ausschließlich über einen PC verwendet werden. Taxonomien werden lokal gespeichert, weswegen eine Zusammenarbeit mehrerer Personen erschwert wird.

Um diese Unzulänglichkeiten zu beseitigen soll eine von Grund auf neu erstellte App entwickelt werden. Die neue Applikation soll die Funktionen des schon vorhandenen Editors umsetzen und diesen zusätzlich um die Möglichkeit der Bewertung von Vorschlägen erweitern. Zudem soll diese mehreren Benutzern einen zeitgleichen Zugriff auf eine Taxonomie erlauben. Um das wichtige Thema der mobilen Nutzung zu beachten, soll diese auf Tablets ausgerichtet werden. Smartphones sollen aufgrund der geringen Bildschirmgröße keine Berücksichtigung finden.

Die Forschungsfrage, die sich hiermit für diese Arbeit ergibt, lautet wie folgt:

> *Wie lässt sich eine Taxonomie sinnvoll auf einem Tablet präsentieren, gleichzeitig von verschiedenen Benutzern bearbeiten und um gegebene Vorschläge erweitern?*

## 1.2. Aufbau der Arbeit

An dieser Stelle wird eine Übersicht über die nachfolgenden Kapitel gegeben. Dies soll dem Leser eine einfachere Orientierung innerhalb der Arbeit ermöglichen.

**Kapitel 2** beschäftigt sich mit den wissenschaftlichen Grundlagen der Arbeit. Zuerst wird eine Einführung in das Thema der Taxonomien und Ontologien gegeben (Kapitel 2.1). Anschließend werden Arbeiten vorgestellt, die sich mit der Visualisierung von Baumstrukturen auf Tablets beschäftigen (Kapitel 2.2). Abschließend findet eine Betrachtung der Arbeiten zur Speicherung von hierarchischen Strukturen in relationalen Datenbanken statt (Kapitel 2.3).

**Kapitel 3** stellt verwendete Technologien vor, die zum Verständnis der Arbeit zwingend notwendig sind und nicht vorausgesetzt werden. Hierzu werden die Grundlagen des *Representational State Transfers (REST)* erläutert (Kapitel 3.1) und eine kurze Einführung in das clientseitige JavaScript Framework *AngularJS* gegeben (Kapitel 3.2).

**Kapitel 4** beinhaltet eine genaue Analyse der Ausgangssituation und es werden verschiedene Techniken und Technologien näher betrachtet, die bei der Realisierung der App vonnöten sind. Zu Beginn werden die genauen Anforderungen an die App ermittelt (Kapitel 4.1). Darauf folgend wird der technische Aufbau der schon vorhandenen Taxonomie (Kapitel 4.2) und des vorhandenen Taxonomieeditor (Kapitel 4.3) näher betrachtet. Anschließend findet eine Analyse verschiedener Technologien statt, die zum Erstellen einer App genutzt werden können (Kapitel 4.4). Zum Schluss werden verschiedene Möglichkeiten untersucht eine Baumstruktur, sprich die gegebene Taxonomie, auf einem Tablet zu visualisieren (Kapitel 4.5).

**Kapitel 5** behandelt den Entwurf und die Realisierung der App, beziehungsweise des kompletten Systems. Das System besteht aus einem Client (Kapitel 5.3), sprich der zu entwickelnden App, und aus einem Serverteil (Kapitel 5.2), welcher die Verwaltung der Taxonomie übernimmt.

**Kapitel 6** beschäftigt sich mit der Evaluation des Systems. Zu Beginn findet ein Vergleich zwischen einem bereits existierenden Taxonomie-Editor und der entwickelten App statt (Kapitel 6.1). Anschließend wird eine Performance-Evaluation des Systems durchgeführt. Getestet wird zum einen inwiefern das System bei einer großen Menge kleiner Taxonomien verhält (Kapitel 6.2.1). Zum anderen wird evaluiert wie das System mit einer stetig in der Größe anwachsenden Taxonomie zurecht kommt (Kapitel 6.2.2).

**Kapitel 7** liefert eine kurze Zusammenfassung der Arbeit, die alle wesentlichen Eckpunkte enthält.

**Kapitel 8** gibt eine Ausblick darauf, wie die Ergebnisse dieser Arbeit in der Zukunft verwendet werden können und welche sinnvollen Erweiterungen es für die App, beziehungsweise das komplette System geben könnte.

# 2. Wissenschaftliche Grundlagen

Dieses Kapitel beschäftigt sich mit wissenschaftlichen Grundlagen, die von Bedeutung für diese Arbeit sind. Zu Beginn wird eine Einleitung in das Thema der Taxonomien und Ontologien gegeben. Anschließend wird der aktuelle Stand der Forschung bei der Visualisierung von Baumstrukturen auf mobilen Endgeräten betrachtet. Hierbei wird der Fokus auf Tablets gelegt. Zum Schluss werden Arbeiten vorgestellt, die sich mit der Speicherung von hierarchischen Strukturen, insbesondere azyklischen gerichteten Graphen, beschäftigen.

## 2.1. Taxonomien und Ontologien

Das grundlegende Thema dieser Arbeit bezieht sich auf Taxonomien, beziehungsweise Ontologien. Um den Inhalt der weiteren Ausarbeitung verstehen zu können, sind diese Begriffe und deren Bedeutung zu verstehen.

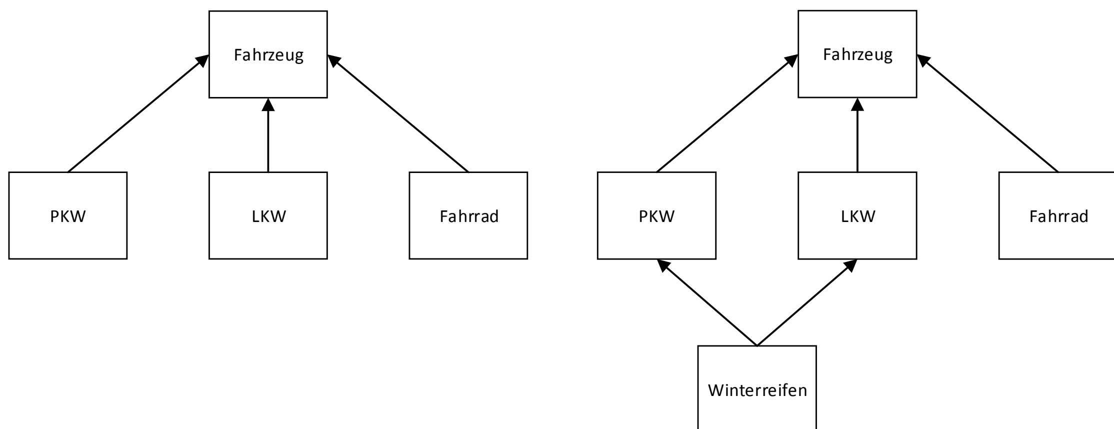
### 2.1.1. Taxonomien

Eine Taxonomie ist eine abstrakte Struktur, die eine Klassifikation verschiedener Begriffen beinhaltet. Grundsätzlich wird hiermit versucht ein Teil der realen Welt und deren Zusammenhänge in einem Model abzubilden. Die erste bekannt Taxonomie stammt von dem schwedischen Naturforscher Carl von Linné, welcher im 18. Jahrhundert eine hierarchische Klassifikation von Lebensformen erstellt hat. Noch heute ist dieses System die Grundlage für moderne zoologische und botanische Klassifikation [Gar04].

Eine Taxonomie besteht aus einer Menge von Begriffen, wobei zusammengehörige Begriffe gruppiert werden. Im Englischen werden diese Begriffe als *Concepts* bezeichnet, weswegen auch im Deutschen der Begriff Konzept gebräuchlich ist. Im Folgend werden die Wörter *Begriff* und *Konzept* synonym zu einander verwendet, da diese im Kontext von Taxonomien dieselbe Bedeutung haben.

Folgend soll der grundlegende Aufbau einer Taxonomie anhand eines Beispiels erläutert werden. Eine visuelle Repräsentation des Beispiels ist in Abbildung 2.1a zu sehen.

Der Begriff in der obersten Ebene einer Taxonomie wird als Wurzel bezeichnet, wobei jede Taxonomie immer genau eine Wurzel hat. Im Falle des Beispiels ist dies der Begriff `Fahrzeug`. Unterhalb des Begriffs `Fahrzeug` sind die Begriffe `PKW`, `LKW` und `Fahrrad` angeordnet. Hieraus lässt sich schlussfolgern, dass diese drei Begriffe Fahrzeuge beschreiben. Im Folgenden werden des Öfteren die Begriffe Kindkonzept, beziehungsweise Kindbegriff und Elternkonzept, beziehungsweise Elternbegriff verwendet. Ein Kindkonzept/Kindbegriff ist ein Konzept, das mindestens ein übergeordnetes Konzept




**(a)** Aufbau einer einfachen Taxonomie, die in einer Baumstruktur vorliegt

**(b)** Aufbau einer einfachen Taxonomie, die als gerichteter azyklischer Graph vorliegt

**Abbildung 2.1.:** Aufbau einfacher Taxonomien

besitzt. Im Falle des Beispiels wären beispielsweise `PKW` und `LKW` Kindkonzepte von `Fahrzeug`. Ein Elternkonzept/Elternbegriff ist ein Konzept, welches mindestens ein untergeordnetes Konzept besitzt. Auf das Beispiel bezogen wäre das Konzept `Fahrzeug` das Elternkonzept von `PKW` und `LKW`. Durch die Eltern/Kind Beziehung zwischen den Konzepten ergibt sich für eine Taxonomie immer eine hierarchische Struktur. Im Allgemeinen liegt diese in Form einer Baumstruktur vor.

Taxonomien sind allerdings nicht immer reine Baumstrukturen. So ist es zum Beispiel möglich, dass ein Kindkonzept mehr als ein Elternkonzept besitzt. Hierdurch wird aus der Baumstruktur ein gerichteter azyklischer Graph. Eine Erweiterung des Beispiels von Abbildung 2.1a ist in Abbildung 2.1b zu sehen. In dem erweiterten Beispiel wurde zusätzlich der Begriff `Winterreifen` eingeführt. Nun können sowohl PKWs als auch LKWs Winterreifen besitzen (in diesem Beispiel wird davon ausgegangen, dass das Fahrrad keine Winterreifen besitzt). Um duplizierte Begriffe zu vermeiden, können einem Kindkonzept mehrere Elternkonzepte zugewiesen werden. Im Falle des Beispiels hat das Kindkonzept `Winterreifen` die zwei Elternkonzepte `PKW` und `LKW`.

### 2.1.2. Ontologien

Eine Ontologie weist die Merkmale einer Taxonomie auf, erweitert diese allerdings an diversen Stellen. Wie bei einer Taxonomie wird eine Ontologie durch eine Menge von Konzepten gebildet, die zueinander in Relation stehen. Hierdurch ergibt sich wie zuvor eine Baumstruktur, beziehungsweise ein gerichteter azyklischer Graph. Der Unterschied zu einer Taxonomie besteht darin, dass eine Ontologie zusätzliche Informationen enthält. Zudem macht nicht der genaue Wortlaut ein Konzept aus, sondern vielmehr dessen Bedeutung [CJB99][UG96].

Eine dieser zusätzlichen Informationen beschreibt wie verschiedene Konzepte in Relation zueinander stehen. Um dies zu verdeutlichen wurde in Abbildung 2.2 das vorherige Beispiel erweitert. Wie zu

sehen, wurden die Kanten des Graphen beschriftet um die Relation zwischen den einzelnen Konzepten genauer zu spezifizieren. So sieht man, dass ein `PKW` ein `Fahrzeug`, allerdings ein `Winterreifen` an einem `PKW` oder `LKW` montiert ist. Allgemein wird diese Zusatzinformation zu einer Relation als Attribut bezeichnet. Attribute sind nicht nur auf Relationen beschränkt, sondern können auch auf Konzepte angewendet werden.

Zudem ist es möglich, dass ein Konzept in mehreren Sprachen vorhanden ist. Um die Verwaltung dieser zu vereinfachen werden im Allgemeinen die Synonyme einer Sprache pro Konzept gruppiert. Die Konzepte in Abbildung 2.2 besitzen zusätzlich zu deren Bezeichnung, die Information in welcher diese vorliegt. `de` steht für Deutsch und `en` für Englisch. Das Konzept mit dem deutschen Synonym `LKW` hat zusätzlich ein englisches mit dem Wert `Truck`.

Die hier genannten Beispiele für zusätzliche Informationen, die eine Ontologie enthalten kann, erheben keinen Anspruch auf Vollständigkeit. Vielmehr kann je nachdem wie der Aufbau einer Ontologie definiert ist, diese verschiedenste Informationen enthalten. Im Folgenden werden die Begriffe Taxonomie und Ontologie synonym zueinander verwendet, da deren genaue Definition im Weiteren nicht relevant ist.

Ein Beispiel für eine große allgemeingültige Ontologie ist WordNet [MBF+90]. Dieses beinhaltet ausschließlich englischsprachige Begriffe und ist frei verfügbar. Strukturell ist es aus Synsets, einer unsortierten Gruppe von Synonymen, aufgebaut. So würden zum Beispiel die Begriffe `PKW` und `Auto` in einem Synset zusammengefasst werden. Eine ähnliche Ressource für die deutsche Sprache ist GermanNet [HF97]. Um den Aufwand des Erstellens und der Wartung zu verringern, wurde mit WikiNet [NS13] eine aus dem Kategoriebaum des Wikipedia Projekts automatisch generierte Ontologie vorgestellt. Zusätzlich wurde diese mithilfe von mit Textmustern durchsuchten Wikipedia Artikeln erweitert.

Viele Anwendungsfälle erfordern keine allgemeingültige, sondern eine speziell auf ein Anwendungsgebiet abgestimmte Taxonomie. Diese Art von Taxonomien werden oft manuell durch einen Experten erstellt [Sch11]. Auf diese Weise wurde die für diese Arbeit als Grundlage dienende multilinguale Taxonomie erstellt [ST08][ST10]. Diese wurde bereits für die Analyse deutscher Social Media Daten [Ban13], als auch für die Analyse englischer, von Mechanikern erstellten Fehlerberichten verwendet [HS10].

Ein Verwendungszweck von Taxonomien ist die Analyse von Texten auf deren Bedeutung hin. Durch Synonyme können wichtige Begriffe erkannt und über den hierarchischen Aufbau der Taxonomie können Zusammenhänge zwischen verschiedenen Synonymen hergestellt werden. Enthält ein Text die Begriffe `Winterreifen` und `PKW`, so kann ein System mit der in Abbildung 2.2 zu sehenden Taxonomie, darauf schließen, dass die `Winterreifen` an dem `PKW` montiert sind.

Um eine Taxonomie sinnvoll für die Textanalyse verwenden zu können, muss diese mit möglichst vielen Synonymen für jedes Konzept angereichert werden. Gerade diese Anreicherung von Konzepten mit möglichst vielen Variationen der Terminologie, ist ein bekanntes Problem, das gelöst werden muss [NS13][SBRZ10].

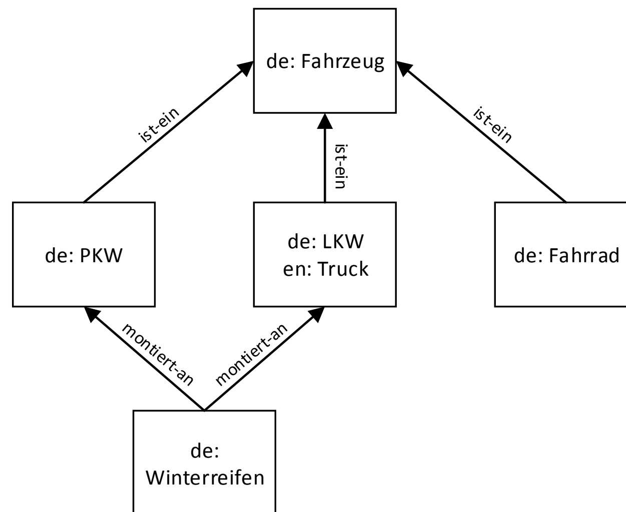



**Abbildung 2.2.:** Aufbau einer einfachen Ontologie

## 2.2. Visualisierung von Baumstrukturen auf Tablets

Ein wichtiger Punkt dieser Arbeit ist es eine Taxonomie auf Tablets zu präsentieren. [Chi06] und [YC06] beschäftigen sich damit, wie sich welcher Typ von Information auf einem mobilen Gerät anzeigen lässt. Hierbei wurde ermittelt, dass sich Informationen auf mobilen Geräten aufgrund von diversen Limitierungen, wie zum Beispiel der Größe des Bildschirms, nicht in gleicher Weise visualisieren lassen wie auf Desktop Computern. Aus diesem Grund müssen hierfür neue, für Tablets und Touch-Bedienung geeignete, Visualisierungen gefunden werden.

Taxonomien liegen im Allgemeinen als hierarchische Struktur vor (siehe Kapitel 4.2). Wenige Konzepte, der in [ST10] vorgestellten und als Ausgangspunkt für diese Arbeit dienenden Taxonomie, besitzen mehr als ein Elternkonzept. Aus diesem Grund wird der vorliegende azyklische gerichtete Graph als Baumstruktur visualisiert. Die folgenden Arbeiten beschäftigen sich mit der Visualisierung von Baumstrukturen für Tablets, beziehungsweise allgemein mobile Endgeräte.

[SPH11] stellt die Methode *Tablorer (Table Explorer)* vor, die explizit für die Verwendung auf Tablets ausgelegt ist. Diese versucht den vorhandenen Platz auf dem Bildschirm möglichst effizient durch eine tabellarische Anordnung der Daten zu nutzen. Durch empirisch durchgeführte Methoden, wurde eine um 22% reduzierte Zeit zum Suchen von Elementen innerhalb der Hierarchie ermittelt. Verglichen wurde Tablorer mit einer eingerückten Liste (siehe Kapitel 4.5.1), einer horizontalen Liste (siehe Kapitel 4.5.2) und einer in dieser Arbeit nicht näher betrachteten vertikalen Liste.

In [EBB+14] wird *Paisley Trees*, eine komplexe Visualisierung, vorgestellt. Deren Ziel es ist große hierarchische Strukturen auf übersichtliche Weise auf kompaktem Raum darzustellen. Zum Erkunden eines Baums muss der Benutzer weder Zooming noch Panning einsetzen.

[HZ], [HGZ+10] und [CZJ13] beschäftigen sich mit Radial Edgeless Trees (RELTs). Eine RELT wurde zur bildschirmfüllenden Visualisierung von hierarchischen Strukturen auf mobilen Geräten entwickelt. Durch Restriktionen dieser Visualisierung ist diese nur für kleine Baumstrukturen, wie zum Beispiel Menüs, geeignet.

Eine genaue Betrachtung verschiedener Visualisierungstechniken ist in Kapitel 4.5 zu finden.

## 2.3. Hierarchische Strukturen in relationalen Datenbanken

Zur Speicherung einer Taxonomie wird auf den Einsatz einer relationalen Datenbank gesetzt. Der größte Teil der vorliegenden Taxonomie besteht aus einer Baumstruktur. Wenige Elemente besitzen mehr als ein Elternelement, weshalb die Taxonomie als Ganzes als gerichteter azyklischer Graph vorliegt. Um einen effizienten Zugriff auf die Daten zu gewährleisten, wird auf die Verwendung einer transitiven Hülle gesetzt. [DLSW99] beschäftigt sich mit der Erstellung und Wartung einer transitiven Hülle in Graphen. [Erd08] hingegen beschäftigt sich speziell mit dem Erstellen und Warten einer transitiven Hülle für azyklische gerichtete Graphen.

# 3. Technologische Grundlagen

Dieses Kapitel beschreibt verschiedene Konzepte und Technologien, die zum weiteren Verständnis der Arbeit von Bedeutung sind. Es wird ein Überblick über das Thema *Representational State Transfer (REST)* und eine kurze Einführung in das JavaScript-Framework *AngularJS* gegeben.

## 3.1. Representational State Transfer (REST)

Um flexibel mit den verwendeten Technologien des Clients zu sein, soll ein Webservice zur Kommunikation zwischen einem Client und Server verwendet werden. Aufgrund der vergleichsweise einfachen Implementierung eines REST basierten Webservice und der damit erreichbaren losen Kopplung wurde diese Technologie gewählt.

Representational State Transfer (REST) ist ein von Roy Fielding in seiner Dissertation [Fie00] beschriebener Architekturstil zur zustandslosen Kommunikation zwischen einem Client und einem Server. Als Grundlage von REST dient das Hypertext Transfer Protocol (HTTP), dessen Verben und Uniform Resource Identifiers (URIs). HTTP-Verben sind grundlegende Funktionen, die vom HTTP-Protokoll angeboten werden. Ein URI stellt eine Möglichkeit dar, eine Ressource eindeutig zu identifizieren. Genauere Erläuterungen der Begriffe werden im späteren Verlauf des Kapitels gegeben.

Bei REST repräsentiert jeder URI eine Ressource, beziehungsweise eine Sammlung von Ressourcen. Durch die Kombination eines URI und einem HTTP-Verb lässt sich diese abfragen, modifizieren, löschen oder Instanzen dieser neu anlegen.

Wird die verwendete REST-Architektur auf das Wesentliche reduziert, so ergeben sich fünf verschiedene Punkte, die die Merkmale der Architektur zusammenfassen [Til15][RRD07]. Folgend werden diese näher betrachtet.

### 3.1.1. Eindeutige Identifikation von Ressourcen

Innerhalb der REST Architektur müssen einzelne Ressourcen eindeutig über IDs identifizierbar sein. Hierzu wird im Allgemeinen ein URI (Uniform Resource Identifier) verwendet. Ein URI ist eine von Menschen lesbare Zeichenkette zur eindeutigen Identifikation einer Ressource. Der Aufbau eines URI ist im RFC 3986 [BLFW+05] definiert und der wohl bekannteste Einsatzort für dieses Identifikationsschema dürfte die Verwendung im World Wide Web sein. Das Benutzen eines einheitlichen und standardisierten Schemas für die Ressourcenidentifizierung hat Vorteile. So muss ein Entwickler zum Beispiel kein neues Schema erarbeiten, sondern kann sich auf ein bekanntes und weltweit verbreitetes verlassen. Hierdurch ergibt sich wiederum der Vorteil, dass das Schema, sprich die URI,

weltweit verstanden wird. Somit lässt sich eine Fehlkommunikation und Missverständnisse zwischen verschiedenen Personen vermeiden.

Die REST Architektur sieht vor sowohl Mengen einer Ressource als auch einzelne Ressourcen über einen eindeutigen Identifikator ansprechbar zu machen. Folgendes Beispiel soll diesen Grundgedanken verdeutlichen:

Angenommen man bietet einen REST Service unter der Adresse `http://example.com` an. Über diesen Service möchte man verschiedene Produkte bereitstellen. Zuerst soll die Menge aller verfügbaren Produkte abzufragen sein. Hierzu wird ein REST Endpunkt implementiert, der über den URI `http://example.com/products` aufrufbar ist. Anschließend sollen Produkte auch einzeln abrufbar sein. Um dies zu bewerkstelligen wird der vorhandene URI jeweils um die ID eines Produkts erweitert. So ergibt sich für das Produkt mit der ID 1 die URI `http://example.com/products/1` und für das Produkt mit der ID 2 die URI `http://example.com/products/2`.

### 3.1.2. Hypermedia

Dieses Prinzip wird auch als *Hypermedia as the engine of application state* bezeichnet. Der Grundgedanke hier ist es, den Zustand einer Applikation in Form von Links an den Client zu übergeben. Über diese Links kann dieser an weitere Informationen zu einer bestimmten Ressource gelangen, ohne den genauen Aufbau der Applikation zu kennen. Das folgende Beispiel soll dieses Prinzip verdeutlichen:

Angenommen es gibt einen REST Service, der über `http://example.com/products` eine Liste von verfügbaren Produkten anbietet. Weiterhin wird angenommen, dass die Antwort der Anfrage im JSON-Format (JavaScript Object Notation) gesendet wird. So könnte sich die in Listing 3.1 zu sehende Antwort ergeben. Die Antwort besteht aus einem Array, welches zwei Produkte enthält. Jedes Produkt enthält die Eigenschaft `href`, über welche sich weitere Informationen zu einem einzelnen Produkt abfragen lassen.

```
1  [
2      {
3          "name": "Kleiner Ball",
4          "price": "5",
5          "href": "http://example.com/products/1"
6      },
7      {
8          "name": "Großer Ball",
9          "price": "10",
10         "href": "http://example.com/products/2"
11     }
12 ]
```

**Listing 3.1:** Antwort von `http://example.com/products` im JSON-Format mit Links

### 3.1.3. HTTP-Verben

Wie in Kapitel 3.1.1 beschrieben werden innerhalb der REST Architektur URIs verwendet, um Ressourcen eindeutig zu identifizieren. Nun ergibt sich allerdings das Problem, dass im Regelfall verschiedene

Operationen für eine bestimmte Ressource durchgeführt werden sollen. So wäre es zum Beispiel denkbar, dass man eine Ressource entweder abfragen, ändern oder löschen möchte. Um zu vermeiden, dass für jede dieser Operationen ein eigener URI erstellt muss, werden die sogenannten HTTP-Verben, auch als HTTP-Methoden bezeichnet, eingesetzt. Die Verben ermöglichen es für ein URI verschiedene Operationen durchzuführen.

In der HTTP/1.1 Spezifikation [Fie99] werden acht verschiedene Verben genannt, wobei in dieser Arbeit nur auf die vier in Bezug auf REST wichtigsten eingegangen wird.

#### GET

GET-Anfragen werden dazu verwendet um Ressourcen vom Server abzufragen. Hierbei wird durch die HTTP/1.1 Spezifikation sichergestellt, dass durch eine GET-Methode keine Ressource auf dem Server verändert wird. Zudem sind GET-Anfragen idempotent, was bedeutet, dass jeder Aufruf eines URI mit den gleichen Parameter immer das gleiche Ergebnis liefert.

#### POST

Die POST-Methode wird im REST Umfeld im Allgemeinen dazu verwendet eine neue Ressource anzulegen. Diese ist die einzige der vier vorgestellten Methoden, die nicht idempotent ist. Des Weiteren kann diese Methode dazu verwendet werden beliebige Verarbeitungsschritte innerhalb der Applikation anzustoßen.

#### PUT

Über die PUT-Methode wird innerhalb eines REST-Services im Allgemeinen eine schon vorhandene Ressource aktualisiert. Die Methode kann allerdings auch dazu verwendet werden eine neue Ressource zu erstellen, falls diese noch nicht vorhanden ist. Wird eine URI mit der PUT-Methode mehrfach mit den gleichen Daten aufgerufen, so ist das Endergebnis bei jedem Aufruf dasselbe. Aus diesem Grund ist die PUT-Methode idempotent.

#### DELETE

Wie sich aus dem Namen der Methode schlussfolgern lässt, wird die DELETE-Methode zum Löschen einer Ressource verwendet. Wie die GET- und PUT-Methode ist auch diese idempotent.

### 3.1.4. Repräsentationen

Die REST Architekturspezifikation sieht vor, dass eine Ressource verschiedene Repräsentationen besitzen kann. Hierbei kann der Client aus den verfügbaren Repräsentationen die für ihn passende auswählen. Bei Beispiel 3.1 wurde als Repräsentation der Daten zum Beispiel das JSON Format gewählt. Als eine weitere Mögliche Darstellungsform wäre auch eine Repräsentation im XML Format denkbar.

### 3.1.5. Zustandslos

Das Prinzip der Zustandslosigkeit beschreibt, dass kein Zustand für einen spezifischen Client über die Dauer einer Anfrage hinaus gehalten werden soll. Dies bedeutet nicht, dass es keine verschiedenen Zustände geben kann. Zustände sollen vielmehr vom Client gehalten oder vom Server in einen Ressourcenstatus umgewandelt werden, so wie dies in Kapitel 3.1.2 beschrieben wird.

## 3.2. AngularJS

Die Programmlogik von Web Applikationen wird im Allgemeinen in der Programmsprache JavaScript implementiert. Während der Entwicklung einer App treten gewisse Probleme, wie zum Beispiel das Binden von Daten an eine Ansicht, des Öfteren auf. Um diese nicht mit jedem Projekt erneut lösen zu müssen, wird im Allgemeinen ein Framework verwendet. Für das zu entwickelnde System soll AngularJS verwendet werden, da es ein mächtiges und flexibles Framework ist. Zudem existieren sehr viele Module von Drittanbietern, die den Funktionsumfang des Frameworks signifikant erhöhen.

AngularJS ist ein von der US-Amerikanischen Google Inc. seit 2009 entwickeltes Open-Source JavaScript Framework. Dieses Framework ermöglicht es sogenannte Single-Page Webanwendungen mit Hilfe des Model/View/ViewModel (MVVM) Architekturmusters zu erstellen [Goo15a].

### 3.2.1. Single-Page Webanwendung

*Single-Page Webanwendungen* sind Anwendungen, die in einem Browser, wie zum Beispiel Mozilla Firefox oder Google Chrome ablaufen. Als technische Grundlage wird im Allgemeinen HTML und JavaScript verwendet. Die Besonderheit bei Single-Page Webanwendungen liegt darin, dass beim Starten beziehungsweise beim Laden der Anwendung nur ein einzelner Seitenaufruf erfolgt. Die restlichen von der Anwendung benötigt Daten werden dynamisch und nur bei Bedarf mittels JavaScript über Ajax (Asynchronous JavaScript and XML) Aufrufe angefordert. Ajax ermöglicht es Daten asynchron vom Server abzufragen, ohne dass die Webseite, beziehungsweise Anwendung komplett neu geladen werden muss [Gar05]. Hierdurch kann die an den Server zu übertragende Datenmenge klein gehalten werden, wodurch wiederum das Ansprechverhalten und die Reaktionsfähigkeit der Anwendung bei Benutzereingaben verbessert werden kann.

### 3.2.2. Model/View/ViewModel

Das *Model/View/ViewModel (MVVM)* Architekturmuster basiert auf dem *Model/View/Controller* (MVC) Muster. Das MVC Architekturmuster besteht aus drei Komponenten [GD13]:

- Das *Model* enthält die Hauptlogik der Anwendung und ist für die Verarbeitung der Daten zuständig.
- Die *View* ist für die Darstellung der Benutzeroberfläche (GUI) verantwortlich. Sie erhält die darzustellenden Informationen vom Model.
- Der *Controller* nimmt Eingaben des Benutzers entgegen und interpretiert diese. Die interpretierten Eingaben werden an das Model übergeben, welche diese wiederum verarbeitet und der View zur Verfügung stellt.

Der Vorteil des MVC-Konzeptes ist es, dass die einzelnen Komponenten getrennt voneinander entwickelt werden können. So kann zum Beispiel die *View* von einem Designer und das *Model* und der *Controller* von einem Softwareentwickler geschrieben werden.

Eine Weiterentwicklung des MVC-Konzeptes ist das von John Gossman im Jahr 2005 in seinem Blog [Gos05] vorgestellte MVVM Architekturmuster. Ähnlich wie das MVC-Muster besteht auch das MVVM-Muster aus drei Komponenten:

- Das *Model* verhält sich analog zum Model beim MVC-Konzept. Es enthält die Hauptlogik der Anwendung und ist für die Verarbeitung der Daten zuständig.
- Die *View* enthält die darzustellenden Informationen und ist für die Darstellung der GUI verantwortlich. Die View ist über sogenanntes *Data Binding* direkt mit dem Model verbunden. Hierbei muss zwischen *One-Way-Databinding (OWD)* und *Two-Way-Databinding (TWD)* unterschieden werden. Beim OWD bindet sich die View an eine oder mehrere Variablen des Models. Jedes mal, wenn sich eine der Variablen im Model ändert, wird diese Änderung direkt in der View übernommen. Änderungen die in der View über Kontrollelemente, wie zum Beispiel Textfelder, gemacht werden, werden nicht an das Model übertragen. Das TWD verhält sich ähnlich wie OWD, mit dem Zusatz das über Kontrollelemente in der View gemachte Änderungen auch im Model übernommen werden.
- Das *ViewModel* ersetzt den Controller des MVC-Architekturmusters und besitzt mehrere Aufgaben.
    1. Es ist dafür zuständig den Zustand der View zu halten, wie zum Beispiel Selektionen oder den aktuellen Modus der Benutzeroberfläche
    2. In Anwendungen kann es vorkommen, das man nicht alle Variablen direkt an die View binden kann, da zum Beispiel in der View und dem Model verschiedene Datentypen verwendet werden. An dieser Stelle ist das ViewModel dafür verantwortlich die Umwandlung der Datentypen so vorzunehmen, dass die View und das Model die Daten verarbeiten können
    3. Enthält Kommandos/Funktionen, über welche die View mit dem Model interagieren kann

### 3.2.3. Besonderheiten des Frameworks

Im Folgenden werden die wichtigsten Besonderheiten die das AngularJS-Framework bietet vorgestellt.

#### Module

AngularJS Anwendungen werden in sogenannte Module aufgeteilt. Module sind gekapselt und besitzen keinen globalen Zustand, weswegen es möglich ist auf einer einzigen Seite mehrere Anwendungen zu betreiben. Im Normalfall wird ein Hauptmodul definiert, das Abhängigkeiten auf andere Module besitzt.

Module werden in JavaScript-Dateien definiert (siehe Listing 3.2) und können anschließend in HTML-Dateien verwendet werden (siehe Listing 3.3).

```
1  angular.module('taxonomy', ['ngTouch']);
```

**Listing 3.2:** Definition des Modules *taxonomy* in einer JavaScript-Datei, das eine Abhängigkeit auf das Modul *ngTouch* hat

```
1  <html ng-app="taxonomy"><!-- Markup --></html>
```

**Listing 3.3:** Verwendung des AngularJS Moduls *taxonomy* in einer HTML-Datei

#### Controller

Nachdem ein Hauptmodul definiert ist, können Controller verwendet werden um bestimmte Bereiche der Anwendung zu kontrollieren. Controller enthalten die Logik der Anwendung und entsprechen somit dem Model des in Kapitel 3.2.2 vorgestellten MVVM-Architekturmusters. Listing 3.4 zeigt die Definition des Controllers `taxonomyCtrl` der zum Modul `taxonomy` gehört. Über Dependency-Injection (DI) können dem Controller verschiedene Services bereitgestellt werden. DI bedeutet, dass ein Entwickler nicht genau definieren muss wie eine Abhängigkeit aufzulösen ist. Diese wird automatisiert aufgelöst, indem eine passende Ressource gesucht und zur Verfügung gestellt wird.

```
1  angular.module('taxonomy').controller('taxonomyCtrl', function($scope) {
2        $scope.name = 'Yoda';
3  });
```

**Listing 3.4:** Definition des Controllers `taxonomyCtrl` in einer JavaScript-Datei

In Listing 3.4 wird dem Controller das Objekt `$scope` übergeben, welches eine besondere Stellung innerhalb einer AngularJS Anwendungen einnimmt. Dieses ist das Bindeglied zwischen dem Controller und der View. In diesem Objekt können als Eigenschaften Funktionen, Variablen und weitere Objekte angelegt werden, die hierdurch in der View zur Verfügung stehen. In Listing 3.4 wird zum Beispiel die Eigenschaft `name` erzeugt und mit dem Wert *Yoda* befüllt. Listing 3.5 zeigt die Verwendung des Controllers und der Variable `name`, welche im `$scope`-Objekt definiert wurde.

```
<html ng-app="taxonomy">
<body ng-controller="taxonomyCtrl">
      Name: {{ name }}
</body>
</html>
```

**Listing 3.5:** Verwendung des Controllers `taxonomyCtrl` und der in `$scope` definierten Variable `name` innerhalb einer HTML-Datei

### Templates

In AngularJS werden sogenannte *Templates* verwendet um verschiedene Views zu erzeugen. In Kapitel 3.2.1 wurde beschrieben, das AngularJS Anwendungen Single-Page Applikationen sind. Hierdurch wird zu Beginn die Anwendung initial mit einer View geladen. Um beim Wechseln der View nicht die komplette Anwendung neu laden zu müssen, kommen Templates zum Einsatz. Templates enthalten HTML/AngularJS Markup-Fragmente, die bei Bedarf asynchron vom Server angefordert und an eine vorher definierte Stelle in der Haupt-HTML-Datei eingefügt werden. Über sogenannte Router wird innerhalb der Anwendung entschieden, welche Views zu welchem Zeitpunkt angezeigt werden. Durch den modularen Aufbau des AngularJS-Frameworks, kann die Implementierung eines Routers frei gewählt werden.

Im Folgenden wird der Router *AngularUI Router* [Ang] vorgestellt und verwendet. Dieser ist ein von der Google Inc. unabhängiges Projekt. Er hat es zum Ziel den Standard-Router zu ersetzen und zu erweitern. Listing 3.6 zeigt eine HTML-Datei, in welcher der Router verwendet wird. Zu beachten ist der `div-Tag` mit dem Attribut `ui-view`. Das Attribut `ui-view` markiert die Stelle, an welcher die Templates zur Laufzeit eingefügt werden. Damit die Anwendung weiß, wann welches Template verwendet werden soll, muss dies in einer JavaScript-Datei definiert werden. Eine beispielhafte Implementierung zeigt Listing 3.7.

Angenommen der Server läuft lokal und ist über `localhost` erreichbar, so wird beim Aufrufen der URL `http://localhost/#/home` das Template `home.html` geladen und beim Aufrufen von `http://localhost/#/taxonomy` das Template `taxonomy.html`. Alle unbekannten URLs werden zu `/home` umgeleitet, wodurch das Template `home.html` verwendet wird. Die geladenen Templates werden innerhalb des `div`-Elements mit dem Attribut `ui-view` aus Listing 3.6 eingefügt.

```
<html ng-app="taxonomy">
<body ng-controller="taxonomyCtrl">
      <div ui-view></div>
</body>
</html>
```

**Listing 3.6:** Definition der Stelle, bei welcher Templates eingefügt werden sollen. Diese wird durch das Attribut `ui-view` in einer HTML-Datei markiert

```
angular.module('taxonomy').config(function($stateProvider, $urlRouterProvider) {
// Leite alle nicht bekannten URLs weiter zu /home
$urlRouterProvider.otherwise("/home");
```

```
4
5   // Definition der Zustände 'home' und 'taxonomy'
6   $stateProvider
7       .state('home', {
8           url: "/home",
9           templateUrl: "home.html"
10      })
11      .state('taxonomy', {
12          url: "/taxonomy",
13          templateUrl: "taxonomy.html"
14      });
15  });
```

**Listing 3.7:** Definition der Routen des *ui-router* in einer JavaScript-Datei

### Direktiven

Direktiven erlauben es nicht durch die HTML-Spezifikation abgedeckten HTML-Elemente oder Attribute mit Funktionalität zu versehen. Durch AngularJS werden viele vorgefertigte Direktiven mitgeliefert, wobei zusätzlich eigene erstellt werden können. Diese lassen sich anschließend frei in HTML-Dokumenten verwenden.

Ein Beispiel einer Direktive ist in Listing 3.8 zu sehen. Dort wird bei dem HTML spezifischen Element `div` das HTML unspezifische Attribut `ui-view` verwendet. `ui-view` ist hierbei eine Direktive, welche im *ui-router* Modul definiert ist. Direktiven müssen nicht zwangsweise als Attribute verwendet werden. Bei der Erstellung lässt sich festlegen, ob diese entweder als Element oder Attribut verwendet werden können. So kann die Direktive, wie in Listing 3.8 zu sehen, auch direkt als HTML-Element verwendet werden.

```
1   <html ng-app="taxonomy">
2   <body ng-controller="taxonomyCtrl">
3       <ui-view></ui-view>
4   </body>
5   </html>
```

**Listing 3.8:** Definition der Stelle, bei welcher Templates eingefügt werden sollen. Diese wird durch das Element `ui-view` in einer HTML-Datei markiert

### Testen

Das Testen einer Anwendung ist unumstritten ein wichtiger Punkt der Softwareentwicklung. Um dies zu vereinfachen wird zum Testen einer AngularJS Anwendung im Allgemeinen eine Kombination des Kommandozeilenprogramms *Karma* und des Testframeworks *Jasmine* verwendet [Goo15b].

Karma ist eine auf NodeJS[1] basierenden Anwendung, die dafür zuständig ist einen Webserver zu starten, den Programmcode zu laden und die eigentlichen Tests über den neu gestarteten Webserver

[1]Node.js ist eine Plattform, die es erlaubt serverseitig JavaScript Code auszuführen.

auszuführen. Die Test selber können in verschiedenen Browser wie zum Beispiel Mozilla Firefox oder Google Chrome ausgeführt werden. Hierdurch lässt sich die Kompatibilität der Anwendung browserübergreifend testen.

Jasmine ist ein Unit-Test Framework für JavaScript, das insbesondere auf das Testen einer AngularJS Anwendung ausgelegt ist. Das Ziel des Frameworks ist es die Tests in strukturierten und selbst dokumentierenden Form vorliegen zu haben.

Dadurch, dass AngularJS regen Gebrauch von Dependency Injection macht, wird das Testen maßgeblich vereinfacht. Dies liegt daran, dass eine injizierte Abhängigkeit nicht vollständig entwickelt sein muss. Zum Testen können hinter solch einer Abhängigkeit auch Mockups stehen, die die Funktionalität einer Abhängigkeit ausschließlich simulieren.

# 4. Analyse

In diesem Kapitel werden zuerst die Anforderungen an die zu entwickelnde App erörtert. Anschließend findet eine Analyse der für die Arbeit relevanten und schon vorhandenen Ressourcen statt. Hierzu gehören die vorhandene Taxonomie und der dazugehörige Taxonomieeditor. Darauf folgend werden verschiedene Möglichkeiten untersucht eine Baumstruktur beziehungsweise einen gerichteten azyklischen Graphen zu visualisieren.

## 4.1. Anforderungen

Im Rahmen der Arbeit soll eine App für mobile Endgeräte entwickelt werden, die es erlaubt eine vorhandene Taxonomie zu verwalten und zu erweitern. Hieraus ergeben sich die folgenden Anforderungen:

- Die App soll für mobile Endgeräte optimiert sein, hierbei insbesondere für Tablets. Als Tablet werden Geräte mit einer Bildschirmdiagonale von sieben Zoll oder größer angesehen.
- Die technische Plattform für die App ist nicht festgelegt. Es ist somit möglich entweder eine native, eine Web-App oder eine hypride App zu erstellen. Aus diesem Grund gibt es keine Einschränkung der zu verwendenden Technologie, wie zum Beispiel bestimmte Frameworks.
- Die vorhandene Taxonomie soll auf dem Endgerät dargestellt werden können und erkundbar sein. Erkundbar bedeutet in diesem Zusammenhang, dass man bestimmte Teile des Taxonomiebaums genauer betrachten kann, um Details zu verschiedenen Konzepten zu erfahren. Ein Detail ist zum Beispiel ein Synset mit seinen Synonymen und Tokens. Des Weiteren bedeutet erkundbar, dass man durch den Taxonomiebaum navigieren kann.
- Die Taxonomie soll durchsuchbar sein. Man soll sowohl nach einem Namen als auch nach der ID eines Konzepts suchen können.
- Die Taxonomie soll editierbar sein. Das heißt, dass man schon vorhandene Konzepte, Synsets, Synonyme, Token und Attribute ändern oder löschen können soll. Des Weiteren soll es auch möglich sein neue Konzepte, Synsets, Synonyme, Token und Attribute hinzuzufügen.
- Die App soll es ermöglichen Vorschläge für neue Synonyme bewerten und annehmen oder ablehnen zu können. Neue Vorschläge werden außerhalb des Systems generiert und diesem zur Verfügung gestellt. Ein Vorschlag hat einen *Score*, der angibt wie wahrscheinlich es ist, dass dies ein zu dem Konzept passender Vorschlag ist. Er liegt in einem Bereich zwischen 0 und 100, wobei ein höherer Score auf einen besseren Vorschlag hinweist. Einem Experten muss es nun möglich sein diesen Vorschlag anzunehmen oder abzulehnen. Wird er angenommen, soll das

Synonym in die Taxonomie übernommen werden. Wird der Vorschlag abgelehnt, so soll es verworfen werden.

- Innerhalb der Taxonomie soll nichts „wirklich“ gelöscht werden. Dies bedeutet, dass gelöschte Dinge nur als gelöscht markiert werden und somit innerhalb der App nicht mehr angezeigt werden.
- Eine Taxonomie soll abgesichert sein, sodass ein Benutzer sich authentifizieren muss um mit dieser interagieren zu können.
- Das System soll es ermöglichen mehrere Benutzer anzulegen, damit diese mit der App arbeiten können.
- Die App soll es erlauben mehrere Taxonomien zu verwalten, die unabhängig voneinander sind.
- Es soll möglich sein, dass mehrere Benutzer über verschiedene Geräte gleichzeitig mit einer Taxonomie arbeiten können.

## 4.2. Vorhandene Taxonomie

Im Rahmen einer vorhergehenden Forschungsarbeit [ST10] wurde die Taxonomie erarbeitet, für welche eine App zu deren Verwaltung erstellt werden soll. Im weiteren Verlauf der Arbeit wird nicht auf den inhaltlichen Aufbau der Taxonomie eingegangen, da dieser für diese Arbeit nicht relevant ist. Von Relevanz ist ausschließlich der strukturelle Aufbau dieser.

Folgend wird der strukturelle Aufbau derselben erläutert. Eine graphische Visualisierung der Struktur der Taxonomie ist in Abbildung 4.1 zu sehen.

Die Taxonomie ist aus Konzepten aufgebaut, die in Relation zueinander stehen. Ein Konzept kann mehrere Elternkonzepte besitzen, allerdings dürfen keine Kreise innerhalb des Graphen entstehen. Hierdurch kann sich unter Umständen ein gerichteter azyklischer Graph ergeben. Hat jedes Konzept immer nur ein Elternelement, so liegt die Taxonomie in einer Baumstruktur vor.

Ein Konzept beinhaltet Synonyme in einer oder mehreren Sprachen. Ein Konzept kann beliebig viele Synonyme enthalten, muss allerdings mindestens eines besitzen. Die Synonyme eines Konzepts werden nach der Sprache des Synonyms gruppiert.

Eine Gruppe von Synonyme in einer bestimmten Sprache wird als Synset bezeichnet. Der Begriff Synset kommt aus dem Englischen und bezeichnet eine Menge von Synonymen (Set of Synonyms). Ein Synonym kann aus beliebig vielen Wörtern zusammengesetzt sein, wobei ein einzelnes Wort eines Synonyms als Token bezeichnet wird.

Sowohl Konzepte, als auch Synsets, Synonyme und Tokens können Attribute besitzen. Diese Art der Aufteilung der semantischen Ressource lehnt sich an die Aufbau der Ontologie *WordNet* [MBF+90] an. In der vorliegenden Taxonomie wird von den Attributen allerdings sehr selten Gebrauch gemacht.

Die gegebene Taxonomie liegt in zwei verschiedenen XML-Dateien vor die mit UTF-16 BE (Big Endian) kodiert sind. Die Dateien besitzen alle die Dateiendung `.tax`.

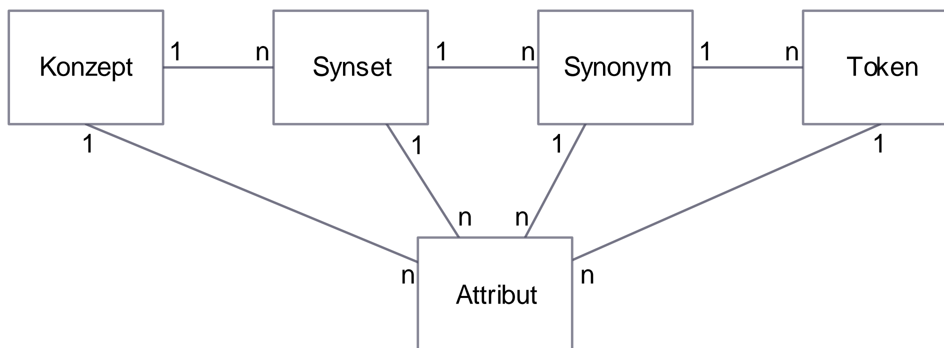



**Abbildung 4.1.:** Der strukturelle Aufbau der gegebenen Taxonomie. Ein Konzept besteht aus Synsets, welches wiederum aus Synonymen besteht. Ein Synonym setzt sich aus Tokens zusammen. Konzepte, Synsets, Synonyme und Tokens können beliebig viele Attribute besitzen.

Zur Verfügung stehen zusätzliche weitere Taxonomien, die im selben Format vorliegen allerdings für die Arbeit nicht relevant sind. Aus den verschiedenen zur Verfügung stehenden Taxonomien ist erkenntlich, dass es ein festes Namensschema für die beiden Dateien pro Taxonomie gibt. Folgend wird näher auf die einzelnen Dateien eingegangen.

### 4.2.1. `*_data.tax`

Dateien mit der Endung `_data.tax` enthalten die Daten der Taxonomie. Zum einfacheren Verständnis des Aufbaus dieser Datei wurde das in Listing 4.1 zu sehende Beispiel erstellt. Im Folgenden soll der allgemeine Aufbau der Datei anhand dieses Beispiels erläutert werden.

In Zeile 1 wird der XML-Kopf definiert. Anzumerken ist hier, dass statt der üblich verwendeten *UTF-8* Kodierung eine *UTF-16* Kodierung verwendet wird. Zeile 2 leitet den Beginn der Taxonomie ein. Zudem wird über das Attribut *root* die ID des Konzeptes angegeben, welches als Wurzel-Element des Taxonomiebaums verwendet wird. Im Falle des Beispiels ist das Konzept mit der ID 1 das Wurzelkonzept. Anschließend werden in den Zeilen 3-6 die in der Taxonomie verwendeten Sprachen spezifiziert.

Zwischen den Zeilen 7-27 befindet sich der eigentliche Inhalt der Taxonomie. Die Beispieltaxonomie enthält zwei verschiedene Konzepte. Wie zuvor erwähnt besteht ein Konzept der gegebenen Taxonomie aus sogenannten Synsets. Ein Konzept kann zwar prinzipiell beliebig viele Synsets enthalten, allerdings wurde in dem Beispiel pro Konzept nur ein Synset verwendet, um dieses möglichst klein zu halten.

Zeile 10-13 und 21-23 definieren die Synonyme eines Synsets. Auch hier können pro Synset beliebig viele Synonyme existieren, allerdings wurde auch hier zugunsten eines übersichtlichen Beispiels nur

ein Synonym pro Synset verwendet. Ein Synonym besitzt das Attribut `name`, welches den Inhalt aller Tokens in einer Zeichenkette widerspiegelt.

Die Zeilen 12, 13 und 22 enthalten die Tokens der Synonyme. Pro Wort eines Synonyms wird ein Token verwendet. So besteht das Synonym mit der ID 1 aus den zwei Wörtern *PKW* und *Reifen*, weshalb hier zwei verschiedene Tokens verwendet werden.

Konzepte, Synsets, Synonyme und Tokens können Attribute besitzen. Da diese in der vorliegenden Taxonomie sehr selten verwendet werden, wurde im Beispiel darauf verzichtet. Jedes der genannten Elemente besitzt das XML-Attribut `deleted`. Wird eines dieser Elemente gelöscht, wird dieses nicht aus der Datei entfernt. Stattdessen wird der Inhalt des Attributs von `false` auf `true` gesetzt.

```
1  <?xml version="1.0" encoding="utf-16"?>
2  <taxonomydata root="1">
3        <languages>
4              <language name="de" />
5              <language name="en" />
6        </languages>
7        <concepts>
8              <concept deleted="false" id="1">
9                    <synsets>
10                         <synset deleted="false" id="1" language="de">
11                               <mwu deleted="false" id="1" name="PKW Reifen">
12                                     <token content="PKW" deleted="false" id="1"/>
13                                     <token content="Reifen" deleted="false" id="2"/>
14                               </mwu>
15                         </synset>
16                   </synsets>
17             </concept>
18             <concept deleted="false" id="2">
19                   <synsets>
20                         <synset deleted="false" id="2" language="en">
21                               <mwu deleted="false" id="2" name="Valve">
22                                     <token content="Valve" deleted="false" id="3"/>
23                               </mwu>
24                         </synset>
25                   </synsets>
26             </concept>
27       </concepts>
28 </taxonomydata>
```

**Listing 4.1:** Daten einer Beispieltaxonomie, die zwei verschiedene Konzepte enthält. Beim ersten Konzept existiert nur ein deutsches Synonym und beim zweiten Konzept nur ein englisches

### 4.2.2. `*_forschung.tax`

Dateien mit der Endung `_forschung.tax` enthalten die Relationen zwischen den Konzepte, die in der `*_data.tax`-Datei beschrieben sind. Zudem wird in dieser Datei die Standardsprache der Taxonomie definiert. Zum besseren Verständnis wurde in Listing 4.2 ein minimales Beispiel erstellt, das sich auf die in Listing 4.1 gezeigten Konzepte bezieht.

In der ersten Zeile des Beispiels wird der XML-Kopf definiert. Dieser Kopf ist identisch zu dem Kapitel 4.2.1 vorgestellten Kopf. Somit wird auch in der `*_forschung.tax`-Datei *UTF-16* als Kodierung eingesetzt.

In Zeile 2 beginnt der eigentliche Inhalt der Datei. Mit dem Attribut `id` wird der Name der Taxonomie definiert. In der darauf folgenden Zeile wird die Standardsprache der Taxonomie festgelegt.

Ab Zeile 3 werden die Relationen zwischen den Konzepten der Taxonomie definiert. In dem Beispiel wird eine Relation zwischen dem Konzept mit der ID 1 und 2 erstellt. Das Konzept mit der ID 1 ist somit das Elternkonzept für das Konzept mit der ID 2.

Des Weiteren können Relationen beliebig viele Attribute enthalten, es müssen allerdings keine enthalten sein. Bei dem gezeigten Beispiel besitzt die Relation ein Attribut `type`, das die Beziehung zwischen den Konzepten genauer definiert. So hat das Konzept mit der ID 2 eine *ist-ein-Teil-von* Beziehung zu dem Konzept mit der ID 1. Verwendet man anstatt den IDs der Konzepte, die Synonyme, so sieht man, dass ein *Ventil* ein Teil eines *PKW Reifens* ist.

```
1  <?xml version="1.0" encoding="utf-16"?>
2  <taxonomyprofile id="test_taxonomy">
3        <defaultlanguage language="de" />
4        <relations>
5              <relation deleted="false" id="1" from="2" to="1">
6                    <attribute deleted="false" id="1" name="type">
7                          <value value="meronym" />
8                    </attribute>
9              </relation>
10       </relations>
11 </taxonomyprofile>
```

**Listing 4.2:** Relationen zwischen den in der `*_data.tax`-Datei definierten Konzepte

## 4.3. Vorhandener Taxonomieeditor

Zur Verwaltung des in Kapitel 4.2 vorgestellten Taxonomieformats existiert ein Editor. Folgend werden die Eigenschaften und Schwächen des Editors näher betrachtet

### 4.3.1. Eigenschaften des Editors

Der Editor ist in Java geschrieben und erlaubt das Laden einer Taxonomie. Hierzu muss über den Öffnen-Dialog die `*_forschung.tax` ausgewählt werden. Damit die Taxonomie erfolgreich geladen werden kann, muss sich eine Datei mit dem Suffix `_data.tax` und dem gleichen Präfix wie bei der gewählten Datei in selben Ordner befinden. Wird beispielsweise eine die Datei `automotive_foschung.tax` geladen, so muss auch die Datei `automotive_data.tax` in demselben Ordner vorhanden sein. Ein Mockup des Editors mit einer geladenen Taxonomie ist in Abbildung 4.2 zu sehen.

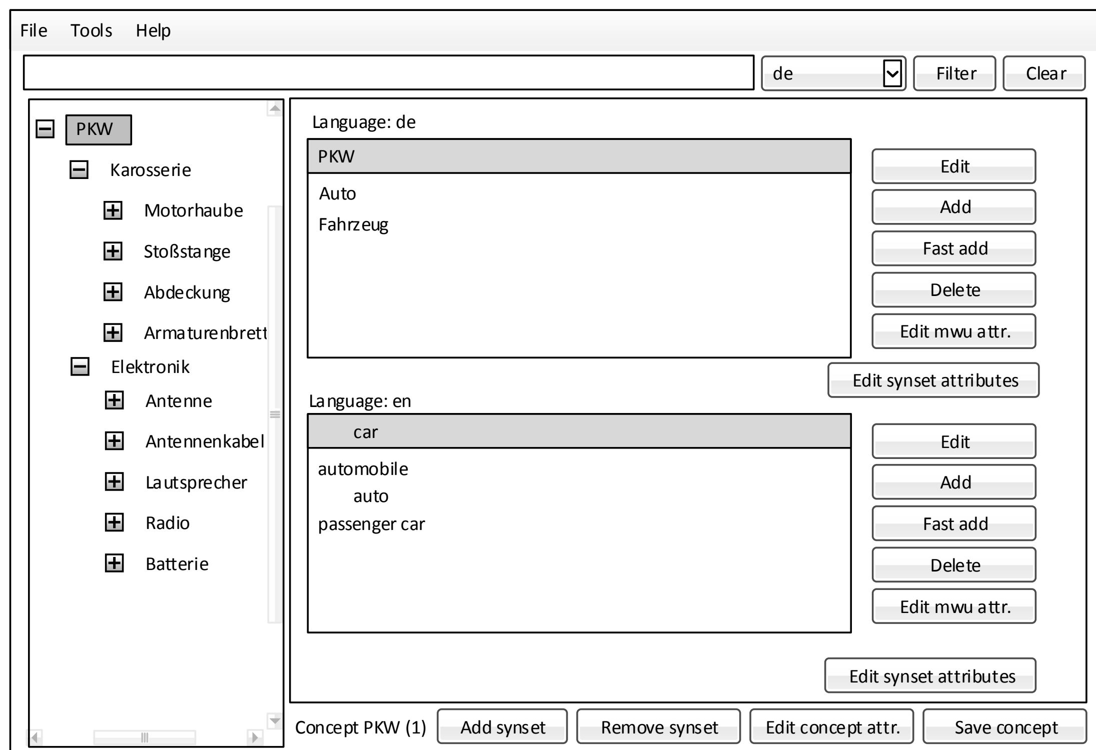



**Abbildung 4.2.:** Mockup des schon vorhandenen Editors zum Bearbeiten der in Kapitel 4.2 vorgestellten Taxonomie

Nachdem eine Taxonomie geladen ist, wird auf der linken Seite der Taxonomiebaum angezeigt. Der Großteil der Oberfläche ist für die rechts befindliche Ansicht der Details eines Konzepts reserviert. Die Inhalte der Detailansicht lassen sich über das Auswählen eines Konzepts im Taxonomiebaum bestimmen.

Zu den Details zählen die Synsets und Synonyme. Alle Synsets eines Konzepts werden untereinander aufgelistet und jedes Synset enthält eine Liste mit dessen Synonymen. Über diverse Buttons neben einem Synset lassen sich neue Synonyme hinzufügen oder vorhandene ändern und löschen. Zudem können die Attribute eines Synsets editiert werden. Jeder dieser Buttons öffnete weitere Fenster/Dialoge in welchen die jeweilige Aktion durchgeführt werden kann.

In der Fußleiste der Detailansicht befindet sich von links nach rechts

- der Anzeigename des ausgewählten Konzepts
- in einer Klammer die ID des gewählten Konzepts
- eine Möglichkeit neue Synsets hinzuzufügen
- ein Button zum Löschen eines schon vorhandenen Synsets

- eine Möglichkeit die Attribute eines Konzepts zu verändern
- ein Button zum Speichern der geänderten Daten.

Wird ein neues Konzept im Taxonomiebaum angeklickt, obwohl Änderungen an dem aktuell gewählten Konzept durchgeführt wurden, so weist der Taxonomieeditor hierauf hin und bietet die Speicherung des geänderten Konzepts an.

Über dem Taxonomiebaum und der Detailansicht befindet sich eine Such- beziehungsweise Filterfunktion. In das Textfeld kann der Name oder Teile des Namens eines Synonyms eingegeben werden und über die Combobox lässt sich die zu durchsuchende Sprache wählen. Nach einem Klick auf den Button *Filter* zeigt der Taxonomiebaum nur noch die Konzepte an, deren Synonyme den Suchbegriffe enthalten.

Konzepte können innerhalb des Baums verschoben werden. Hierzu gibt es zwei verschiedene Möglichkeiten. Zum einen kann ein Konzept über Drag'n'Drop direkt innerhalb des Taxonomiebaums an die gewünschte Stelle gezogen werden. Zum anderen lässt sich über das Menü *Tools* nach der Auswahl zweier Konzepte im Taxonomiebaum eine Relation zwischen diesen erstellen. Auf diese Weise ist es möglich einem Konzept mehr als zwei Elternkonzepte zuzuweisen.

### 4.3.2. Schwächen des Editors

Während der Analyse sind einige Probleme, beziehungsweise Unzulänglichkeiten des Editor gefunden worden. Die wichtigsten in Bezug auf diese Arbeit, werden im Folgenden vorgestellt.

Wie zuvor erwähnt ist es möglich, dass ein Konzept der Taxonomie mehr als ein Elternkonzept besitzt. Eine Analyse des Editors hat ergeben, dass ein Konzept immer nur an einer Stelle des Baums angezeigt werden kann. Dieses Problem soll anhand des in Abbildung 4.3 zu sehenden Beispiels genauer erläutert werden.

In diesem Beispiel gibt es das Konzept *Ventil*, das zwei Elternkonzepte besitzt: *PKW Reifen* und *LKW Reifen*. Hierdurch ist die Beispieltaxonomie kein Baum mehr, sondern ein gerichteter azyklischer Graph. Der Editor hat nun das Problem, dass er das Konzept *Ventil* nur unterhalb eines Elternkonzepts anzeigt. Zu erwarten wäre, dass das Konzept unterhalb beider Elternkonzepte angezeigt wird.

Ein weiteres Problem ist, dass immer nur nach einem Synonym in einer Sprache gleichzeitig gesucht werden kann. Möchte man testen, ob ein Synonym mit der gleichen Bezeichnung in mehreren Sprachen vorhanden ist, so müssen verschiedene Suchvorgänge gestartet werden.

Eine weitere Unzulänglichkeit der Suche ist die Einschränkung auf Synonymnamen. Ist dem Benutzer ausschließlich die ID eines Konzepts bekannt, so kann diese nur durch manuelles durchsuchen des Taxonomiebaums gefunden werden. Über die Suchfunktion kann nicht nach diesen gesucht werden.

Eine Unzulänglichkeit des Editors ist es, dass zum Bearbeiten von Synsets, Synonymen, Tokens und Attributen jeweils mindestens ein extra Fenster geöffnet wird. Hierbei kann es sehr schnell passieren, dass man die Übersicht verliert.

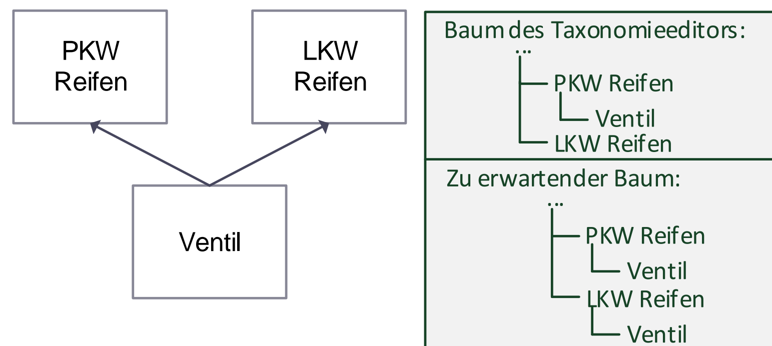



**Abbildung 4.3.:** Visualisierung des Problems, das der Taxonomieeditor bei Konzepten mit mehr als einem Elternkonzept besitzt

Der Editor wurde als Desktop-Applikation entwickelt. Aus diesem Grund ist dessen Benutzung auf die PC-Plattform eingeschränkt. Mit modernen Geräten wie Smartphones oder Tablets gibt es keine Möglichkeit diesen zu benutzen.

Zum Öffnen einer Taxonomie, muss diese lokal vorliegen. Beim Laden wird diese deserialisiert und komplett in den Hauptspeicher des Computers geladen. Benötigt eine Taxonomie bedingt durch ihre Größe mehr Speicher als der Computer bereitstellen kann, so kann diese nicht bearbeitet werden. Der verfügbare Hauptspeicher limitiert somit die Größe einer Taxonomie.

## 4.4. Technologieanalyse

Bei dieser Arbeit existieren keine Vorgaben zur zu verwendenden Technologie. Aus diesem Grund findet an dieser Stelle eine Analyse der möglichen Arten von Apps für Tablets statt. Zuerst werden die Vor- und Nachteile einer nativen App betrachtet, darauf folgend werden sogenannte hybride Apps und zum Schluss die sogenannten Web-App analysiert. Im Folgenden werden nur die Punkte betrachtet, die von Relevanz für die Arbeit sind. Ausgeschlossen hiervon sind zum Beispiel die Möglichkeiten einer Monetarisierung einer App mit den verschiedenen Technologien.

### 4.4.1. Native App

Eine native App wird direkt für eine Zielplattform entwickelt, weshalb diese auch nur auf dieser verfügbar ist. Soll die App für andere Zielplattformen zur Verfügung stehen, muss diese im Normalfall in einer anderen Programmiersprache neu geschrieben werden. So wird im Android Umfeld im Allgemeinen die Sprache Java und im iOS Umfeld Objective C beziehungsweise Swift verwendet. Im Folgenden werden die spezifische Funktionen und Besonderheiten einzelner Plattformen nicht weiter betrachtet. Es werden ausschließliche für native Apps allgemeingültige Prinzipien veranschaulicht.

Native Apps haben den Vorteil, dass diese perfekt an die Zielplattform angepasst werden können. Hierdurch kann die Leistung der zugrundeliegenden Hardware bestmöglich ausgenutzt werden. Nicht nur leistungsmäßig, sondern auch visuell ist eine Anpassung an das zugrunde liegende System mit verhältnismäßig wenig Aufwand möglich. So bieten die Frameworks der einzelnen Plattformen im Regelfall vorgefertigte Komponenten wie zum Beispiel Buttons oder Listen an, die sich visuell perfekt in das Gesamtsystem einfügen.

Ein weiterer Vorteil nativer Apps ist es, dass alle Funktionen der zugrunde liegenden Hardware genutzt werden können. Insbesondere moderne Smartphones und Tablets besitzen eine Vielzahl von Sensoren, die im Allgemeinen auch in einer nativen App verwendet werden können.

Dadurch, dass eine native App auf dem Zielgerät installiert werden muss, befinden sich alle für die Ausführung benötigten Daten im Speicher des Gerätes. Hierdurch ist im Allgemeinen eine offline Nutzung der App möglich. Ausgenommen hiervon sind Apps, die als zentralen Bestandteil Informationen von einem Server anfordern um mit diesen zu arbeiten.

Den Vorteilen gegenüber steht der signifikante Nachteil der Plattformgebundenheit. Wie zuvor erwähnt wird eine native App spezifisch für eine Plattform, wie zum Beispiel Android oder iOS entwickelt. Soll die App auch auf anderen Plattformen verfügbar sein, muss diese im Allgemeinen neu implementiert werden.

Ein weiterer Nachteil einer nativen App ist während der Entwicklungszeit zu finden. Um geänderten Code zu testen, muss der Code nach dem kompilieren entweder auf einem realen Gerät oder einem Emulator der Zielplattform installiert werden. Zwar ist dieser Prozess bei den meisten Plattformen voll automatisiert, benötigt im Allgemeinen allerdings eine gewisse Zeit zur Durchführung. Aus diesem Grund kann der Entwicklungsprozess unter Umständen durch dieses Verhalten ausgebremst werden.

### 4.4.2. Web-App

Eine Web-App ist eine Art von App, die komplett im Browser des Zielgerätes läuft und im Regelfall mit einer Kombination aus HTML, CSS und JavaScript entwickelt wird. Das Verhalten einer Web-App unterscheidet sich im optimalen Fall nicht von ein nativen App, um dem Benutzer eine möglichst gute Benutzererfahrung zu bieten.

Ein signifikanter Vorteil von einer Web-App ist es, dass diese nicht an eine bestimmte Plattform gebunden ist. Voraussetzung für die korrekt Funktion einer Web-App ist ein Browser der die von der App genutzten Funktionen unterstützt. Alle modernen mobilen Browser wie zum Beispiel *Chrome mobile* oder *Firefox mobile* bieten eine volle Unterstützung von HTML5 an [Fir15].

Ein weiterer Vorteil ist es, dass der Benutzer allein durch das Aufrufen einer Webseite Zugang zu der App erhält. Aufwändige Installationsprozesse entfallen. Hierdurch ergibt sich zudem der Vorteil, dass beim Aktualisieren der App keine komplizierten Updateroutinen erstellt werden müssen. Für eine Aktualisierung beim Anwender genügt ein erneutes Laden der Webseite, beziehungsweise App.

Ein Nachteil einer Web-App ist, dass JavaScript eine Skriptsprache ist und somit die implementierte Logik zur Laufzeit kompiliert und ausgeführt wird. Aus diesem Grund und dadurch, dass der Browser

als zusätzliche Anwendungsschicht zwischen der eigentlichen App und der zugrundeliegenden Hardware liegt, kann die Leistung beeinträchtigt werden.

Eine Web-App ist für einen Anwender im Grunde eine einfache Webseite. Hierdurch wird eine Netzwerkverbindung zu dem Server, über welche diese ausgeliefert wird, vorausgesetzt. Dies wiederum schließt eine Offline-Nutzung der App in vielen Fällen aus. Ausgenommen hiervon sind Apps, die schon einmal vom Benutzer geladen wurden und anschließend auch ohne Kontakt zum Server funktionieren.

Die zwingend benötigte Verbindung zu einem Server bringt auch Vorteile mit sich. So kann es aus Sicherheitsgründen nicht gewünscht sein, sensible Daten direkt auf dem Gerät zu speichern. Durch konstant vorhandene Verbindung ist es möglich, anstatt alle Daten auf einmal, nur die momentan benötigten dem Client zur Verfügung zu stellen. Hierdurch können diese vor unberechtigtem Zugriff geschützt werden.

### 4.4.3. Hybride App

Eine hybride App kombiniert eine Web-App mit einer nativen App. So werden Teile in der nativen Sprache der jeweiligen Plattform und andere Teile mit HTML/CSS/JavaScript implementiert. Im Allgemeinen wird die Benutzeroberfläche mit HTML/CSS/JavaScript gestaltet und in einem *headless Browser* ausgeführt. Headless bedeutet in diesem Fall, dass der Browser keine Kontrollelemente, wie zum Beispiel eine Adresszeile enthält. Hierdurch soll es für den Benutzer nicht ersichtlich sein, dass er einen Browser verwendet.

In einer hybriden App können die Vorteile einer nativen App mit den Vorteilen einer Web-App verbunden werden. Der Benutzer kann diese Art von App wie jede andere auf dem System installierte App starten und muss sich nicht wie bei einer Web-App um das Aufrufen einer bestimmten URL im Browser kümmern. Trotz dieser nativen Benutzererfahrung ist es für den Entwickler einfacher die App auf verschiedenen Plattformen zu portieren, da es eine gemeinsame Codebasis gibt.

Frameworks wie *PhoneGap* [Ado15] ermöglichen es die komplette Logik und Oberfläche der App mit HTML, CSS und JavaScript zu erstellen. Die Integration der App in die zugrundeliegende Betriebssysteme wird von PhoneGap übernommen. Ein weiterer Vorteil dieses Frameworks ist es, dass Funktionen die in einem normalen Browser verfügbar sind, auch in der hybriden App verwendet werden können. Hierzu zählt zum Beispiel die Verwendung der Sensoren des mobilen Endgeräts.

Hybride Apps haben allerdings nicht nur Vorteile. Durch den extensiven Einsatz von Web Technologien haben diese Apps im Allgemeinen genau wie Web-Apps eine schlechtere Leistung als komplett native Apps. Aus diesem Grund eignet sich diese Art von App nicht für aufwändige Anwendungen wie zum Beispiel Spiele.

### 4.4.4. Bewertung der verschiedenen Arten von Apps

Dieses Unterkapitel vergleicht die wichtigsten Eigenschaften der drei vorgestellten Arten von Apps, um eine bessere Übersicht über die diversen Vor- und Nachteile der einzelnen Arten zu bekommen.

### Leistung

Die verschiedenen Arten von Apps sind je nach Anwendungsfall unterschiedlich leistungsfähig. Native Apps können die zugrundeliegende Hardware am effizientesten verwenden, da hier kompilierter Code verwendet wird und keine zusätzliche Schicht wie ein Browser zwischen der App und der Hardware liegt. Hybride Apps, bei welchen die komplette App mit Webtechniken erstellt wurden und Web-Apps bieten in etwa die gleiche Leistung. Dies liegt daran, dass hier eine Skriptsprache verwendet wird und durch den Browser eine zusätzliche Anwendungsschicht zur Ausführung der App benötigt wird.

Muss eine App entwickelt werden, bei der eine gute Leistung sehr wichtig ist, sollte eine native App entwickelt werden. Werden lediglich Informationen konsumiert und das Benutzerinterface dient hauptsächlich dazu diese Informationen anzuzeigen, so eignen sich auch hybride und Web-Apps.

### Wartbarkeit

Nachdem eine Software ausgeliefert wird, ist deren Entwicklung im Regelfall nicht abgeschlossen. Aus diesem Grund ist die Wartbarkeit ein wichtiger Punkt bei der Wahl der Technologie mit welcher diese Software entwickelt wird.

Von den drei vorgestellten Typen für Apps ist die Wartung nativer Apps am aufwändigsten. Insbesondere wenn eine native App auf verschiedenen Plattformen verfügbar sein soll, müssen diese einzelnen Plattformen einzeln gepflegt werden. Außerdem müssen aktualisierte Versionen der App über einen Updatemechanismus verteilt werden.

Hybride Apps sind einfacher zu warten als native Apps, da diese eine Codebasis für alle unterstützen Plattformen haben. Sollten Anpassungen am nativen Teil nötig sein, so müssen diese pro Plattform einzeln umgesetzt werden. Aus diesem Grund ist eine hybride App besser zu warten als eine native App, allerdings schlechter als eine Web-App. Bei der Aktualisierung der App ergibt sich bei einer hybriden App die gleiche Problematik wie bei der Aktualisierung einer nativen App, da auch hier ein Updatemechanismus vorhanden sein muss.

Reine Web-Apps sind am einfachsten zu warten, da diese nicht innerhalb des Systems installiert sind und somit auf einen aufwändigen Updatemechanismus verzichten können. Allein durch das erneute Laden der App vom Server kann der Benutzer eine aktualisierte Version erhalten. Zudem gibt es nur eine Codebasis die bei einer Aktualisierung berücksichtigt werden muss.

### Plattformabhängigkeit

In der Welt der Apps für mobile Endgeräte gibt es eine Vielzahl von Plattformen. Sollen nur die drei größten Plattformen Android, iOS und Windows Phone unterstützt werden, so benötigt man bei einer nativen App im Grunde drei verschiedene Apps, die jeweils mit unterschiedlichen Technologien entwickelt werden müssen.

Durch die geteilte Codebasis sind hybride Apps leichter auf andere Plattformen portierbar als native Apps. Allerdings besteht auch hier ein gewisser Teil der App aus nativem Code, der portiert werden muss.

Die beste Plattformunabhängigkeit besitzen Web-Apps. Diese sind nur auf einen Browser angewiesen, der aktuelle Technologien wie zum Beispiel HTML5 unterstützt. Aktuelle Versionen der großen Plattformen Android, iOS und Windows Phone bringen solch eine Unterstützung von Haus aus mit.

#### Gerätefunktionen

Aktuelle mobile Endgeräte besitzen eine Vielzahl von Sensoren und Funktionen. Native Apps und der native Teil von hybriden Apps haben durch den direkten Zugriff auf das System im Allgemeinen Zugriff auf diese. Web-Apps hingegen haben hierauf nur sehr eingeschränkten Zugriff, da ein Browser die App vom restlichen System abkapselt.

#### Optische Integration an die Zielplattform

Um einem Nutzer die beste Erfahrung zu bieten ist eine möglichst gute Integration der App in das vorhandene System nötig. Durch die Nähe zu der jeweiligen Plattform bieten native App die beste Benutzererfahrung. Die Frameworks der einzelnen Plattformen bringen vorgestaltete Elemente wie zum Beispiel Buttons mit, die sich perfekt in die jeweilige Plattform integrieren.

Hybride Apps und Web-Apps haben eine Benutzeroberfläche, die mit Webtechniken gestaltet sind. Zudem ist diese Oberfläche meist für die Verwendung in verschiedensten Plattformen ausgelegt. Aus diesem Grund haben diese nicht die gleiche Nähe zum System wie eine native App. Somit lässt sich eine an die Zielplattform angepasste Optik nur mit erhöhtem Aufwand erreichen.

### 4.4.5. Fazit

Für die Realisierung der App wurde die Technologie der Web-App gewählt. Der Hauptgrund hierfür ist die Plattformunabhängigkeit. Zudem werden von der Apps hauptsächlich zuvor aufbereitete Daten angezeigt, weswegen hierfür keine große Rechenleistung nötig ist. Somit kann ein Teil dieser für das Anzeigen der App im Browser verwendet werden. Funktionen, wie zum Beispiel spezielle Sensoren, die ausschließlich über native Apps zugänglich sind, werden nicht benötigt.

## 4.5. Visualisierung einer Baumstruktur auf einem Tablet

Das Anzeigen der Taxonomie auf einem Tablet ist eine der Hauptfunktionen, die die zu entwickelnde App bieten soll. Die Analyse der vorhandenen Taxonomie aus Kapitel 4.2 hat ergeben, dass diese als azyklischer gerichteter Graph und nicht in einer Baumstruktur vorliegt. Trotz dieser Tatsache, verwendet der in Kapitel 4.3 vorgestellte Taxonomie Editor eine Baumstruktur zur Visualisierung der Taxonomie.

Auch in der App soll eine Baumstruktur verwendet werden. Dies hat den Hintergrund, dass sich solch eine Struktur leichter als einen kompletten azyklischen gerichteten Graphen anzeigen lässt. Zudem liegen große Teile der vorhandenen Taxonomie in einer reinen Baumstruktur vor und nur sehr

wenige Konzepte haben mehr als ein Elternkonzept. Im weiteren Verlauf wird davon ausgegangen, dass alle in dieser Form vorliegenden Taxonomien diese Eigenschaft aufweisen. Um den Fall abbilden zu können, in welchem ein Konzept mehr als ein Elternkonzept hat, wird eine Redundanz bei der Anzeige des Baumes in Kauf genommen. Hat ein Konzept mehr als ein Elternkonzept, so erscheint dieses unterhalb eines jeden Elternkonzepts im Taxonomiebaum. Wird das Kindkonzept an einer Stelle im Baum verändert, so wirkt sich diese Änderung auch auf alle anderen Stellen aus, in welchen dieses vorkommt.

Um eine Baumstruktur auf einem Tablet abzubilden, wurden sechs verschiedene Ansätze untersucht. Die Ansätze lassen sich in die zwei Kategorien bildschirmfüllende und nicht bildschirmfüllende Visualisierungen einteilen. Bei den bildschirmfüllendenen Visualisierungen wird der komplette verfügbare Platz des Bereichs für den Taxonomiebaum genutzt. Zu diesen Ansätzen gehören die *Treemap* und *RELT*. Die *einfache Liste*, *Side-by-Side Liste*, *eingerückte Liste* und *Tablorer* gehören zu den nicht bildschirmfüllenden Visualisierungen. Platz der nicht für den Taxonomiebaums benötigt wird, kann hier zum Anzeigen der Details eines Konzepts verwendet werden. Zu diesen Details gehören die Synsets, Synonyme, Tokens und Attribute des Konzepts.

Bei der Suche nach einer für das Problem passende Visualisierung wurde besonders auf die folgenden Punkte geachtet:

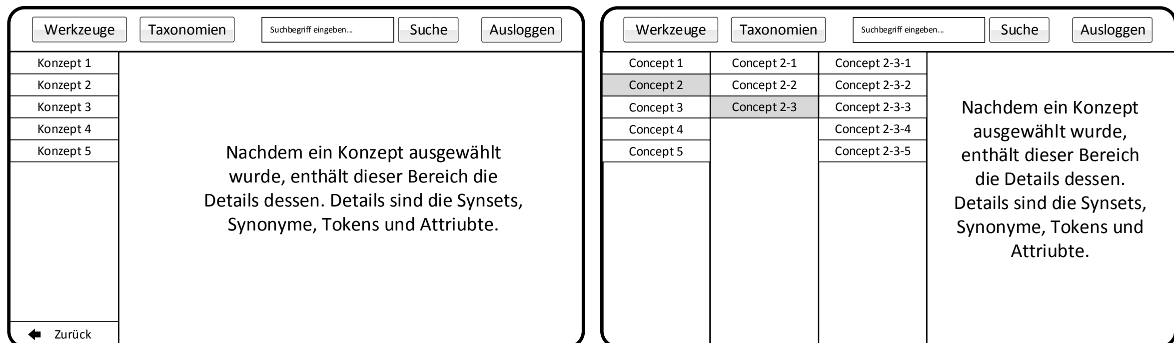
- **Intuitive Benutzbarkeit**. Ein Benutzer soll mit keiner oder nur kurzer Einweisung die Visualisierung verstehen und benutzen können.
- **Leichte Implementierbarkeit**. Durch den beschränkten Zeitrahmen der Arbeit und dadurch, dass die Visualisierung der Taxonomie nicht das einzig zu lösende Problem ist, soll die Visualisierung einfach zu implementieren sein.
- **Interaktivität**. Der Benutzer soll mit der Visualisierung interagieren, um den Taxonomiebaum zu erkunden und sich Details zu einem Konzept anzeigen lassen zu können.
- **Lazy Loading**. Es sollen nur die gerade benötigten Daten einer Taxonomie geladen und visualisiert werden, damit auch sehr große Taxonomien verarbeitet werden können.

Die folgende Analyse der verschiedenen Vorgehensweisen der Visualisierung enthält für jede ein Mockup zur besseren Veranschaulichung.

### 4.5.1. Einfache Liste

Der einfachste der vorgestellten Ansätze verwendet eine einfache vertikale Liste zur Visualisierung der Baumstruktur. Hierbei wird eine Ebene, sprich ein Konzept und dessen Geschwister, in einer Liste zur gleichen Zeit angezeigt.

Wird ein Konzept innerhalb der Liste angeklickt, so wird diese geleert und mit den Kindkonzepten des gewählten Konzeptes befüllt. Auf diese Weise lässt sich der Baum in der Tiefe erkunden. Um auf eine höhere Ebene des Baums zu gelangen, muss der sich am unteren Ende der Liste befindliche Zurück-Button verwendet werden. Existieren auf einer Ebene mehr Konzepte, als auf den Bildschirm passen, so kann die Liste vertikal gescrollt werden. Eine graphische Visualisierung des beschriebenen Konzepts ist in Abbildung 4.4a zu sehen.




**(a)** Mockup einer Visualisierung einer Baumstruktur mit einer einfachen Liste

**(b)** Mockup einer Visualisierung einer Baumstruktur mit einer Side-by-Side Liste

**Abbildung 4.4.:** Visualisierungen einer Baumstruktur mit einer einfachen und einer Side-by-Side Liste

Dieser Ansatz hat den Vorteil, dass er sehr leicht zu implementieren ist und einfach von einem Benutzer verstanden werden kann, da diese mit dieser Art von Visualisierung vertraut sind. Diese wird zum Beispiel bei modernen Android Smartphones im Einstellungsmenü verwendet.

Ein weitere Vorteil dieser Visualisierung ist es, dass nicht alle Daten der Taxonomie auf dem Tablet vorgehalten werden müssen. Dadurch, dass immer nur eine Ebene zur gleichen Zeit angezeigt wird, können die benötigten Daten nach Bedarf im Speicher des Tablets liegen.

Durch das Anzeigen der Konzepte in einer einzigen Liste, kann der Benutzer bei komplizierten Bäumen schnell die Übersicht verlieren, in welchem Teil des Baumes er sich momentan befindet. Um sich bei der Orientierung innerhalb des Baumes nicht auf das Gedächtnis des Anwenders verlassen zu müssen, wäre es denkbar einen Breadcrumb mit dem aktuellen Pfad zu verwenden. Ein Breadcrumb ist eine horizontale Liste bestehend aus Textelementen oder Links, die den gewählten Pfad innerhalb des Baumes visualisieren.

Ein weiterer Nachteil dieser Visualisierung ist die schlechte Nutzung des vorhandenen Platzes. Auf kleinen Bildschirmen wie zum Beispiel bei Smartphones ist es oft sinnvoll, dass die Liste die komplette Bildschirmbreite in Anspruch nimmt. Bei Tablets würden sich bei dieser Vorgehensweise sehr breite Elemente bilden, die an den meisten Stellen leer wären. Der verfügbare Platz wird schlecht genutzt. Aus diesem Grund wurde bei Abbildung 4.4a nur ein Teil der verfügbaren Fläche für das Anzeigen der Liste verwendet und der Rest für die Visualisierung der Details eines ausgewählten Konzepts.

### 4.5.2. Side-by-Side Liste

Als eine Erweiterung der einfachen Liste ist die als Side-by-Side Liste bezeichnete Visualisierung anzusehen. Hierbei wird für jede Ebene des Baumes eine neue Liste erstellt. Geöffnete Listen werden horizontal nebeneinander dargestellt, wobei von links nach rechts jede Liste eine tiefere Ebene des Baumes repräsentiert. Enthält eine einzelne Liste mehr Elemente, als auf dem Bildschirm angezeigt werden können, so kann diese unabhängig von den anderen gescrollt werden. Besitzt der Baum zu

viele Ebenen und es wird horizontal mehr Platz benötigt, als der Bildschirm bietet, so können alle angezeigten Listen gemeinsam horizontal gescrollt werden.

Abbildung 4.4b zeigt die beschriebene Visualisierung anhand eines Beispiels. In der Abbildung werden drei Ebenen eines Baumes gezeigt. Auf der obersten der gezeigten Ebenen enthält der Baum fünf verschiedene Konzepte. Das Konzept mit der Bezeichnung *Concept 2* besitzt einen grauen Hintergrund, was darauf hinweist, dass dieses zuvor ausgewählt wurde. Durch das Auswählen hat sich rechts von der ersten Liste eine zweite geöffnet. Diese enthält wiederum die drei Kindkonzepte des zuvor gewählten Konzepts. In der zweiten Liste wurde das Konzept mit der Bezeichnung *Concept 2-3* gewählt, wodurch sich wiederum rechts davon eine neue List mit dessen Kindkonzepten geöffnet hat. Auf diese Weise lässt sich die komplette Tiefe eines Baums erkunden.

Möchte man in dem Baum zurück gehen, so muss ein Konzept in einer der angezeigten Liste gewählt werden. Anschließend werden alle Listen, die sich rechts von dem gewählten Konzept befinden geschlossen, um Platz für die Kindkonzepte des soeben gewählten Konzeptes zu schaffen.

Diese Art der Visualisierung einer Baumstruktur hat den Vorteil, dass der Benutzer anhand der markierten Konzepte stets den gewählten Pfad innerhalb des Baumes nachvollziehen kann. Erst bei einer sehr großen Anzahl von Ebenen und Kindkonzepten pro Konzept kann dies unübersichtlich für einen Benutzer werden, da er viel scrollen muss.

Ein weiterer Vorteil diese Ansatzes ist der einfache Aufbau, was wiederum zu einer einfachen Implementierung führt. Durch Animationen beim Öffnen einer neuen Liste kann es für den Benutzer leichter ersichtlich sein, woher diese neue List stammt.

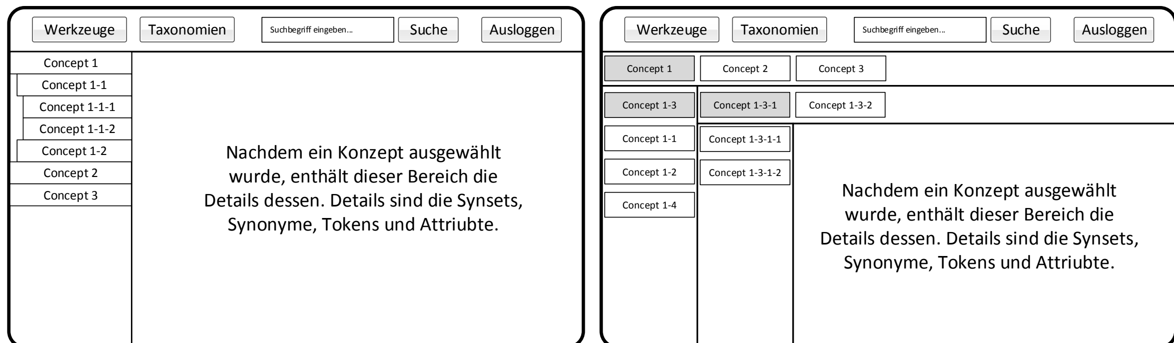
Bei dieser Visualisierung wird nicht von Anfang an der verfügbare Platz des Bildschirms verwendet, was als Nachteil angesehen werden kann. Erst nachdem mehrere Ebenen des Baumes geöffnet wurden, findet eine bessere Nutzung des vorhandenen Platzes statt.

### 4.5.3. Eingerückte Liste

Eine weitere Variante der einfachen Liste ist die eingerückte Liste. Hierbei werden Kinder eines Konzepts unterhalb dessen ein Stück nach rechts verschoben. Diese Art der Visualisierung eines Baumes wird bei einer Vielzahl von Applikationen verwendet. Die wohl bekannteste ist der Windows Explorer, der auf diese Weise das Dateisystem visualisiert. Auch der in Kapitel 4.3 beschriebene Taxonomieeditor verwendet diese Art der Visualisierung.

Abbildung 4.5a zeigt ein Mockup, bei welchem eine eingerückte Liste verwendet wird. Auf der obersten Ebene befinden sich hier vier Konzepte. Angezeigt werden zusätzlich die Kinder und Kindeskinder des Konzepts mit dem Namen *Concept 1*. Dieses Konzept besitzt zwei direkte Kindkonzepte und über das Konzept mit dem Namen *Concept 1-1* zwei Kindeskinder.

Ähnlich wie die zwei zuvor vorgestellten listenbasierten Visualisierung ist auch diese verhältnismäßig einfach zu implementieren. Zudem ist dieses Konzept vielen Nutzern vertraut, weshalb dieses intuitiv genutzt werden kann. Durch das Einrücken der einzelnen Konzepte lässt sich einfach nachvollziehen, an welcher Stelle ein Konzept in dem Taxonomiebaum steht.




**(a)** Mockup einer Visualisierung einer Baumstruktur mit einer eingerückten Liste

**(b)** Mockup einer Visualisierung einer Baumstruktur mit *Tablorer*

**Abbildung 4.5.:** Visualisierungen einer Baumstruktur mit einer eingerückten List und Tablorer

Alle Konzepte werden untereinander aufgelistet, weswegen der horizontal verfügbare Platz schlecht genutzt wird. Zudem muss bei großen Taxonomien viel vertikal gescrollt werden, was dazu führen kann, dass der Benutzer die Übersicht verliert.

### 4.5.4. Tablorer

Tablorer ist ein in [SPH11] von HyunJu Shin et. al. vorgestelltes Systeme zur Visualisierung von Bäumen auf Tablets. Es basiert auf horizontalen und vertikalen Listen, die alternierend im Sichtbereich des Nutzers (Viewport) angeordnet werden. Anhand des in Abbildung 4.5b zu sehenden Mockups soll das Konzept von Tablorer genauer erklärt werden.

Die erste Ebene des Baums wird in einer horizontalen Liste, die sich am oberen Ende des Viewports befindet, dargestellt. In dem Beispiel wurde das Konzept mit der Bezeichnung *Concept 1* ausgewählt. Dieses wurde mit einer grauen Hintergrundfarbe markiert. Unterhalb der ersten Liste wird eine zweite, dieses mal vertikale Liste, mit den Kindkonzepten des gewählten Konzepts geöffnet. Hier wird das Konzept mit der Bezeichnung *Concept 1-3* gewählt. Im Normalfall würde dieses Konzept an dritter Stelle der Liste stehen. Bei Tablorer wird allerdings das gewählte Element immer an die erste Stelle der jeweiligen Liste verschoben, damit der Benutzer sofort und ohne visuelle Suche die gewählten Elemente erfassen kann.

Die Kinder des Konzepts *Concept 1-3* werden anschließend in einer horizontalen Liste dargestellt. Kindkonzepte von Konzepten aus dieser Liste werden wiederum in einer vertikalen Liste angezeigt. Durch diese alternierende Vorgehensweise lässt sich der komplette Baum mit hoher Effizienz erkunden.

Sollte innerhalb einer der Listen nicht genügend Platz zum Anzeigen aller Elemente existieren, so lassen sich diese je nach ihrer Ausrichtung horizontal oder vertikal scrollen. Hat der anzuzeigende Baum zu viele Ebenen, als das diese auf der vorhandenen Bildschirmfläche Platz finden würden, so ist ein vertikales und horizontales Scrollen des kompletten Viewports möglich.

Dieser Ansatz hat den Vorteil, dass der vorhandene Platz im Vergleich zu den meisten anderen bisher vorgestellten Ansätzen besser genutzt wird. Durch das Verschieben der ausgewählten Konzepte an die erste Stelle der jeweiligen Liste ermöglicht es ohne großen Aufwand den Pfad der gewählten Elemente bis zur Wurzel schnell zu erfassen.

Der Nachteil dieses Ansatzes ist es der im Vergleich zu den bisher vorgestellten Ansätzen erhöhte Aufwand bei der Implementierung. Zudem kann das Verschieben der gewählten Elemente an die erste Stelle der jeweiligen Liste die Benutzer insofern verwirren, da sich die Sortierung innerhalb der Liste verändert.

### 4.5.5. Treemap

Eine Treemap ist eine Technik die den kompletten Platz auf einem Bildschirm zur Visualisierung einer Baumstruktur verwendet. Treemaps gibt es in verschiedensten Ausprägungen, wobei hier eine einfache Version analysiert wird. Im Allgemeinen ist eine Treemap eine Struktur, die durch ineinander verschachtelte Rechtecke dargestellt wird. Bei der hier vorgestellten Treemap wird immer nur eine Ebene des Baums durch Rechtecke gleichzeitig dargestellt, wodurch diese nicht ineinander verschachtelt sind.

Diese einfache Art wurde gewählt, da eine traditionelle Treemap voraussetzt, dass alle Daten lokal zur Verfügung stehen und somit dem Ziel gegenüber steht Daten nur bei Bedarf zu laden und zu visualisieren. Abbildung 4.6a zeigt solch eine Treemap in einem Mockup.

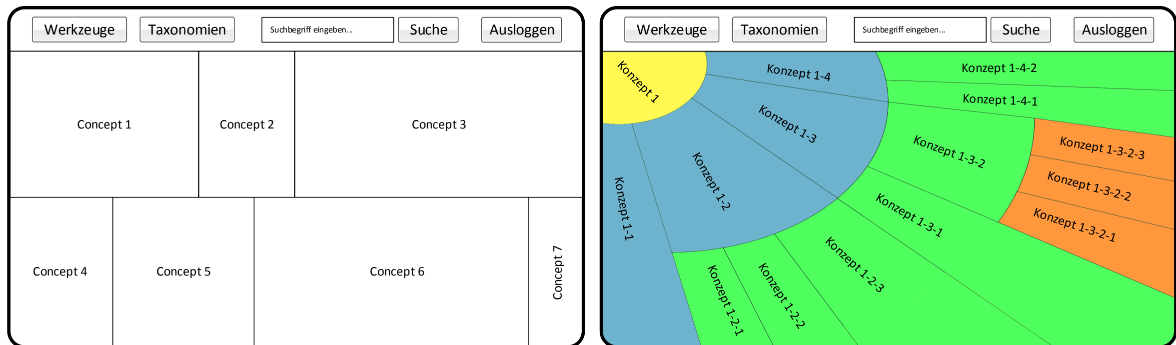
Die Konzepte werden als einfache Rechtecke dargestellt, wobei die Fläche eines Rechtecks die Anzahl der Kindkonzepte repräsentiert. Je größer die Fläche ist, desto mehr Kindkonzepte hat ein Konzept. Unabhängig von der Anzahl der Konzepte wird immer die komplette verfügbare Bildschirmfläche genutzt.

Diese Art der Visualisierung hat den Vorteil, dass immer der komplette verfügbare Platz verwendet wird. Dies ist für das Anzeigen des Taxonomiebaums erstrebenswert. Gleichzeitig hat diese Eigenschaft den Nachteil, dass es für die Details eines Konzepts keine verfügbare Fläche mehr gibt. Diese müssen in einer eigenen Ansicht präsentiert werden. Dadurch ist es nicht möglich zeitgleich den Taxonomiebaum und die Details eines Konzepts zu betrachten.

Des Weiteren ist die Implementierung komplizierter als die zuvor vorgestellten Methoden, da für die anzuzeigenden Konzepte eine passende Größe des zu zeichnenden Rechtecks gefunden werden muss. Zudem ist es für einen Benutzer schwer nachzuvollziehen an welcher Stelle im Taxonomiebaum er sich momentan befindet, da immer nur eine Ebene des Baumes zur gleichen Zeit dargestellt werden kann.

### 4.5.6. Radial Edgeless Tree (RELT)

Radial Edgeless Trees sind eine in [CZJ13], [HGZ$^{+}$10] und [HZ] vorgestellte Methode zur Visualisierung von Baumstrukturen auf mobilen Endgeräten. Ein RELT ist eine bildschirmfüllende Struktur, die sich von der oberen linken Ecke des Bildschirms nach unten rechts ausbreitet. Die genaue Funktionsweise soll anhand des in Abbildung 4.6b zu sehenden Beispiels gezeigt werden.




**(a)** Mockup einer Visualisierung einer Baumstruktur mit einer Treemap

**(b)** Mockup einer Visualisierung einer Baumstruktur mit *RELT*

**Abbildung 4.6.:** Visualisierungen einer Baumstruktur mit einer Treemap und RELT

Das Wurzelkonzept, in diesem Fall mit *Concept 1* bezeichnet, befindet sich in der oberen linken Ecke und ist mit einer gelben Hintergrundfarbe gekennzeichnet. Die nächste Ebene des Baumes ist kreisförmig um das Wurzelkonzept herum angeordnet und mit einer blauen Hintergrundfarbe markiert. Grundsätzlich wird eine Ebene immer mit der gleichen Farbe, die sich grundsätzlich von allen direkt anliegenden Farben unterscheidet, gekennzeichnet. Dies soll dem Benutzer helfen die zu einer Ebene gehörenden Elemente einfacher zu identifizieren.

Konzepte die keine Kindkonzepte haben gehen grundsätzlich bis zum Rand des Bildschirmausschnittes, sodass kein Stelle ungenutzt bleibt. Sollten mehr Konzepte existieren, als sinnvoll mit dem verfügbaren Platz dargestellt werden können, so unterstützt ein RELT das sogenannte *panning*. Beim Panning kann der Bildschirmausschnitt in eine beliebige Richtung verschoben werden. Diese Technik erlaubt es somit gleichzeitig horizontal und vertikal zu scrollen. Durch das Verschieben werden mehr Konzepte sichtbar, welche auch erst zu diesem Zeitpunkt geladen werden müssen.

Diese Art der Visualisierung hat den Vorteil, dass der komplette verfügbare Platz auf einem Bildschirm verwendet wird und der Benutzer schnell eine Übersicht über die vorhanden Taxonomie erlangen kann.

Ein gravierender Nachteil dieser Technik ist die Komplexität, die sich beim Implementieren ergibt. So muss entschieden werden wie viele Elemente momentan sinnvoll auf dem Bildschirm Platz finden und wie groß diese sein müssen. Zudem dürfte auch die Zeichenroutine zum Erstellen des RELT nicht trivial sein.

Des Weiteren können sich bei vielen Konzept schnell sehr schmale Elemente innerhalb der Visualisierung ergeben, mit welchen sich über Touchgesten nur schwer interagieren lässt. Sollen schmale Elemente vermieden werden, so muss unter Umständen viel gescrollt werden.

### 4.5.7. Fazit

Zur Visualisierung des Taxonomiebaums wird die vorgestellte Side-by-Side Liste gewählt. Folgende Gründe sind ausschlaggebend für die Verwendung:

- **Gute Nutzung der verfügbaren Bildschirmfläche.** Mit dieser Technik ist es sowohl möglich die komplette verfügbare Bildschirmfläche zu nutzen, als auch nur einen Teil dieser. Je nach Anforderung kann die Aufteilung zur Laufzeit geändert werden.
- **Gute Skalierbarkeit.** Es wird nicht der komplette Taxonomiebaum zu Beginn benötigt, da ausschließlich gerade angezeigt Ebenen des Baums im Hauptspeicher vorgehalten werden müssen. Enthält eine Ebene sehr viele Konzepte, so ist es denkbar nur ein Teil dieser zu laden und bei Bedarf weitere vom Server anzufordern.
- **Einfaches Erkennen des gewählten Pfades.** Es wird immer der komplette Pfad der aktuell gewählten Konzepte angezeigt. Ein mit grauem Rahmen markiertes Konzepts innerhalb des Taxonomiebaums, stellt das aktuell gewählte Konzept dar. Alle Listen die links von diesem dargestellt werden, sind Kindkonzepte dessen.
- **Einfache Navigation durch den Taxonomiebaum.** Durch die Verwendung eines einfachen Buttons lassen sich die Kindkonzepte eines bestimmten Konzeptes vom Server abrufen und anzeigen. Dadurch, dass immer der komplette Pfad zur Wurzel der momentan gezeigten Elemente angezeigt wird, is es ohne weiteres möglich innerhalb des Baums zurück zu navigieren.
- **Einfach Implementierung.** Verglichen mit anderen in Kapitel 4.5 vorgestellten Visualisierungen von Baumstrukturen, ist die gewählte Side-by-Side Liste verhältnismäßig einfach umzusetzen. Dies ist damit zu begründen, dass das Anzeigen mehrerer horizontal angeordneten vertikaler Listen mit Webtechniken einfach zu realisieren ist.

# 5. Entwurf und Realisierung

Dieses Kapitel behandelt den Entwurf und die Realisierung des Gesamtsystems. Hierzu zählt auf der Clientseite die App für Tablets und auf der Serverseite ein Server zur Verwaltung der Taxonomien. Zuerst wird ein Überblick über das komplette System gegeben. Anschließend findet eine detaillierte Aufbereitung der Serverseite und darauf folgend der Clientseite statt.

## 5.1. Aufbau des Gesamtsystems

In Kapitel 4.2 wird beschrieben, dass die vorhandene Taxonomie in XML-Dateien vorliegt. Um diese auf einem Tablet verfügbar zu machen, gibt es prinzipiell zwei verschiedene Möglichkeiten. Zum einen kann die Taxonomie direkt auf dem Zielgerät gespeichert werden, um lokal mit dieser zu arbeiten. Zum anderen kann die Taxonomie auf einem Server vorliegen, welcher diese dem Client zur Verfügung stellt.

Wird die Taxonomie direkt auf dem Gerät gespeichert entfällt die Implementierung eines Servers. Hierdurch fällt der Aufwand der Realisierung des Gesamtsystems gegenüber der zweiten Möglichkeit geringer aus. Ein großer Nachteil der ersten Möglichkeit ist die Synchronisation einer Taxonomie über mehrere Geräte hinweg. Zudem ist es nicht möglich, dass verschiedene Benutzer mit verschiedenen Geräten zeitgleich an einer Taxonomie arbeiten. Ein weiterer Nachteil ist das Thema Sicherheit. Wird ein Geräte mit der vollständigen Taxonomie entwendet, so kann zum Beispiel über die in [ST10] vorgestellte Taxonomie viel über die in Fahrzeugen verwendeten Bauteile und deren Schwächen und Probleme in Erfahrung gebracht werden.

Diese Probleme lassen sich durch das Verwenden einer Client-Server-Architektur beheben. Wird eine Taxonomie in einer an den Server angebundenen Datenbank gespeichert, so wird diese im Gegensatz zur lokalen Speicherung nur an einer Stelle des Systems vorgehalten. Hierdurch entfällt das aufwändige Synchronisieren zwischen verschiedenen Geräten, allerdings muss stattdessen ein Server zur Verwaltung der Taxonomien entwickelt werden. Dies erhöht den Aufwand zur Realisierung des Gesamtsystems.

Ein Server bietet die Möglichkeit dem Client jeweils nur gerade benötigte Teile einer Taxonomie auszuliefern. Diese werden zudem ausschließlich im Hauptspeicher des Clients vorgehalten. Hierdurch kennt dieser zu keinem Zeitpunkt die komplette Taxonomie, was die Angreifbarkeit des Systems verringert.

Aufgrund der Synchronisationsproblematik wurde der Client-Server Ansatz gewählt. Abbildung 5.1 zeigt den prinzipiellen Aufbau des zu Systems. In diesem System greift der Client, sprich die App,

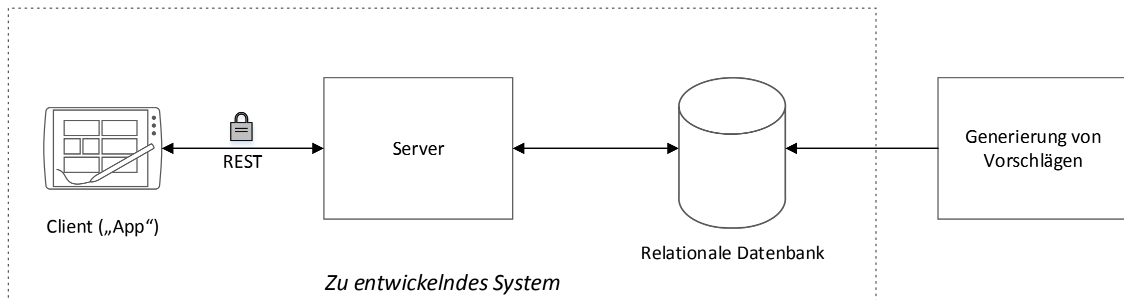



**Abbildung 5.1.:** Aufbau des Gesamtsystems. Ein Client greift über einen Server auf die in der relationalen Datenbank gespeicherten Daten zu. Vorschläge für neue Synonyme werden außerhalb des zu entwickelnden Systems generiert.

über eine REST-Schnittstelle auf einen Server zu. Um die Kommunikation abzusichern wird jeglicher Datenverkehr verschlüsselt. Zudem erfordert die REST-Schnittstelle eine Authentifizierung zur Verwendung.

Der Server ist mit einer relationalen Datenbank verbunden, die zur Speicherung der Daten verwendet wird. Diese drei Komponenten – Client, Server und SQL-Datenbank – bilden das zu entwickelnde System.

Das Gesamtsystem hingegen besteht aus einer zusätzlichen Komponente. Diese geniert Vorschläge, die für die Erweiterung einer Taxonomie verwendet werden können. Diese generierten Vorschläge werden direkt in die Datenbank geschrieben, um anschließend von dem zu entwickelnden System verarbeitet zu werden. Die Generierung der Vorschläge ist nicht Teil dieser Arbeit.

## 5.2. Serverseite

Folgend wird der Entwurf und Realisierung der Serverseite betrachtet. Zu Beginn wird auf die verwendeten Technologien eingegangen. Anschließend wird ein Überblick über den Aufbau des Servers gegeben. Daraufhin findet eine Betrachtung der von [SFB+15] übernommenen Benutzer- und Rechteverwaltung statt. Nachfolgend wird erläutert, wie Taxonomien über eine REST Schnittstelle verwaltet werden können. Abschließend werden Entwurf und Realisierung des Imports, Exports und der Fehlerbehandlung betrachtet.

### 5.2.1. Verwendete Technologien und Frameworks

In der heutigen IT-Welt werden in den meisten Projekten Technologien und Frameworks eingesetzt, die extern und unabhängig von diesen entwickelt und gewartet werden. Ohne dieses Vorgehen wäre die Entwicklung komplexer Systeme nur mit sehr großem Aufwand und somit Kosten möglich. Aus diesem Grund werden im Rahmen dieser Arbeit verschiedene schon vorhandene Technologien

verwendet. Folgend werden die Technologien, die für die Realisierung des Servers verwendet wurden, näher betrachtet.

Bei der Implementierung des Servers wird auf die *Java Enterprise Edition (JEE)* gesetzt. Der JEE Code läuft in dem Anwendungsserver *GlassFish*. GlassFish [Ora14] ist Open-Source und wurde ursprünglich von Sun Microsystems entwickelt. Inzwischen wird diese Entwicklung von der Oracale Corporation übernommen. Anwendungsserver führen Anwendungen serverseitig aus und bieten darüber hinaus im Allgemeinen zusätzliche Funktionen an. So übernehmen diese zum Beispiel die Anbindung an Datenbanken, so dass diese über eine fest definierte Schnittstelle genutzt werden können.

Die REST Schnittstelle wird über das *Jersey* Framework realisiert. Jersey ist die Referenzimplementierung der *Java API for RESTful Web Services (JAX-RS)* [OPC15]. Eine JAX-RS kompatible Implementierung ermöglicht es über wenige Annotationen eine Methode einer Java-Klasse über HTTP erreichbar zu machen. Diese über HTTP erreichbare Methoden wiederum erlauben es eine REST Schnittstelle zu implementieren.

Zur persistenten Speicherung der Daten wird eine relationale Datenbank verwendet. Um SQL Anfragen gegen diese auszuführen, bietet Java die sogenannte *Java Database Connectivity (JDBC)* API[1]. Diese API erlaubt es Datenbankanfragen unabhängig von der verwendeten relationalen Datenbank zu erstellen, da diese von der API gekapselt werden [Ora15]. Das Problem von JDBC ist es, dass verhältnismäßig viel Code für eine Anfrage geschrieben werden muss. Um dies zu vereinfachen, wird das *sql2o* Framework verwendet. sql2o kapselt den JDBC Code und ermöglicht es mit wenigen Zeilen Code den gleichen Effekt wie mit reinem JDBC zu erhalten [Aab15].

Zur Speicherung der Daten wird die relationale Datenbank *MySQL* verwendet. MySQL steht unter der *GNU General Public License (GPL)* und ist somit Open-Source-Software. Diese Datenbank ist eine der weltweit am häufigsten verwendeten Datenbanken [Sch13], wurde erstmals 1995 veröffentlicht und ist heutzutage für die am häufigsten verwendeten Betriebssysteme erhältlich (unter anderem für Windows, Linux und OS X).

### 5.2.2. Allgemeiner Aufbau des Servers

Der Serverteil des Systems besteht aus mehreren Komponenten, die im Folgenden anhand des in Abbildung 5.2 zu sehenden Paketdiagramms, erklärt werden soll. Dargestellt werden in diesem Diagramm die Pakete auf der obersten Ebene des Servers.

Das Paket `rest` enthält alle Klassen, die allgemein für die Bereitstellung der REST Schnittstelle benötigt werden. Diese werden von den, in den Paketen `usermanagement` und `taxonomy` enthaltenen Klassen, zur Realisierung einer REST Schnittstelle verwendet. Die persistenten Speicherung der anfallenden Daten wird von einer relationalen Datenbank übernommen. Um den Zugriff auf diese zu vereinfachen und zu zentralisieren, greifen die Klassen in den Paketen `usermanagment` und `taxonomy` über die im Paket `db` enthaltenen Klassen, auf die Datenbank zu. In dem Paket `util` befinden sich Klassen,

[1]Ein *Application Programming Interface (API)* ist Schnittstelle, die es einem Programmierer erlaubt mit einem System zu interagieren.

die nicht eindeutig einem anderen Paket zugeordnet werden können und allgemeine Funktionen enthalten.

Eine genauere Betrachtung des Pakets `usermanagement` ist in Kapitel 5.2.3 zu finden. Der Inhalt des Paketes `taxonomy` wird in den Kapiteln 5.2.4, 5.2.5, 5.2.6, 5.2.7 und 5.2.8 detailliert vorgestellt.

Das Paket `util` enthält verschiedene Klassen mit Hilfsmethoden und wird im Folgenden nicht näher behandelt. Das Paket `rest` dagegen enthält einige Klassen, die für das Funktionieren des Servers von großer Relevanz sind. Diese werden im Folgenden genauer betrachtet:

- **`TaxonomyApplication`**. Das Jersey-Framework besitzt die Fähigkeit automatisch Klassen zu erkennen, die zum Betreiben einer REST-Schnittstelle benötigt werden. Hierzu zählen sowohl die Klassen, die die eigentliche Schnittstelle bereitstellen, als auch jene, die zur Konfiguration des Frameworks benötigt werden. Damit die automatische Erkennung funktioniert, müssen für Jersey die Pfade zu den Paketen angegeben werden, die für das Framework relevante Klassen enthalten. Diese Pfadangabe zu den Paketen wird in dieser Klasse durchgeführt.
- **`BaseResource`**. Stellt eine Basisressource bereit, die das Erstellen, Löschen und Abfragen eines in der Datenbank speicherbaren Objekts implementiert. In der Datenbank speicherbar sind alle Objekte, deren Klassen von der Basisklasse `Storable` ableiten.
- **`PaginationFilter`** und **`PaginatedCollection`**. Diese zwei Klassen werden dazu verwendet eine Sammlung von Objekten mit Informationen anzureichern, die für die sogenannten Pagination benötigt werden. Pagination ermöglicht es Teile einer großen Sammlung von Objekten abzufragen, um diese einfacher im Client verarbeiten zu können. Aus diesem Grund enthält eine `PaginatedCollection` die Gesamtanzahl der Objekte, die in einer Sammlung vorhanden sind (`total`), die Menge der Objekte die momentan abgefragt werden (`limit`) und den Startindex, ab welchem die Objekte in der Sammlung abgefragt werden (`offset`).
- **`exceptionmapper`**. Dies beschreibt keine Klasse, sondern ein Paket. Dieses enthält diverse Klassen zur Abbildung einer Exception auf einen geeigneten HTML-Statuscode. Diese Mapper

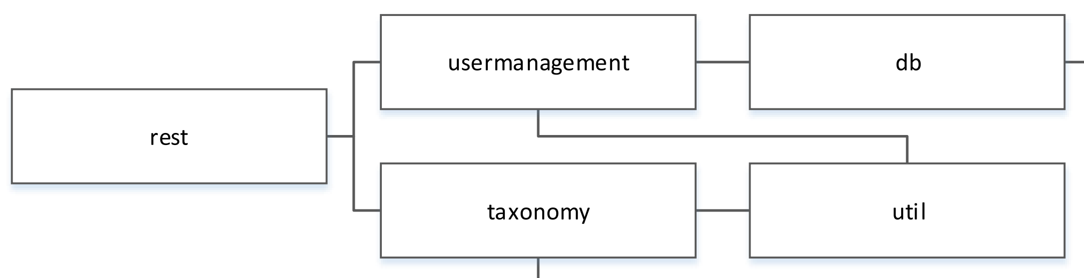



**Abbildung 5.2.:** Visualisierung des Zusammenhangs der einzelnen Paket auf der obersten Ebene des Servers.

ermöglichen es durch das Werfen einer Exception anstatt eines `Internal Server Errors` eine zu der jeweiligen Exception passende Fehlermeldung zu erzeugen.

Nicht alle in den Pakten enthaltenen Klassen wurde im Rahmen dieses Projekts entwickelt. Teile der in den Paketen `db` und `rest` enthaltenen Klassen wurden aus [SFB+15] übernommen und für diese Arbeit angepasst. Der Inhalt des Pakets `usermanagement` hingegen wird nahezu unmodifiziert aus [SFB+15] übernommen.

### 5.2.3. Benutzer- und Rechteverwaltung

Eine Anforderung an die App ist es verschiedene Benutzer innerhalb des Systems zu unterstützen. Aus diesem Grund wird eine Verwaltung dieser Benutzer in einer beliebigen Form benötigt. Der Hauptfokus dieser Arbeit liegt nicht in der Entwicklung einer Benutzer und Rechteverwaltung, weshalb diese mit geringen Änderungen aus einer vorhergehenden Arbeit übernommen und nicht im Rahmen dieser entwickelt wurde [SFB+15]. Aufgrund mangelnder Dokumentation der vorherigen Arbeit, werden an dieser Stelle die wichtigsten Konzepte und der grundlegende Aufbau der Benutzer- und Rechteverwaltung erläutert.

*Hinweis:* Alle im Folgenden gezeigten Konzepte und Implementierungsdetails der Benutzer- und Rechteverwaltung wurden nicht im Rahmen dieser Arbeit erstellt. Diese sind in [SFB+15] entwickelt worden. Alleinig die Dokumentation der Benutzer- und Rechteverwaltung hat im Rahmen dieser Arbeit stattgefunden.

#### Grundkonzept

Die Benutzerverwaltung ermöglicht es mehrere Benutzer anzulegen und diese mit unterschiedlichen Rechten auszustatten. Ein Recht ist ein Konstrukt, das es einem Benutzer erlaubt bestimmte Dinge auszuführen. So könnte zum Beispiel das Recht `taxonomy:view` einem Benutzer erlauben Taxonomien anzuschauen. Das Bearbeiten einer Taxonomie ist diesem allerdings nur möglich, wenn dieser auch das Recht `taxonomy:edit` besitzt. Einem Benutzer können beliebig viele Rechte zugewiesen werden.

Um die Verwaltung der einzelnen Rechte für die Benutzer zu vereinfachen, wird das Konzept der Rollen verwendet. Ein Benutzer kann beliebig viele Rollen haben und einer Rolle wiederum können beliebig viele Rechte zugeordnet sein. So könnte es zum Beispiel die Rolle `administrator` geben, welcher das Recht `user:add` zugeordnet ist, mit welchem neue Benutzer in dem System angelegt werden können. Wird nun einem Benutzer die Rolle `administartor` zugeordnet, so hat dieser automatisch die Rechte einen neuen Benutzer anzulegen.

Eine Visualisierung der Zusammenhänge zwischen Benutzer, Rollen und Rechte ist in Abbildung 5.3 zu sehen. Die beschriebenen Rollen und Rechte ermöglichen die Autorisierung eines Benutzers, sprich es wird festgelegt, was ein Benutzer innerhalb des Systems machen darf. Bevor ein Benutzer allerdings autorisiert werden kann, muss dieser authentifiziert werden. Beim Prozess der Authentifizierung wird festgestellt, welche Entität aktuell eine Anfrage an das System stellt. Zusätzlich wird überprüft, ob die Entität auch die ist, für welche diese sich ausgibt. Im Falle der zu entwickelnden App ist eine Entität im Allgemeinen der Benutzer, der diese aktuell bedient.

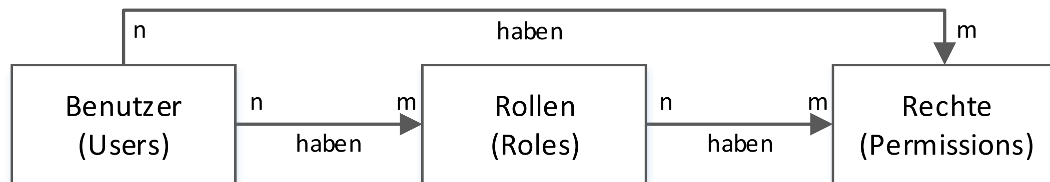



**Abbildung 5.3.:** Realisierung der Autorisierung von Benutzern

Bei der Authentifizierung sendet der Benutzer über eine SSL[2] gesicherte Verbindung, eine Benutzername-Passwort-Kombination an den Server. Dieser testet, ob diese valide ist. Bei Bejahung wird ein Authentifizierungstoken und ein Refresh-Token generiert. Der Authentifizierungstoken ist eine begrenzte Zeit gültig und muss im Header jeder HTTP-Nachricht mitgesendet werden. Ist dieser abgelaufen, kann mithilfe des Refresh-Token ein neuer Authentifizierungstoken generiert werden.

Alle Daten, die von der Benutzer- und Rechteverwaltung benötigt werden, sind in einer relationalen Datenbank gespeichert.

Die Verwaltung von Benutzern, Rollen und Rechten kann über eine REST-Schnittstelle erledigt werden. Da ein Benutzer authentifiziert sein muss, um diese Verwaltung durchzuführen, muss ein initialer Benutzer mit entsprechenden Rechten von Hand in der Datenbank angelegt werden.

### Technischer Aufbau der Benutzer- und Rechteverwaltung

Abbildung 5.4 zeigt ein vereinfachtes Klassendiagramm der Benutzer- und Rechteverwaltung. Alle gezeigten Klassen befinden sich im Paket `usermanagement` (siehe Abbildung 5.2). Das Diagramm zeigt nur den Zusammenhang der Klassen innerhalb des Pakets, weswegen aus Übersichtsgründen auf eine Visualisierung von Methoden, Feldern, und so weiter verzichtet wurde. Zudem wurden Assoziationen mit Klassen außerhalb des Pakets nicht berücksichtigt. Die verschiedenen Farbtöne der Assoziationen haben keine Bedeutung und dienen ausschließlich zur besseren Verständlichkeit des Diagramms. Diverse Klassen besitzen ein + in der unteren Rechte Ecke. Dieses Symbol weist darauf hin, dass hier jeweils mehrere Klassen zusammengefasst wurden um die Komplexität des Diagramms zu senken. Zusammengefasst wurden ausschließlich Klassen, die einen ähnlichen Zweck haben.

Auf die Benutzer- und Rechteverwaltung kann auf zwei verschiedene Wege zugegriffen werden – entweder über die nach außen hin erreichbare REST-Schnittstelle oder über die interne Fassade.

Möchte man als Endbenutzer die Verwaltung verwenden, so kann auf diese über die zuvor erwähnte REST-Schnittstelle zugegriffen werden. Die REST Endpunkte sind in den Klassen

[2]Secure Socket Layer (SSL) verschlüsselt den Datenverkehr zwischen zwei Kommunikationspartner.

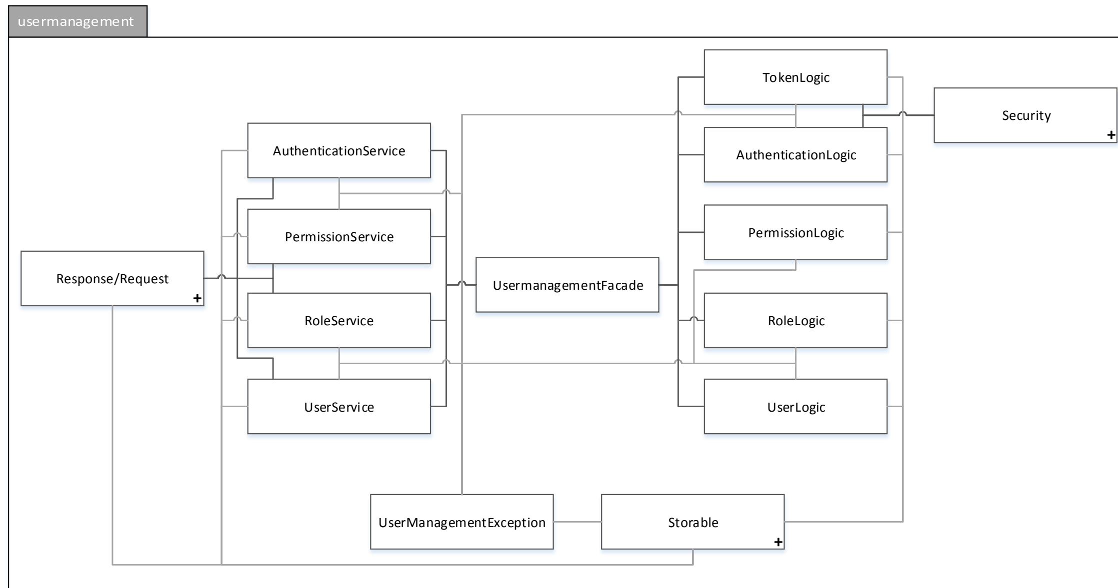



**Abbildung 5.4.:** Vereinfachtes Klassendiagramm der aus [SFB+15] übernommenen Benutzer- und Rechteverwaltung

`AuthenticationService`, `PermissionService`, `RoleService` und `UserService` zu finden. Folgend werden die wichtigsten Methoden, Eigenschaften und Endpunkte dieser Klassen erläutert.

- **`AuthenticationService`**. Diese Klasse implementiert Endpunkte unter dem Pfad `auth`, die die Authentifizierung eines Benutzers übernehmen. Ein Benutzer kann sich über diese ein- und ausloggen.
- **`PermissionService`**. Über die in dieser Klasse implementierten Endpunkte können die Rechte (Permissions) des kompletten Systems verwaltet werden. Hierzu gehört das anlegen, löschen, aktualisieren und abfragen von Rechten. Alle in dieser Klasse implementierten Endpunkte sind über den Pfad `permissions` erreichbar.
- **`RoleService`**. Die Endpunkte in dieser Klassen ermöglichen es Rollen anzulegen, zu löschen, zu aktualisieren und abzufragen. Zusätzlich können alle einer Rolle zugeordneten Rechte abgefragt werden. Alle in dieser Klasse implementierten Endpunkte sind über den Pfad `roles` erreichbar.
- **`UserService`**. Diese Klasse ermöglicht es Benutzer anzulegen, zu löschen, zu aktualisieren und abzufragen. Zudem können die Rollen und die Rechte eines Benutzers abgefragt werden. Alle in dieser Klasse implementierten Endpunkte sind über den Pfad `users` erreichbar.

Die verschiedenen Services implementieren die jeweilige Funktionalität nicht in den Service Klassen. Jegliche Logik befindet sich in den Klassen, mit der Endung *Logic* (zum Beispiel `TokenLogic`). Die Service Klassen greifen nicht direkt auf diese Logik Klassen zu. Alle Zugriffe gehen über die Klasse `UsermanagementFacade`, welche wiederum die Anfragen an die jeweilige Logikklassen zur Bearbeitung übergibt.

Die Klasse `UsermanagementFacade` ist ein Singleton und kann auch außerhalb des Pakets `usermanagement` verwendet werden um auf die Funktionalität der Benutzer- und Rechteverwaltung zuzugreifen.

Ein weiterer wichtiger Teil des Klassendiagramms sind die sogenannten *Storable-Klassen*. Ein Storable ist ein Objekt, das sich in der Datenbank speichern lässt. Dies ist beispielsweise ein Benutzer oder eine Rolle. Dadurch, dass diese Objekte die zu verarbeitenden Informationen des Projekts enthalten, sind diese wie in dem Diagramm zu sehen an vielen Stellen vertreten. Häufig werden diese Storable-Objekte direkt von Endpunkten der Service-Klassen an den Benutzer zurückgegeben. In einigen Fällen repräsentieren diese nicht die vom Endpunkt gewünschten Informationen. Aus diesem Grund gibt es die sogenannten *Response* und *Request* Klassen, die die jeweils benötigten Informationen enthalten.

Tritt während der Verarbeitung einer Anfrage innerhalb des `usermanagement` Pakets ein unerwarteter Fehler auf, so wird dieser in einer `UserManagementException` gekapselt. Die Fehlernachricht wird verhältnismäßig allgemein gehalten, um kein Angriffsvektoren durch zu genau Fehlermeldungen zu eröffnen.

### 5.2.4. REST Schnittstelle für Taxonomien

Das Hauptziel dieser Arbeit ist es Taxonomien verwalten und warten zu können. Zudem sollen diese über gegebene Vorschläge erweiterbar sein. Um diese Funktionalitäten zu verwirklichen, wird eine Schnittstelle nach außen benötigt. Wie in 5.2.3 erwähnt, wurde die Benutzer- und Rechteverwaltung aus einem anderen Projekt übernommen. Innerhalb dieses Projektes wurde eine auf REST basierende Schnittstelle zur Kommunikation mit Clients verwendet. Hierdurch konnten auch die grundlegenden Strukturen, wie beispielsweise Konfigurationen, für eine neue REST Schnittstelle mit übernommen werden.

Zusätzlich zu dem verringerten initialen Aufwand bei der Verwendung einer REST Schnittstelle für Taxonomien, hat diese Art eines Web Services den Vorteil, dass diese verhältnismäßig einfach von verschiedenen Clients verwendet werden kann. Dies ermöglicht es zusätzliche Einsatzgebiete für eine Taxonomie zu finden. Aus diesen Gründen, wurde für diese Arbeit eine REST Schnittstelle zur Verwaltung von Taxonomien gewählt.

#### Pfade

Erreichbar ist die Taxonomieschnittstelle unter dem Pfad `taxonomies`. Folgend werden die wichtigsten Methoden/Pfade der Schnittstelle näher erläutert.

*Anmerkung:* Mit einem `GET` auf eine Ressource wird immer die Ressource selbst als Antwort erwartet. Mit einem `POST` wird gleichzeitig immer eine Instanz einer Ressource mitgesendet, damit diese auf dem Server angelegt werden kann. Teile eines Pfads die mit geschweiften Klammern umgeben sind, sind variable Teile des Pfads und können sich je nach Anfrage unterscheiden. Im Regelfall wird dies genutzt um eine einzelne Ressource zu identifizieren. Manche der Ressourcen unterstützen die sogenannte *Pagination*. Hierbei wird es dem Client ermöglicht nur einen Teil der in der Ressource enthaltenen Einträge abzufragen. Dies hat den Vorteil, dass weniger Daten übertragen und verarbeitet

werden müssen und somit prinzipiell eine beliebig große Menge von Einträgen in einer Ressource vorhanden sein können, ohne eine Überlastung des Systems zu verursachen.

- **GET/POST `/taxonomies`.** Ermöglicht es alle vorhandenen Taxonomien abzufragen oder eine neu anzulegen. Bei einem `GET` auf diese Ressource werden ausschließlich Metadaten, wie zum Beispiel die Bezeichnung oder die Standardsprache der Taxonomie, gesendet.
- **POST `/taxonomies/{taxId}/import`.** Über diesen Pfad kann eine neue Taxonomie in das System importiert werden. Eine genaue Beschreibung des Importvorgangs ist in Kapitel 5.2.6 und Kapitel 5.3.3 zu finden.
- **GET `/taxonomies/{taxId}/export`.** Taxonomien, die im System vorhanden sind, können über diesen Pfad exportiert und in das Ausgangsformat zurück gewandelt werden. Eine genaue Beschreibung des Exportvorgangs ist in Kapitel 5.2.7 und Kapitel 5.3.3 zu finden.
- **GET `/taxonomies/{taxId}/suggestions`.** Sind für eine Taxonomie unbearbeitete Vorschläge vorhanden, so können diese über diesen Pfad abgefragt werden.
- **GET `/taxonomies/{taxId}/languages`.** Die Synonyme einer Taxonomie können in verschiedenen Sprachen vorliegen. Um eine Übersicht über die verwendeten Sprachen zu bekommen, können alle in der Taxonomie zum Einsatz kommenden Sprachen über diesen Pfad abgefragt werden.
- **GET `/taxonomies/{taxId}/languages/default`.** Je nach Taxonomie kann diese sehr viele verschiedene Sprachen enthalten. Um einem Client die Anzeige der Synonyme in einer einheitlichen Sprache zu ermöglichen, besitzt jede Taxonomie eine Standardsprache, die über diesen Pfad abgefragt werden kann.
- **GET/POST `/taxonomies/{taxId}/concepts`.** Jede Taxonomie besteht aus Konzepten. Über diesen Pfad können alle Konzepte einer Taxonomie abgefragt werden. Um die Datenmenge bei sehr großen Taxonomien beherrschbar zu halten, wird Pagination unterstützt.
- **GET `/taxonomies/{taxId}/concepts/root`.** Um die Struktur einer Taxonomie zu erkunden, wird eine Ausgangspunkt benötigt. Hierfür wird im Allgemeinen das Wurzelkonzept des Taxonomiebaums beziehungsweise Graphen verwendet. Dieser Pfad liefert immer genau ein Konzept zurück. Dieses markiert die Wurzel der Taxonomie.
- **GET/POST `/taxonomies/{taxId}/concepts/{conceptId}/children`.** Um die Struktur einer Taxonomie abfragen zu können, ist es essentiell die Kindkonzepte eines bestimmten Konzeptes abfragen zu können. Unterstützt Pagination.
- **GET/POST `/taxonomies/{taxId}/concepts/{conceptId}/parents`.** Die Navigation durch den Taxonomiebaum beziehungsweise Graphen erfordert es an vielen Stellen, dass man die Elternkonzepte eines bestimmten Konzeptes kennt. Genau dies kann mit diesem Pfad erreicht werden. Unterstützt Pagination.
- **GET/POST `/taxonomies/{taxId}/concepts/{conceptId}/synsets`.** Jedes Konzept besitzt eine beliebige Menge von Synsets. Dieser Pfad ermöglicht es die Synsets eines bestimmten Konzepts abzufragen. Unterstützt Pagination.

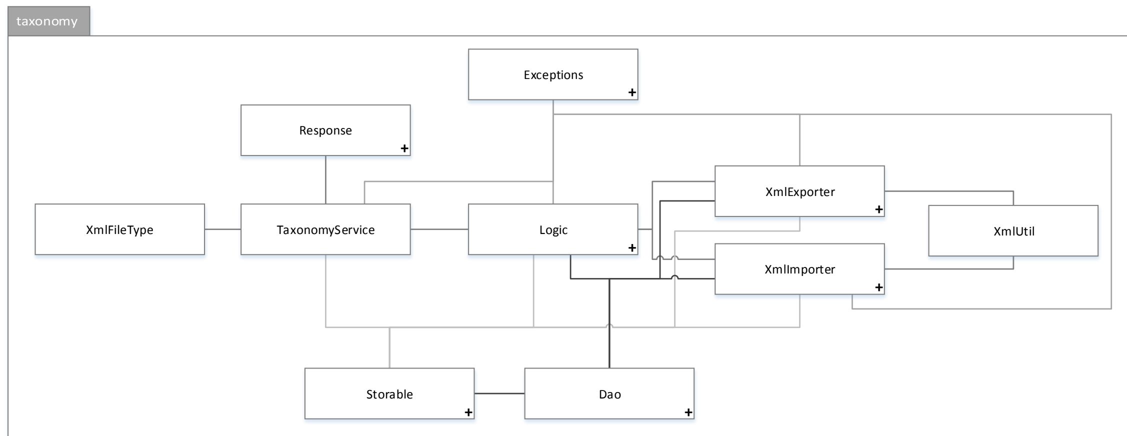



**Abbildung 5.5.:** Vereinfachtes Klassendiagramm des Paketes `taxonomy`

- **GET/POST `/taxonomies/{taxId}/concepts/{conceptId}/synsets{synsetId}/synonyms`.** Synsets besitzen beliebig viele Synonyme, welche über diesen Pfad abgefragt werden können. Unterstützt Pagination.
- **GET/POST `/taxonomies/{taxId}/concepts/{conceptId}/synsets{synsetId}/synonyms {synonymId}/tokens`.** Ein Synonym setzt sich aus beliebig vielen Tokens zusammen, die über diesen Pfad abgefragt werden können. Unterstützt Pagination.

### Technische Betrachtung

Anhand des in Abbildung 5.5 zu sehenden Klassendiagramms soll der Aufbau des Taxonomiepaketes näher erklärt werden. Dieses Diagramm veranschaulicht ausschließlich den Zusammenhang zwischen den einzelnen Klassen, weswegen auf die Visualisierung einzelner Methoden und Attribute verzichtet wurde. Zudem wurden diverse Klassen, die prinzipiell denselben Aufbau und die selbe Funktionsweise besitzen, zusammengefasst. Zur Verdeutlichung, dass mehrere Klassen zusammengefasst wurden, befindet sich im rechten unteren Eck einer Klasse ein +. Die verschiedenen Farben der Assoziationen haben keine weitere Bedeutung, sondern dienen nur zur besseren Visualisierung.

Die Klasse `TaxonomyService` enthält die Implementierung der zuvor vorgestellten REST Schnittstelle. In Abbildung 4.2 wurde erläutert, dass eine Taxonomie ursprünglich in zwei verschiedenen XML-Dateien vorliegt. Um beim Exportieren, beziehungsweise Importieren einer Taxonomie feststellen zu könnem, welche der XML-Dateien momentan übergeben wird, gibt es die Enumeration `XmlFileType`. Mit Hilfe dieser kann die entsprechende Datei definiert werden. Die einzelnen Entitäten einer Taxonomie, wie beispielsweise Konzepte oder Synset, werden mit sogenannten `Storable` Klassen abgebildet. In vielen Fällen kann vom `TaxonomyService` direkt ein Objekt, beziehungsweise eine Liste von Objekten einer `Storable` Klasse an das Jersey Framework zur Serialisierung dessen übergeben werden. In manchen Fällen kann eine REST Antwort allerdings nicht über eine `Storable` Klasse erzeugt werden. Für diesen Fall gibt es gesonderte `Response` Klassen.

Die Klasse `TaxonomyService` selbst enthält keine Logik, um deren Umfang überschaubar zu halten. Die eigentliche Logik ist in den `Logic` Klassen enthalten. Für verschiedene Entitäten, wie beispielsweise Konzepte oder Synsets existiert eine eigene Logikklasse. Jegliche Logik, die sich nicht direkt einer Entität zuordnen lässt, befindet sich in der Klasse `TaxonomyLogic`. Sollen Objekte persistent gespeichert werden, so müssen diese in die Datenbank geschrieben werden. Diese Speicherung übernehmen die `Dao` (Data Access Object) Klassen. Hierbei gibt es für jede Entität eine eigene Klasse. Eine genauere Erläuterung der persistenten Speicherung von Objekten ist in Kapitel 5.2.5 zu finden.

Klassen, die für das Importieren einer neuen Taxonomie zuständig sind, befinden sich in `XmlImporter`. Für das Exportieren zuständige Klassen hingegen befinden sich in `XmlExporter`. Methoden, die sowohl beim Importieren, als auch beim Exportieren benötigt werden, befinden sich in der Klasse `XmlUtil`. Tritt währen der Verarbeitung von Anfragen ein unerwarteter Fehler auf, so gibt es diverse `Exceptions`, die diese Fehler genauer spezifizieren.

#### Bestimmung des Anzeigenamens eines Konzepts

Da ein Konzept mehrere Synonyme besitzen kann, muss für die Anzeige eines Konzepts ein sinnvoller Anzeigename ermittelt werden. Hierbei ergibt sich die Schwierigkeit, dass in den ursprünglichen Daten kein dediziertes Synonym als Anzeigename definiert ist. Aus diesem Grund werden innerhalb einer einzelnen SQL-Abfrage logisch die folgenden Schritte durchgeführt:

1. Selektiere alle Konzepte, die in Verbindung mit den gesuchten Konzepten stehen.
2. Ordne jedem Synonym eine der Sprache entsprechende Priorität zu. Die Priorität einer Sprache ist in der Tabelle `languages` definiert
3. Ordne das Zwischenergebnis nach der ID der Konzepte und der aufsteigenden Priorität
4. Gruppiere alle Ergebnisse mit der gleichen Konzept-ID. Hierdurch wird ausschließlich der Name des Synonyms übernommen, das die höchste Priorität hat

### 5.2.5. Persistierung der Daten

Zur Speicherung der Daten wird eine relationale Datenbank verwendet. Mit Daten werden sowohl die zum Benutzer- und Rechtemanagement gehörenden, als auch die einer Taxonomie zuzuordnenden Dinge bezeichnet. In diesem Unterkapitel wird zuerst das Datenbankmodell anhand eines Entitiy-Relationship-Modells (ER-Modell) vorgestellt. Anschließend wird darauf eingegangen, wie die Datenbankkommunikation innerhalb der Serverkomponente gehandhabt wird. Zum Schluss wird auf die Speicherung eines azyklischen gerichteten Graphen in einer relationalen Datenbank eingegangen.

### Datenbankmodell

Abbildung 5.6 zeigt das Datenbankmodell als ER-Modell. Dieses kann grob in zwei verschiedene Bereiche eingeteilt werden. Der obere Teil enthält Entitäten, die der Benutzer- und Rechteverwaltung zugeordnet werden können. Im unteren Teil befinden sich die Entitäten, die zur Abbildung von Taxonomien in der Datenbank notwendig sind. Um die Komplexität des Diagramms niedrig zu halten, wurde auf die Abbildung von Attributen verzichtet. Ein vollständiges Diagramm mit allen Attributen ist in Anhang A.1 zu finden.

Wie bereits in Kapitel 5.2.3 erläutert, enthält die Benutzer- und Rechteverwaltung die Entitäten *Benutzer*, *Rolle* und *Berechtigung*. Diese werden in der Datenbank mit den Entitäten `user`, `role` und `permission` abgebildet. Da zwischen den Entitäten *n zu m* Beziehungen bestehen, wurden die Entitäten `users_permission`, `roles_permission` und `users_role` eingeführt. Zusätzlich zu Rollen und Rechten besitzt ein Benutzer Tokens, die zur Authentifizierung dessen verwendet werden. Dieser Teil des Datenbanklayouts wurde weitestgehend von [SFB+15] übernommen. Im Original besteht eine Relation zwischen einem Token und einer Rolle. Diese wurde für die vorliegende Arbeit als nicht relevant erkannt und entfernt.

Der Aufbau des unteren Teils des Diagramms richtet sich nach dem in Kapitel 4.2 vorgestellten Format für Taxonomien. Die Grunddaten, wie beispielsweise der Name oder die Standardsprache, einer Taxonomie sind in der Entität `taxonomie` enthalten. Für jede Taxonomie werden die verfügbaren Sprachen einzeln in der Entität `langauge` vorgehalten. Eine Taxonomie besteht aus vielen Konzepten und diese können wiederum aus verschiedenen Synsets bestehen. Ein Synset enthält mehrere Synonyme, welche aus Tokens aufgebaut sind. Die Relationen zwischen den einzelnen Konzepten werden in der Entität `relation` abgebildet. Konzepte, Synsets, Synonyme, Tokens und Relationen können eine beliebige Anzahl von Attributen besitzen, welche mit der Entität `attribute` abgebildet werden. Ein wesentlicher Teil dieser Arbeit ist die Verarbeitung von gegebenen Vorschlägen für neue Synonyme eines Konzepts. Diese werden gesondert generiert [Est15] und direkt in die Tabelle `suggestion` geschrieben. Obwohl mit den Vorschlägen neue Synonyme vorgeschlagen werden, sind diese in der Datenbankstruktur nicht direkt mit einem Synset assoziiert. Dies hat den Hintergrund, dass die Vorschläge bei der externen Generierung ausschließlich einem Konzept und nicht einem Synset zugeordnet werden.

Die Entität `exportId` wird beim Exportieren einer Taxonomie benötigt. Diese enthält IDs, die zur Generierung eines Downloadlinks verwendet werden. Eine genauere Betrachtung der Exportfunktion ist in Kapitel 5.2.7 zu finden.

### Technische Betrachtung

Die Architektur des Datenbankmanagements soll anhand des in Abbildung 5.7 zu sehenden Klassendiagramms gezeigt werden. Hierbei wird ausschließlich auf den Zusammenhang zwischen den einzelnen Klassen eingegangen. Der genaue Aufbau einer Klasse würde wenig zum Verständnis der Architektur beitragen, weshalb auf eine Darstellung von Attributen und Methoden verzichtet wird.

Die hier vorgestellte Architektur besteht prinzipiell aus zwei verschiedenen Teilen. Zum einen befinden sich im Paket `db` allgemeine Klassen für das Datenbankmanagement und zum anderen befinden sich

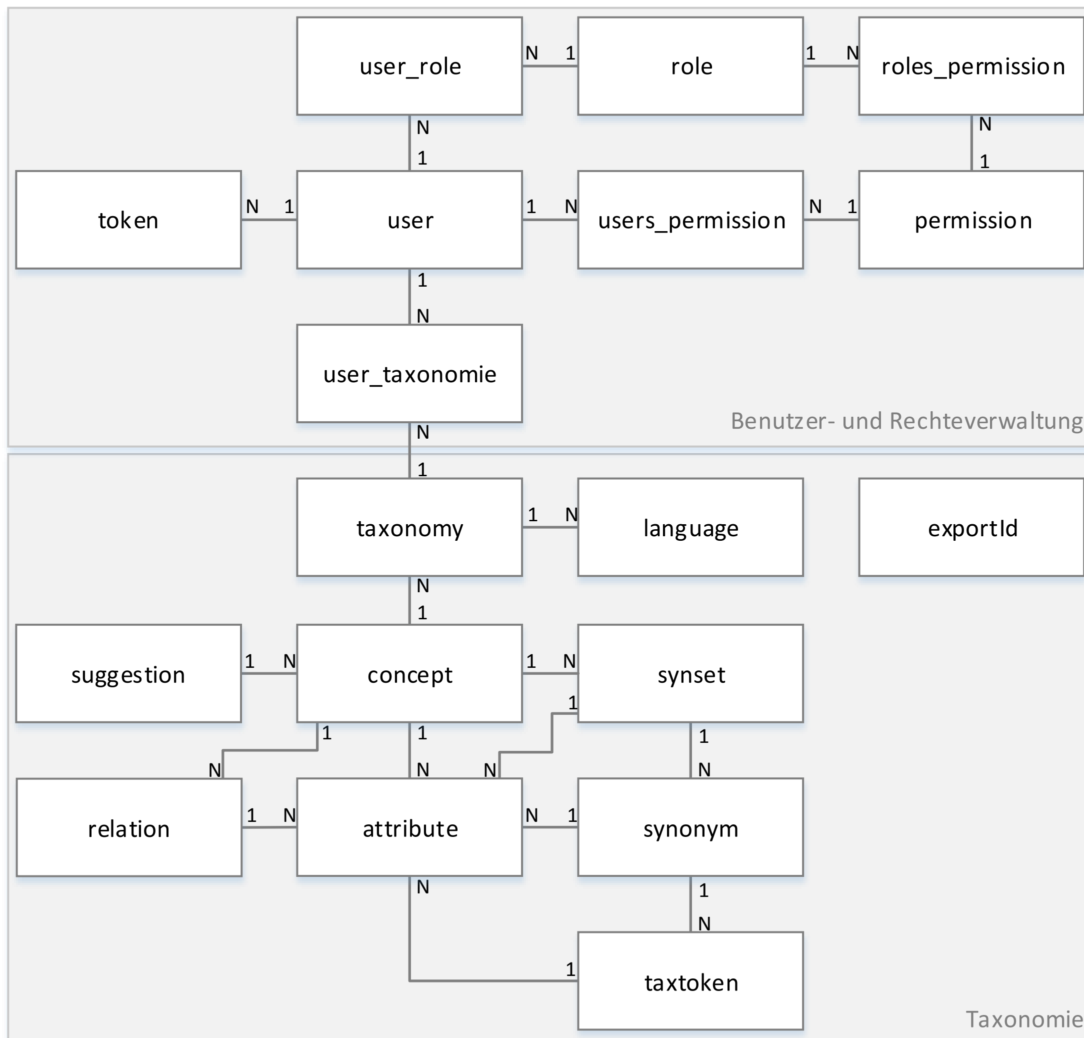



**Abbildung 5.6.:** Vereinfachtes Entity-Relationship-Diagramm der Datenbankstruktur

in jedem Modul Spezialisierungen der allgemeinen Klassen. Als Modul wird das Benutzer- und Rechtemanagement (Paket `usermanagement`) und auf die Taxonomien bezogene Klassen (Paket `taxonomy`) gesehen.

Das Datenbankmanagement baut auf zwei zentralen Elementen auf. Zum einen gibt es die Schnittstelle `DAO`, die spezifiziert über welche Methoden auf die Datenbank zugegriffen werden kann. Als Parameter benötigt diese Schnittstelle den zweiten zentralen Teil – ein `Storable`. Ein `Storable` ist eine Entität, die als Tabelle in der Datenbank vorliegt. Die Klasse selbst ist abstrakt, weshalb nur Spezialisierungen dieser Klasse zum Interagieren mit der Datenbank verwendet werden können. Solche Spezialisierungen sind in diesem Fall zum Beispiel die Klassen `Taxonomy`, `Concept` und `Synset`. In

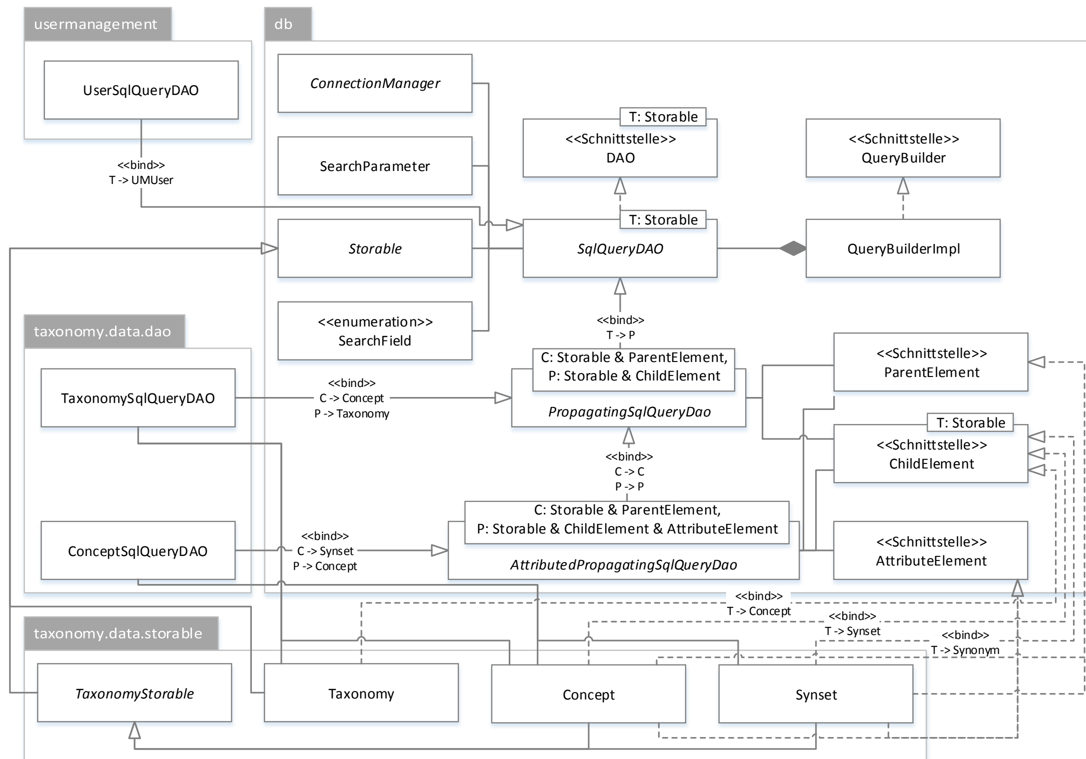



**Abbildung 5.7.:** Vereinfachtes Klassendiagramm der Klassen, die für das persistente Speichern der Daten zuständig sind

den Paketen `usermanagement` und `taxonomy` gibt es noch weitere von `Storable` abgeleitete Klassen, die zugunsten eines kompakteren Diagramms nicht dargestellt sind.

Die Schnittstelle `DAO` ermöglicht es dynamisch Anfragen zu generieren. Um dem System mitzuteilen, in welcher Spalte, nach welchem Wert gesucht werden soll, kann die Klasse `SearchParameter` verwendet werden. Ein SearchParameter erwartet ein `SearchField`, welches die zu durchsuchende Spalte innerhalb einer Datenbanktabelle angibt.

Implementiert wird die Schnittstelle `DAO` momentan ausschließlich von der abstrakten und generischen Klasse `SqlQueryDAO`. Innerhalb dieser Klassen werden für ein `Storable` entsprechende SQL-Abfragen generiert und an die Datenbank gesendet. Die Antwort der Datenbank wird zur weiteren Verarbeitung in ein Objekt umgewandelt. Die Generierung der Abfragen wird von der Klasse `QueryBuilderImpl` übernommen, die die Schnittstelle `QueryBuilder` implementiert. Die abstrakte Klasse `ConnectionManager` wird von den DAO Klassen benutzt, um über die enthaltenen statischem Methoden eine Verbindung zur Datenbank aufzubauen. Durch den Einsatz dieser Klasse ist es möglich mehrere Anfragen in einer Transaktion zu bündeln, um auf aufgetretene Fehler reagieren zu können.

Um den hierarchischen Aufbau einer Taxonomie besser abbilden zu können, wurden die abstrakten SqlQueryDao Spezialisierungen, PropagatingSqlQueryDao (folgend als PSQD bezeichnet) und AttributedPropagatingSqlQueryDao (folgend als APSQD bezeichnet) eingeführt. Ein PSQD ermöglicht es dynamisch, über Rekursion, alle Kindelemente in eine Anfrage miteinzubeziehen. Um diese Funktion bereit zu stellen werden zwei verschiedene Parameter benötigt. Der Parameter C gibt an von welchem Typ die Kindelemente sind. Ein Kindelement muss von der Klasse Storable ableiten und die Schnittstelle ChildElement implementieren. In dem Diagramm wird dies von den Klassen Concept und Synset umgesetzt. Wie in Kapitel 4.2 erläutert, besitzt ein Konzept mehrere Synsets, weshalb das ChildElement der Klasse Concept die Klasse Synset ist. Analog verhält es sich bei Synsets mit Synonymen. Das Diagramm zeigt nicht alle vorhandenen Storable-Klassen, da dies die Komplexität des Diagramms unnötig steigern würde. Der Parameter P eines PSQDs definiert den Typ des Elternelements. Dieses muss von der Klasse Storable ableiten und die Schnittstelle ParentElement implementieren. Im Diagramm wird dies zum Beispiel von den Klassen Taxonomy, Concept und Synset umgesetzt.

Viele Entitäten innerhalb einer Taxonomie haben zu einem gewissen Teil die gleichen Eigenschaften. So besitzen die meisten Klassen zum Beispiel das Attribut deleted, das angibt ob die Entität gelöscht ist. Um keine Code-Klone für diese gemeinsamen Eigenschaften zu schaffen, wurde die abstrakte Klasse TaxonomyStorable eingeführt. Von TaxonomyStorable leiten alle Klassen ab, die diese Eigenschaften teilen.

Eine weitere Spezialisierung eines PSQD ist ein APSQD. Ein APSQD enthält zusätzlich zur Handhabung von hierarchischen Strukturen, die Möglichkeit Attribute einer Entität zu verarbeiten, insofern diese bei der entsprechenden Entität vorhanden sind. Vorhanden sind diese zum Beispiel bei Konzepten und Synsets, weshalb hier auf APSQDs gesetzt wird.

Die in dem Paket db enthaltenen DAO-Klassen sind alle Abstrakt. Aus diesem Grund befinden sich in den verschiedenen Modulen für jede Storable-Klasse eine eigene DAO-Klasse. So enthält die Benutzer- und Rechteverwaltung beispielsweise eine KlasseUserSqlQueryDAO, die eine Verarbeitung von Benutzern ermöglicht. Alle im Paket usermanagement enthaltenen DAO-Klassen leiten direkt von SqlQueryDAO ab. Alle im Paket taxonomy enthaltene DAO-Klassen die keine Verbindung zum hierarchischen Aufbau der Taxonomie haben, leiten direkt von SqlQueryDAO ab. Klassen, die wie zum Beispiel die Klasse Taxonomy zwar ein Teil der hierarchischen Struktur der Taxonomie sind, allerdings keine Attribute besitzen, haben eine DAO-Klasse die von PSQD ableitet (siehe TaxonomySqlQueryDAO). Hat eine Entität, beziehungsweise Klasse, Attribute, so leitet deren DAO-Klasse von APSQD ab (siehe ConceptSqlQueryDAO).

Die DAO- und Storable Klassen des Paketes usermanagment, sowie die Enumeration SearchField, die Schnittstellen DAO, QueryBuilder und die Klassen ConnectionManager, SearchParameter, Storable, SqlQueryDao, QueryBuilderImpl des Pakets db wurden aus einem vorhergehenden Projekt [SFB+15] übernommen Zum Teil haben große Anpassungen der Klassen stattgefunden.

### Speicherung der Relationen zwischen den Konzepten einer Taxonomie

Alle Relationen einer Taxonomie werden in der Tabelle relations innerhalb der Datenbank persistent gespeichert. Bei einem einfachen Modell werden ausschließlich die Kanten des Graphes gespeichert,

die zu einem direkten Elternknoten führen. Abbildung 5.8a zeigt solch ein azyklischen gerichteten Graphen. Wenn ausschließlich die Relation zum direkten Elternknoten gespeichert wird, so ist das auslesen eines Pfades zum Wurzelknoten nur über eine rekursive Abfrage möglich. Möchte man zum Beispiel alle Pfade zur Wurzel des Knotens `E` bekommen, so müssen insgesamt drei Abfragen durchgeführt werden:

- Elternknoten von `E` = `B` und `C`
  - Elternknoten von `B` = `A`
  - Elternknoten von `C` = `A`

Umso größer ein Graph ist, umso aufwändiger wird die Bestimmung eines Pfades. Um die Benutzererfahrung zu verbessern ist eine möglichst schnelle Verarbeitung von Anfragen erstrebenswert. Aus diesem Grund wird bei der Realisierung auf die Repräsentation der Relationen mittels einer transitiven Hülle gesetzt. Beim Einsatz einer transitiven Hülle werden alle Pfade zu den einzelnen Knoten des Graphen, die auf dem Weg zum Wurzelknoten liegen, einzeln gespeichert [DLSW99]. Dies ermöglicht es anstatt über eine rekursive Abfrage, über eine einzelne Abfragen die genauen Pfade zum Wurzelknoten zu bekommen. Eine Visualisierung des Prinzips ist in Abbildung 5.8b zu sehen. Wenn nun eine Anfrage nach den Elternknoten von `E` gestellt wird, so wird die Antwort `A`, `B`, `C` geliefert. An dieser Stelle ergibt sich das Problem, dass auf diese Weise nicht zwischen direkten und indirekten Elternknoten unterschieden werden kann. Aus diesem Grund wird zusätzlich zu jeder Relation die Anzahl der Sprünge (Hops), die zum Erreichen eines Knotens benötigt werden, gespeichert. Der Knoten `B` beispielsweise lässt sich von `E` direkt erreichen, weshalb Null Hops benötigt werden. `A` hingegen lässt sich nur über die Knoten `B` oder `C` erreichen, weshalb hier die Anzahl der Hops von `E` aus gesehen Eins beträgt.

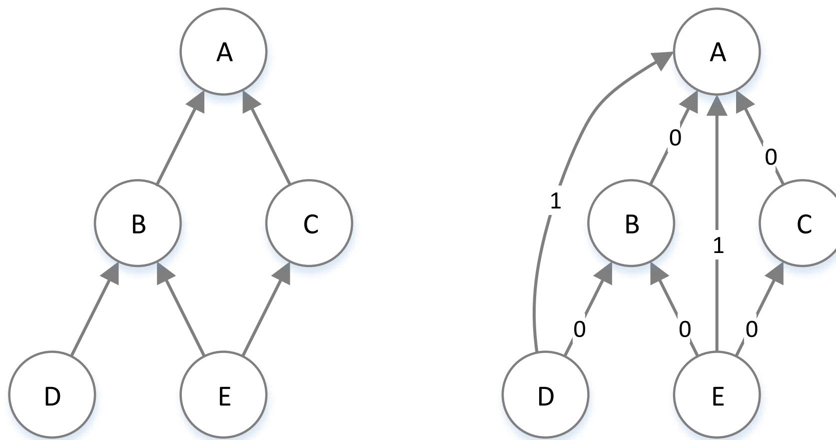
### 5.2.6. Importieren vorhandener XML-Daten

Das System setzt es voraus, dass Taxonomien serverseitig in der Datenbank gespeichert sind. Um eine aufwändige manuelle Eingabe der Daten zu vermeiden, wurde eine Funktion zum Importieren vorhandener Taxonomien implementiert. Zuerst wird die Vorgehensweise des Importvorgangs erläutert und anschließend findet eine technische Betrachtung statt.

#### Vorgehensweise

Importiert werden können solche Taxonomien, die in dem in Kapitel 4.2 beschriebenen Format vorliegen. Da der Inhalt einer Taxonomie auf zwei verschiedene XML-Dateien aufgeteilt ist, muss beim Importieren strikt die in Abbildung 5.9 zu sehende Vorgehensweise eingehalten werden:

1. Zu Beginn muss eine leere Taxonomie erstellt werden, in welche die Daten importiert werden können. Hierzu muss eine POST-Anfrage an den Endpunkt `taxonomies` gesendet werden. Der Rumpf der Anfrage muss den zukünftigen Namen der Taxonomie enthalten. Nach dem erfolgreichen Erstellen der Taxonomie, wird diese als Objekt zurückgegeben. Der wichtigste Teil dieses Objektes ist die enthaltene ID, welche bei den folgenden Schritten benötigt wird.




**(a)** Gerichteter azyklischer Graph ohne eine transitive Hülle

**(b)** Gerichteter azyklischer Graph mit einer transitiven Hülle

**Abbildung 5.8.:** Gerichteter azyklischer Graph mit und ohne einer transitiven Hülle

2. Nachdem erfolgreich eine leere Taxonomie erstellt wurde, kann deren Inhalt importiert werden. Angenommen es wurde zuvor eine Taxonomie mit der ID `1` erstellt, so muss eine POST-Anfrage an den Endpunkt `taxonomies/1/import` mit dem URL-Paramter `type=DATA` gesendet werden. Als Content-Type muss *multipart/form-data* verwendet werden. Mit einem Content-Type, beziehungsweise auch als MIME-Type oder Internet Media Type bezeichnet, werden die Daten im Rumpf einer Nachricht klassifiziert. *multipart/form-data* wurde entwickelt um binäre Daten, wie beispielsweise PDFs, über ein HTML-Formular übertragen zu können [MX98]. Der Parameter `type` mit dem Wert `DATA` weist den Server an die in einer **_data.tax* gespeicherten Daten zu importieren. Je nach Größe der zu importierenden Taxonomie kann der Vorgang wenige Sekunden bis Minuten benötigen. Nach erfolgreichem Beenden des Vorgangs sind alle Konzepte, Synsets, Synonyme, Tokens und Attribute der Taxonomie in der Datenbank vorhanden. Relationen zwischen den einzelnen Konzepten sind an diesem Punkt noch nicht vorhanden. Um diese zu erhalten muss der nächste Schritt durchgeführt werden. Der komplette Importvorgang wird in einer Transaktion ausgeführt, sodass bei einem Fehler kein inkonsistenter Zustand entsteht.

3. Ist der Importvorgang der **_data.tax* erfolgreich können die Relationen zwischen den Konzepten hergestellt werden. Im alten Format (siehe Kapitel 4.2) sind diese in der XML-Datei mit der Bezeichnung **_forschung.tax* gespeichert. Um die Relationen serverseitig zu erstellen, muss die **_forschung.tax*-Datei mit dem MIME-Type `multipart/form-data` in einer POST-Anfrage an den Enpunkt `taxonomies/1/import` mit dem URL-Parameter `type=RELATIONS` gesendet werden. Nachdem dieser Vorgang erfolgreich abgeschlossen ist, enthält die Datenbank die komplette Taxonomie. Aus dem gleichen Grund, wie in Schritt 2., wird auch dieser Teil des Importvorgangs in einer Transaktion ausgeführt.

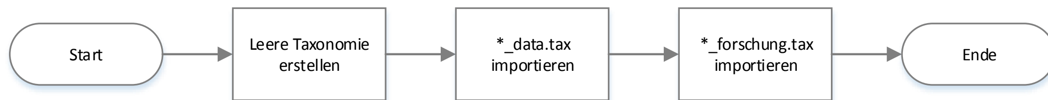



**Abbildung 5.9.:** Vorgehensweise beim Importieren einer Taxonomie

### Technische Betrachtung

Nachdem eine Taxonomie an den Server übertragen wurde, durchläuft diese verschiedene Schritte der Bearbeitung. Im ersten Schritt wird über den URL-Parameter `type` ermittelt, ob die übergebene Datei den Daten oder den Relationen der Taxonomie entspricht. Je nach Art werden diese anschließend unterschiedlich verarbeitet.

Abbildung 5.10 zeigt in einem Klassendiagramm die Klassen, die für das Importieren einer Taxonomie verantwortlich sind. An diversen Stellen wurde das Zeichen $*$ verwendet. Wird dies eingesetzt, so wurden gleichartige Attribute und Methoden zusammengefasst, um das Diagramm zu vereinfachen.

Unabhängig von der Art der übergebenen Daten, ist die Klasse `XmlParser` der Ausgangspunkt. Um die Daten einer Taxonomie zu importieren, muss die Methode `parseAndInsertData()` aufgerufen werden. Innerhalb dieser wird eine Instanz des `XMLStreamReaders` erstellt, welcher eine Implementierung des *Streaming API for XML (StAX)* darstellt. StAX ist ein sogenanntes *PULL-API*, das nach dem Cursorverfahren arbeitet. Dies bedeutet, dass beim Benutzen des API selbstbestimmt das nächste zu verarbeitende Element angefordert werden kann. Der Vorteil dieses API ist es, dass die Daten Stück für Stück verarbeitet werden und nicht vollständig im Hauptspeicher vorliegen müssen. Dies ermöglicht es auch sehr große Taxonomien zu verarbeiten.

Normalerweise wird über die Methode `next()` des `XMLStreamReaders` das nächste Element der XML-Datei abgefragt. Mit dieser Vorgehensweise ist es allerdings nicht möglich zeitgleich den Aufbau der XML-Datei mit einem XML-Schema zu überprüfen. Um gleichzeitig mit dem Validieren die Daten verarbeiten zu können, wird eine Spezialisierung der Klasse `StreamReaderDelegate` verwendet.

Ein `StreamReaderDelegate` ermöglicht es in die Art und Weise, wie ein `XMLStreamReader` arbeitet einzugreifen und über das Überschreiben von diversen Methoden Einfluss auf die Verarbeitung der Daten zu nehmen. Somit enthält die Klasse `XmlDataParserDelegate` die Logik zum Verarbeiten der Daten einer Taxonomie. Intern wird ein Zustandsautomat verwendet, der die Verarbeitung der XML-Elemente steuert. Die möglichen Zustände sind in der Enumeration `State` festgehalten. Die Bezeichnung aller in den XML-Dateien verwendeter Elemente und Attribute sind in der Klasse `XmlUtil` gespeichert, da diese an anderen Stellen, wie beispielsweise beim Exportieren der Daten erneut benötigt werden. Das aktuelle XML-Element wird sofort in die Datenbank geschrieben und anschließend aus dem Hauptspeicher gelöscht, um diesen nicht unnötig zu belasten. Alle Datenbankabfragen werden in einer Transaktion durchgeführt, um im Fehlerfall keinen inkonsistenten Zustand in der Datenbank vorliegen zu haben. Nachdem das `XmlDataParserDelegate` erfolgreich die Daten

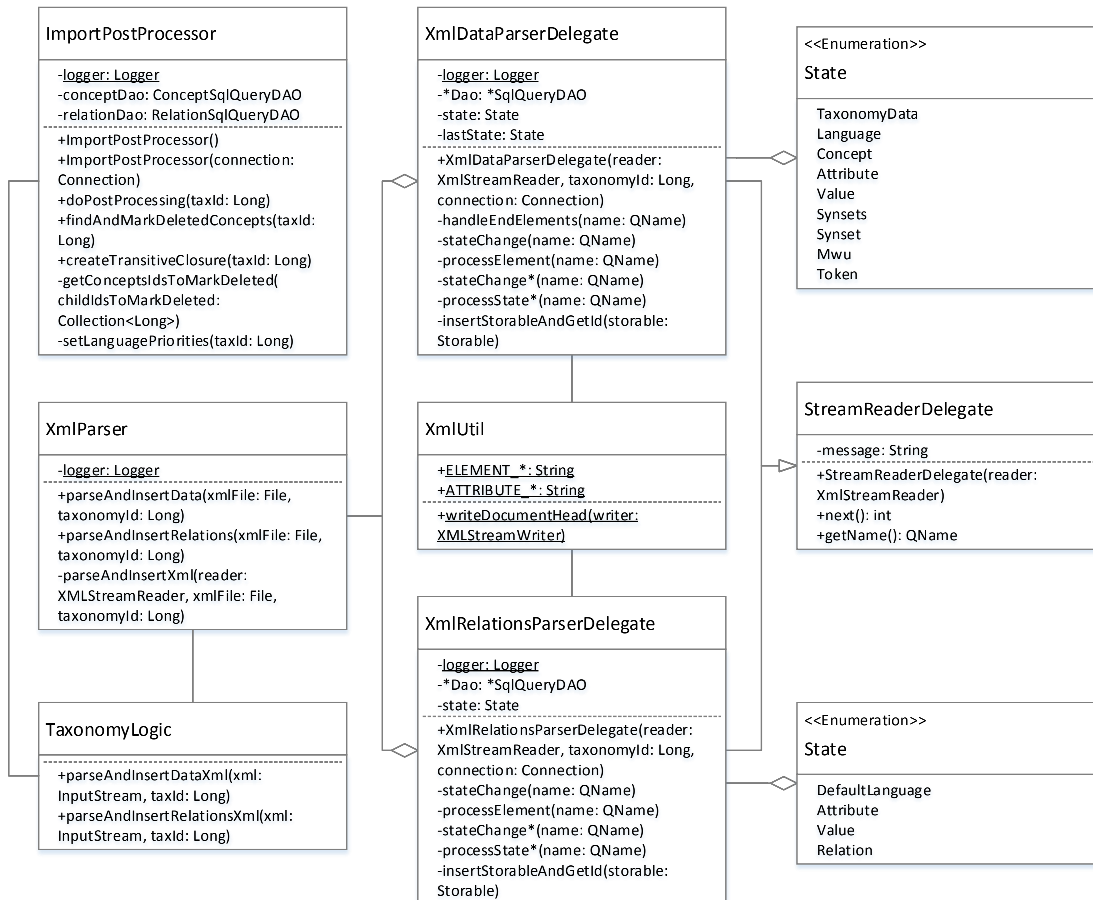



**Abbildung 5.10.:** Klassendiagramm der Klassen, die den Importvorgang von Taxonomien handhaben

importiert hat, ist dieser Vorgang abgeschlossen. Um eine vollständige Taxonomie zu erhalten müssen im nächsten Schritt die Relationen importiert werden.

Um die Relationen zu importieren wird in der `TaxonomyLogic` die Methode `parseAndInsert-RelationsXml()` aufgerufen, welche wiederum die Methode `parseAndInsertRelations()` der Klasse `XmlParser` aufruft. In dieser wird eine Instanz des `XmlRelationsParserDelegate` erstellt und ausgeführt. Wie beim zuvor vorgestellten `XmlDataParserDelegate` werden die Daten über einen Zustandsautomaten verarbeitet. Prinzipiell ist die Funktionsweise der beiden Delegate-Klassen dieselbe. Nachdem die Relationen erfolgreich importiert wurden, finden zusätzliche Schritte statt, um die Repräsentation der Taxonomie in der Datenbank zu verbessern.

In der als XML-Dateien vorliegenden Taxonomien sind sowohl gelöschte als auch nicht gelöschte Konzepte enthalten. In manchen Fällen wurde ausschließlich ein Elternelement als gelöscht markiert und nicht zusätzlich dessen Kindelemente. Dies führt zu einer logischen Inkonsistenz, die über die Methode `findAndMarkDeletedConcepts()` der Klasse `ImportPostProcessor` beseitigt wird. In einem weiteren Schritt der Nachbearbeitung einer Taxonomie, wird die in Kapitel 5.2.5 beschriebene

transitive Hülle des Taxonomiegraphen erstellt. Nachdem diese Nachbearbeitungsschritte erfolgreich ausgeführt wurden, ist die Taxonomie komplett importiert und kann ohne Einschränkungen verwendet werden.

### 5.2.7. Exportieren der Daten

Nachdem eine Taxonomie in die Datenbank importiert wurde, kann diese nicht mehr mit zuvor auf dem XML-Format basierenden Methoden und Tools verwendet werden. Um diese Nutzung einer Taxonomie erneut zu ermöglichen, wurde eine Funktion zum Exportieren einer Taxonomie implementiert. In diesem Unterkapitel wird zuerst die Vorgehensweise beim Exportieren betrachtet. Anschließend findet eine rein technische Betrachtung des Exportsystems statt.

#### Vorgehensweise

Die Daten und Relationen einer Taxonomie können unabhängig voneinander in das in Kapitel 4.2 vorgestellte Format exportiert werden. Hierzu sind vom Benutzer jeweils zwei verschiedene Schritte durchzuführen.

1. Über den REST Endpunkt `taxonomies/taxId/export` mit den URL-Parameter `type` kann je nach zu exportierendem Typ ein passender Downloadlink generiert werden. Der zu exportierende Typ wird über den URL-Parameter `type` festgelegt, welcher die Werte `DATA` oder `RELATIONS` besitzen darf. Dieser Endpunkt kann nur verwendet werden, wenn ein Benutzer authentifiziert ist. Der Parameter `{taxId}` gibt die ID der zu exportierenden Taxonomie an.
2. Nachdem der Downloadlink generiert wurde, kann dieser ohne Authentifizierung zum Exportieren der gewünschten Datei verwendet werden. Jeder Link kann nur einmal benutzt werden, da er nach der ersten erfolgreichen Verwendung unbrauchbar gemacht wird.

Die Gründe für diese zweistufige Vorgehensweise liegen bei der verwendeten Technologie und deren Limitierungen auf der Clientseite. Eine detaillierte Beschreibung der Limitierung und deren Umgehung kann in Kapitel 5.3.3 gefunden werden.

Durch die Generierung eines ohne Authentifizierung nutzbaren Downlaodlinks, ergibt sich der sekundäre Vorteil, dass dieser an beliebige Personen weitergegeben werden kann. Diese Personen brauchen somit kein Benutzerkonto innerhalb des Systems um einmalig eine Taxonomie herunterzuladen.

#### Technische Betrachtung: Generierung des Downloadlinks

Folgend findet eine technische Betrachtung der zuvor vorgestellten Schritte zum Exportieren einer Taxonomie statt. Zu Beginn muss ein Downloadlink für den Benutzer generiert werden. Der hierfür erforderliche serverinterne Vorgang ist in Abbildung 5.11 abgebildet.

Im ersten Schritt wird eine Anfrage des Benutzers über die REST Schnittstelle erhalten. Durch den URL-Parameter `type` kann festgestellt werden, ob die Daten oder die Relationen einer Taxonomie exportiert werden sollen. Anschließend wird eine eindeutige ID generiert, die zum Erstellen des Downloadlinks

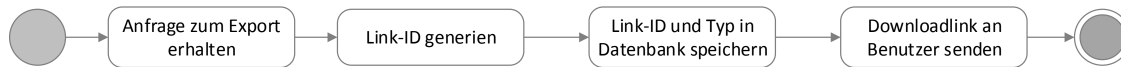



**Abbildung 5.11.:** Vorgehensweise zur Generierung eines Downloadlinks für den Export einer Taxonomie

verwendet wird. Da REST zustandslos ist, muss die generierte Link-ID in der Datenbank gespeichert werden. Zusätzlich zu der ID wird der Typ der zu exportierenden Datei gespeichert, sodass das System beim Aufrufen des Downloadlinks die richtige Art von Daten exportiert. Nachdem die Daten erfolgreich in der Datenbank gespeichert wurden, wird aus der Link-ID ein Downloadlink generiert und an den Benutzer zurückgegeben.

Beim Aufrufen des Downloadlinks wird die Link-ID extrahiert und mit den Einträgen in der Datenbank verglichen. Befindet sich in der Datenbank ein gültiger Eintrag, so wird der Exportvorgang gestartet. Nachdem das Exportieren erfolgreich beendet ist, wird der Eintrag aus der Datenbank gelöscht, sodass der Downloadlink kein zweites Mal verwendet werden kann.

#### Technische Betrachtung: Generierung der zu exportierenden Dateien

Anhand des in Abbildung 5.12 zu sehenden Klassendiagramms soll die Generierung der zu exportierenden Dateien genauer betrachtet werden. Der Exportvorgang beider Typen wird von der Klasse `XmlParser` gehandhabt. Wie beim in Kapitel 5.2.6 beschriebenen Importieren von Taxonomien, wird auch hier auf die StAX-API gesetzt. Dies hat den selben Hintergrund wie beim Importieren – die Größe einer Taxonomie soll nicht durch den verfügbaren Hauptspeicher begrenzt werden.

Gestartet wird die Generierung der Daten über die Methode `createDataXml()` und die der Relationen über die Methode `createRelationsXml()`. Über die `write*Element()` Methoden werden die einzelnen Elemente erstellt, wobei das Zeichen $*$ für jeweils ein Element, wie beispielsweise `Concept` steht. Die Namen der XML-Attribute und Elemente sind in der Klasse XmlUtil gespeichert. Geschrieben werden die Daten über einen Writer, wobei hier im Speziellen eine Instanz der Klasse `OutputStreamWriter` verwendet wird. Der so erzeugte Stream wird direkt zum Benutzer gesendet.

### 5.2.8. Behandlung von Fehlern

Ein wichtiger Punkt bei der Erstellung von Software ist der Umgang mit unerwarteten Fehler. Im optimalen Fall wird ein Benutzer durch einen Fehler kaum bis nicht beeinträchtigt. Falls ein Benutzer beeinträchtigt wird, sollte dieser nach Möglichkeit eine informative Fehlermeldung erhalten. Folgend soll erläutert werden, wie der im Rahmen dieser Arbeit erstellte Serverteil mit unerwarteten Fehler umgeht.

Ein Benutzer kann mit dem Server ausschließlich über die angebotene REST-Schnittstelle kommunizieren. Infolgedessen können Fehlermeldungen auch ausnahmslos über diese Schnittstelle an

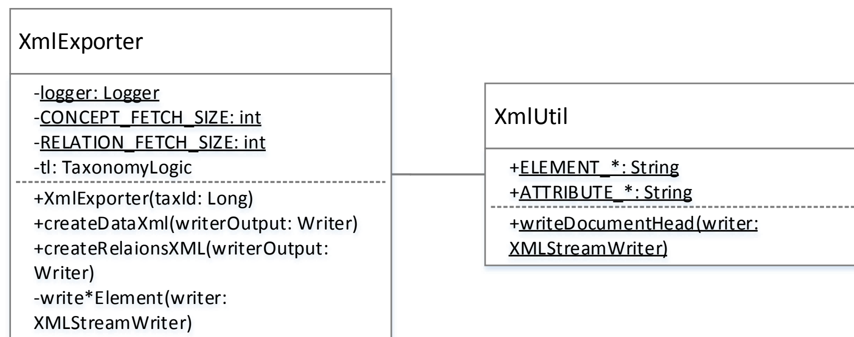



**Abbildung 5.12.:** Klassendiagramm der Klassen, die den Exportvorgang von Taxonomien handhaben

den Benutzer übermittelt werden. Wie in Kapitel 3.1 erläutert findet die Kommunikation über das HTTP-Protokoll statt. Dieses Protokoll spezifiziert sogenannte *Status Codes*, die die Art einer Antwort spezifizieren.

Status Codes die mit einer zwei beginnen, geben an, dass eine Anfrage erfolgreich war. Die hierbei am häufigsten verwendeten Codes sind 200 (OK), 201 (Created) und 204 (No Content). Ist eine Anfrage nicht erfolgreich und es tritt ein Fehler auf, so werden Status Codes die mit vier beziehungsweise fünf anfangen verwendet.

Beginnt ein Status Code mit einer vier, so liegt der Grund für die nicht erfolgreiche Anfrage beim Client. Beginnt er mit einer fünf, so liegt der Fehler auf der Seite des Servers. Folgend werden die im Rahmen dieser Arbeit am häufigsten verwendeten Fehler Status Codes näher erläutert.

- **`400 Bad Request.`** Eine Anfrage des Clients kann nicht beantwortet werden, da die Syntax der Anfrage falsch ist. Dieser Fehler kann beispielsweise durch eine falsche URL oder durch nicht zur URL passende Nutzdaten ausgelöst werden. Ein Client sollte erst nachdem die Anfrage angepasst wurde, diese nochmals an den Server senden.
- **`401 Unauthorized.`** Der Benutzer kann nicht identifiziert werden. Dieser Fehler tritt im Allgemeinen auf, wenn ein Nutzer nicht eingeloggt ist. Nachdem erfolgreichen Einloggen sollte diese Fehlermeldung nicht mehr erscheinen.
- **`403 Forbidden.`** Ein Benutzer ist authentifiziert, sprich eingeloggt, allerdings versucht dieser auf eine Ressource zuzugreifen, für welche dieser keine Berechtigung besitzt. Sobald ein Benutzer eine Berechtigung für die angeforderte Ressource besitzt, sollte diese fehlerfrei verwendet werden können.
- **`404 Not Found.`** Es wird versucht auf eine Ressource zuzugreifen die nicht existiert. Erst nachdem diese erstellte wurde, kann auf diese zugegriffen werden.
- **`500 Internal Server Error.`** Innerhalb des Servers tritt ein unerwarteter Fehler auf, der nicht ordnungsgemäß behandelt wird. Die Anfrage des Clients ist in diesem Fall korrekt und

muss nicht korrigiert werden. Ein Client kann dieselbe Anfrage mehrfach an den Server stellen, in der Aussicht, dass es sich um einen transienten Fehler handelt.

Wird eine Exception innerhalb des Servers geworfen, so wird ohne weiteres Zutun von GlassFish ein `500 Internal Server Error` generiert und an den Client übermittelt. Dieser enthält zur Fehleranalyse den kompletten Stacktrace der aufgetretenen Exception. Im Allgemeinen ist es nicht erwünscht, dass ein Client diesen Stacktrace zu sehen bekommt. Vielmehr soll eine für den Benutzer aussagekräftige Fehlermeldung generiert werden.

Nun gibt es zwei verschiedene Möglichkeiten die Fehlerbehandlung mithilfe des Jersey-Frameworks durchzuführen. Zum einen können in jedem Endpunkt mit einem `try... catch(...)` Konstrukt mögliche Exceptions abgefangen und entsprechende Status Codes manuell an den Client gesendet werden. Diese Vorgehensweise ist allerdings vergleichsweise aufwändig und erfordert im Regelfall viel redundanten Code. Um diese Redundanz zu vermeiden, bietet das Jersey-Framework sogenannte `ExceptionMapper` an.

Zum anderen können durch einen `ExceptionMapper` vorher definierte Exceptions vom Jersey-Framework behandelt werden. Es kann pro Exception entschieden werden, welche Nachricht und welcher Status Code zu dieser passt. Im Falle eines Fehlers kann somit eine zu dem Fehler passende Nachricht an den Benutzer gesendet werden. Abbildung 5.13 zeigt mit Hilfe eines Klassendiagramms den Aufbau der Verwendeten `ExceptionMapper`.

Die Schnittstelle `ExceptionMapper` wird vom Jersey-Framework bereitgestellt und erwartet ein `Throwable`, sprich eine Exception, als Parameter. Implementiert wird diese Schnittstelle von der abstrakten Klasse `ExceptionMapperBase`, die wiederum ein `Throwable` als Parameter erwartet. Das Attribut `logger` dient zum serverseitigen Aufzeichnen des aufgetretenen Fehlers, sodass dieser in einer Log-Datei nachvollzogen werden kann. Über das Attribut `statusCode` wird der zuvor erwähnte HTTP Status Code festgelegt. Zum Konstruieren eines, auf der `ExceptionMapperBase` basierenden Objektes, ist die Angabe eines Status Codes zwingend notwendig. Alle implementierten Exception-Mapper leiten von der Klasse `ExceptionMapperBase` ab. In diesem Diagramm sind dies die Klassen `XMLExceptionMapper` und `NotFoundExceptionMapper`. Insgesamt gibt es deutlich mehr als diese zwei ExceptionMapper, was durch die Klasse mit der Bezeichnung `...` angedeutet werden soll.

Damit das Jersey-Framework weiß auf welche Exception es reagieren soll, muss über den Parameter `T` der Schnittstelle `ExceptionMapper`, diese definiert werden. Bei der hier vorgestellten Architektur geschieht dies durch die Angabe einer Exception als Parameter beim Ableiten von der Klasse `ExceptionMapperBase`.

Tritt nun eine Exception innerhalb des Servers auf, wird diese vom Jersey-Framework gefangen. Anschließend sucht dieses ob ein zu der gefangenen Exception passender ExceptionMapper existiert. Ist dies nicht der Fall, wird die Exception erneut geworfen, damit andere Teile des Servers auf diese reagieren können. Wird allerdings ein passender ExceptionMapper gefunden, so wird bei diesem die durch die implementierte Schnittstelle `ExceptionMapper` vorhandene Methode `toResponse()` aufgerufen. Innerhalb dieser Methode wird eine zu der Exception passende Fehlermeldung geniert. Innerhalb der vorgestellten Architektur wird ein Objekt der Klasse `ExceptionEntity` erstellt. Diese enthält die Fehlernachricht und den passenden HTTP Status Code. Diese Objekt wird anschließend

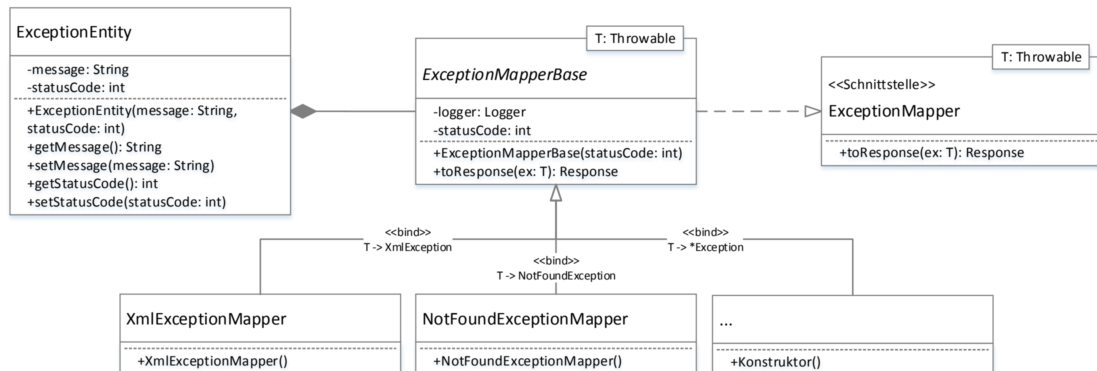



**Abbildung 5.13.:** Klassendiagramm der ExceptionMapper

einem Response Objekt übergeben, welches von Jersey zur Generierung einer validen HTTP Antwort verwendet wird.

## 5.3. Clientseite

Folgend wird der Entwurf und die Realisierung der Clientseite betrachtet. Zu Beginn wird auf die verwendeten Technologien eingegangen. Anschließend wird ein Überblick über den Aufbau des Clients gegeben. Daraufhin wird der Import- und Exportvorgang einer Taxonomie betrachtet. Danach wird die Visualisierung einer Taxonomie und die der Vorschläge für neue Synonyme präsentiert. Hinterher wird auf die Suche innerhalb des Taxonomiebaums eingegangen. Zum Schluss wird erläutert, wie das Kopieren und Verschieben von Konzepten innerhalb einer Taxonomie und die Fehlerbehandlung gehandhabt wird.

### 5.3.1. Verwendete Technologien

Wie in Kapitel 4.4.5 beschrieben, wird für die Realisierung der App die Technologie der Web-App gewählt. Entwickelt werden diese im Allgemeinen mit einer Kombination aus *Hypertext Markup Language (HTML)*, *Cascading Style Sheets (CSS)* und *JavaScript (JS)*. In dieser Arbeit werden zusätzliche Frameworks und Bibliotheken eingesetzt, um den Entwicklungsprozess zu beschleunigen und Standardprobleme mit gut getesteten Komponenten zu lösen. Um die Entwicklung der Programmlogik und das Binden von Daten an eine View zu vereinfachen, wird das in Kapitel 3.2 vorgestellt JavaScript-Framework *AngularJS* eingesetzt. Zusätzlich wird das von der Twitter Inc. entwickelte CSS-Framework *Bootstrap* [OT15] verwendet. Dieses ermöglicht es fertige und optisch schön gestaltete HTML-Komponenten einzusetzen.

Dank des modularen Aufbaus von AngularJS lässt sich dieses einfach um von dritten entwickelnde Module erweitern. Folgend werden die wichtigsten eingesetzten Module näher vorgestellt.

- **AngularUI Bootstrap** kapselt die in Bootstrap vorhandenen Komponenten in AngularJS Direktiven, um diese einfacher innerhalb einer Angular Umgebung nutzbar zu machen.
- **AngularUI Router** ist eine Alternative zu dem bei AngularJS mitgeliefertem Standardrouter. Ein Router ist dafür zuständig, je nach URL die passende View anzuzeigen. Der hier eingesetzte Router hat gegenüber dem Standardrouter den Vorteil, dass dieser auf ineinander verschachtelbaren Zuständen basiert. Dies ermöglicht eine deutlich bessere Strukturierung der App.
- **ngTouch** ist dafür verantwortlich Touch-Gesten innerhalb einer AngularJS Umgebung verfügbar zu machen. Zudem wird der `ng-click`-Handler dahingehende verbessert, dass er die von vielen mobilen Browsern verwendete 300ms Verzögerung entfernt, die zwischen dem Berühren des Bildschirms und dem Auslösen des entsprechenden Events liegt.
- **ngAnimate** ermöglicht es Änderungen in der View, die durch viele Standardkonstrukte, wie beispielsweise `ng-repeat` ausgelöst werden, zu animieren. Die Animation findet rein über CSS statt.
- **Restangular** stellt Funktionen bereit, die eine Kommunikation über eine REST-Schnittstelle vereinfachen. Hierbei wird neben der eigentlichen Kommunikation auch die Authentifizierung gegenüber dem Server vorgenommen.
- **Angular IScroll** ermöglicht es verschiedene Komponenten innerhalb einer View beliebig scrollbar zu machen. Angular IScroll selbst ist nur ein Wrapper für die allgemeine JavaScript Bibliothek *IScroll 5*.

### 5.3.2. Allgemeiner Aufbau der App

Im Folgenden soll der logische Zusammenhang zwischen den einzelnen Views der App erläutert werden. Dies entspricht zugleich auch größtenteils dem technischen Aufbau, da einzelne Views ausschließlich über deren Routen miteinander verbunden sind. Eine Visualisierung des Zusammenhangs ist in Abbildung 5.14 zu sehen.

Beim ersten Aufrufen der App wird man automatisch zur Login-View geleitet. Hier müssen Benutzername und Passwort eingegeben werden, die zur Authentifizierung gegenüber dem Server genutzt werden sollen. Ist diese erfolgreich, so wird man zur Selection-View weitergeleitet. Diese ist, wie die restlichen vorgestellten Views, ein Kind der abstrakten View `taxonomy`. Abstrakte Views haben die Besonderheit, dass diese nicht direkt angezeigt werden, für Kinder allerdings allgemeine Funktionen und HTML-Elemente bereitstellen können. So bindet `taxonomy` zum Beispiel die abstrakte View `navbar` ein, die eine Navigationsleiste in jeder anderen View einblendet, die `taxonomy` als Elternteil hat.

Die View `selection` selbst, ist dafür verantwortlich alle verfügbaren Taxonomien aufzulisten, editier- und selektierbar zu machen. Während des Editierens ist es möglich den Namen und die Standardsprache der Taxonomie zu verändern. Zudem können an dieser Stelle nicht mehr benötigte Taxonomien gelöscht werden. Sollten in dem System noch keine Taxonomien vorhanden sein, so muss eine neue

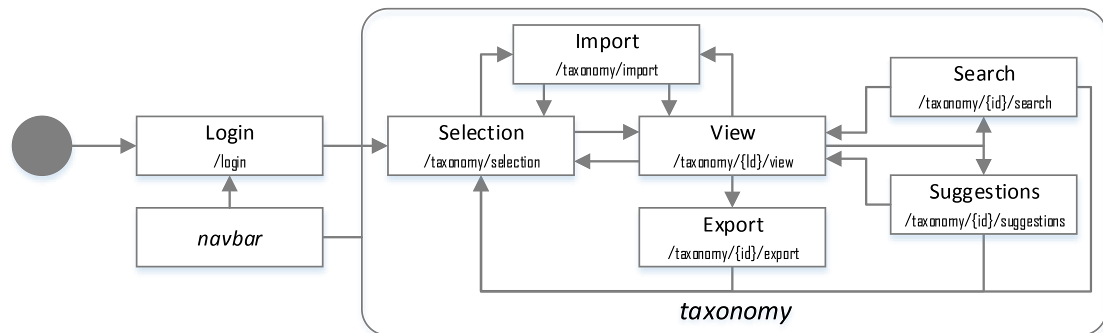



**Abbildung 5.14.:** Logischer Zusammenhang zwischen den einzelnen Views des Clients

erstellt werden. Dies kann in der View `import` erledigt werden. Eine genauere Betrachtung des Erstellen und Importieren von Taxonomien ist in 5.3.3 zu finden.

Wird eine Taxonomie selektiert, so wird der Benutzer in die View `View` weitergeleitet, welche für das Anzeigen und Bearbeiten der Inhalte, sprich der Konzepte, Synsets, Synonyme und Tokens, zuständig ist. Nachdem eine Taxonomie ausgewählt wurde, ist es zudem möglich eine Suche innerhalb dieser durchzuführen. Die Suchergebnisse werden anschließend in der View `Search` dargestellt. Durch die Selektion eines bestimmten Suchergebnisses wird der Benutzer in die View `View` weitergeleitet. Innerhalb der View wird der Pfad von der Wurzel bis zu dem selektierten Konzept geöffnet. Eine detaillierte Erklärung der Suchfunktion ist in Kapitel 5.3.6 zu finden.

Ein wesentlicher Bestandteil dieser Arbeit ist es Vorschläge für neue Synonyme anzuzeigen, einem Benutzer zu präsentieren und diese entweder in die Taxonomie aufzunehmen oder abzulehnen. Dies kann zum einen direkt in der View `View` vorgenommen werden. Zum anderen gibt es die dedizierte View `Suggestions`, die eine Übersicht über alle Vorschläge gibt und das Annehmen oder Ablehnen dieser ermöglicht.

In manchen Anwendungsfällen ist es erstrebenswert die in der Datenbank vorliegende Taxonomie in ein anders Format zu exportieren. Das Exportieren kann über die View `Export` vorgenommen werden, welche sich nach dem Selektieren einer Taxonomie über die Navigationsleiste erreichen lässt.

Das in dieser Arbeit vorgestellte System ermöglicht es mehrere Taxonomien zu verwalten. Um schnell zwischen verschiedenen wechseln zu können, ist es nach dem Einloggen immer möglich die View `Selection` zu erreichen, in welcher eine andere Taxonomie ausgewählt werden kann.

Nach dem Einloggen wird auf jeder View die Navigationsleiste angezeigt. Diese ermöglicht es nicht nur allgemeine Funktionen, wie das Selektieren neuer Taxonomien jederzeit zu ermöglichen, sondern bietet zudem die Möglichkeit einen Benutzer vom System abzumelden. Ein abgemeldeter Benutzer wird zur View `login` geleitet und hat keinen Zugriff mehr auf die Taxonomien. Erst nach dem erneuten erfolgreichen Anmelden ist dieser erneut gegeben.

**Abbildung 5.15.:** Screenshot der View zum Anlegen und Importieren einer Taxonomie. Gezeigt ist der erste Schritt des Importvorgangs

### 5.3.3. Importieren/Exportieren von Taxonomien

In diesem Kapitel wird die Benutzeroberfläche zum Importieren und Exportieren von Taxonomien vorgestellt.

#### Anlegen und Importieren von Taxonomien

Zu Beginn enthält das System keine Taxonomien. Im Allgemeinen werden diese aus in bereits existierenden Taxonomien in das System importiert. Nachdem ein Benutzer authentifiziert ist, kann über die Navigationsleiste die in Abbildung 5.15 View zum Anlegen und Importieren von Taxonomien aufgerufen werden. Gezeigt wird hierbei der erste von drei Schritten die durchgeführt werden müssen um eine Taxonomie zu importieren. Abbildungen der anderen Schritte sind in Anhang A.2 zu finden.

Der komplette Vorgang ist in Form eines Assistenten aufgebaut, der den Benutzer durch den Importvorgang begleitet. Nachdem die View aufgerufen wird, wird immer der erste Schritt angezeigt. Innerhalb dieses Schrittes gibt es verschiedene Möglichkeiten fortzufahren. Zum einen gibt es die Möglichkeit vorhandene Taxonomien mit Daten, beziehungsweise Relationen anzureichern. Hierfür müssen allerdings bestimmte Voraussetzungen gegeben sein. Um eine Taxonomie mit Daten anzureichern, muss diese komplett leer sein. Dies bedeutet, dass weder Daten noch Relationen in dieser existieren dürfen, die mit dieser assoziiert sind. Ist diese Voraussetzung erfüllt, so steht diese Taxonomie zur Auswahl bereit. Enthält eine Taxonomie bereits Daten, aber noch keine Relationen, so steht auch diese zur Auswahl bereit. Da bereits Daten enthalten sind, wird in diesem Fall der zweite Schritt übersprungen und es kann sofort mit Schritt drei fortgefahren werden. Stehen keine

kompatiblen Taxonomien zur Auswahl oder es soll eine neue erstellt werden, so kann dies unter der Angabe des Namens getan werden.

Nachdem Erstellen, beziehungsweise Auswählen einer Taxonomie wird das Panel mit dem ersten Schritt ausgeblendet und das Panel für den zweiten Schritt eingeblendet. Bei diesem muss die Datei, welche die Daten enthält ausgewählt werden. Über den Button *Upload* kann anschließend der Importvorgang gestartet werden. Je nach Größe der Taxonomie kann dieser eine gewisse Zeit benötigen. Während des Vorgangs wird eine Ladeanzeige eingeblendet, die ein verarbeiten der Datei signalisiert. Ein wichtiges Detail, das beim Importvorgang beachtet werden muss ist die Kodierung der zu importierenden Datei. Verarbeitet werden können ausschließlich Dateien, die im *UTF-16 BE (Big Endian)* Format vorliegen.

Ist das Importieren der Daten erfolgreich, wird automatisch das Panel des zweiten Schrittes geschlossen und mit das Panel für den dritten Schritt geöffnet. Bei diesem muss die Datei, welche die Relationen der zuvor importierten Daten enthält, angegeben werden. Erneut kann mit dem Button *Upload* der Importvorgang gestartet werden. Auch dieser Teil des Importvorgangs kann je nach Größe der Taxonomie eine gewisse Zeit in Anspruch nehmen. Zudem muss die zu importierende Datei im *UTF-16 BE (Big Endian)* Format vorliegen. Ist der Vorgang erfolgreich, so findet eine Weiterleitung in die View `View` statt, welche die neu importierte Taxonomie anzeigt.

#### Exportieren von Taxonomien

Manche Anwendungsfälle erfordern es eine Taxonomie, in dem in Kapitel 4.2 vorgestellten Format vorliegen zu haben. Aus diesem Grund wurde eine Möglichkeit zum Export derselben entwickelt. Die Exportfunktion steht für vollständig importierte Taxonomien zur Verfügung, die über die View `Selection` ausgewählt wurde. Zu erreichen ist die View `Export` über die Navigationsleiste. Ein Screenshot der View ist in Abbildung 5.16 zu sehen.

Obwohl die Benutzeroberfläche der View trivial ist, sind es die im Hintergrund ablaufenden Prozesse nicht. Aus der Sicht eines Benutzers wählt man zuerst das Zielformat aus, wobei momentan nur das in Kapitel 4.2 vorgestellte XML-Format unterstützt wird. Anschließend können über die zwei darunter befindlichen Buttons die Daten und Relationen exportiert werden.

Wie bereits erwähnt ist die technische Realisierung des Exportvorgangs nicht so trivial wie die Benutzeroberfläche. Die sich ergebenden Probleme lassen sich auf JavaScript und das AngularJS-Framework zurückführen. Das Hauptproblem ist es, dass JavaScript aus Sicherheitsgründen nicht direkt auf den persistenten Speicher eines Clients zugreifen kann. Aus diesem Grund muss dieser Zugriff dem zugrundeliegenden Browser überlassen werden. Diese Limitierung wird allgemein dadurch Umgangen, dass der Downloadlink direkt dem Browser übergeben wird. Dieser ermittelt nun anhand des MIME-Type, wie weiter mit der Datei verfahren werden soll. Liegt die Zieldatei beispielsweise im PDF-Format vor, kann diese im Allgemeinen direkt im Browser angezeigt werden. Hat der Browser keine Möglichkeit die Datei direkt zu verarbeiten, wird diese dem Benutzer zum Herunterladen angeboten.

JavaScript ermöglicht es über die Methode `window.open(URL)` direkt einen Link an den Browser zu übergeben und dieser verarbeitet diesen wie zuvor angegeben. Diese Methode hat den Nachteil, dass

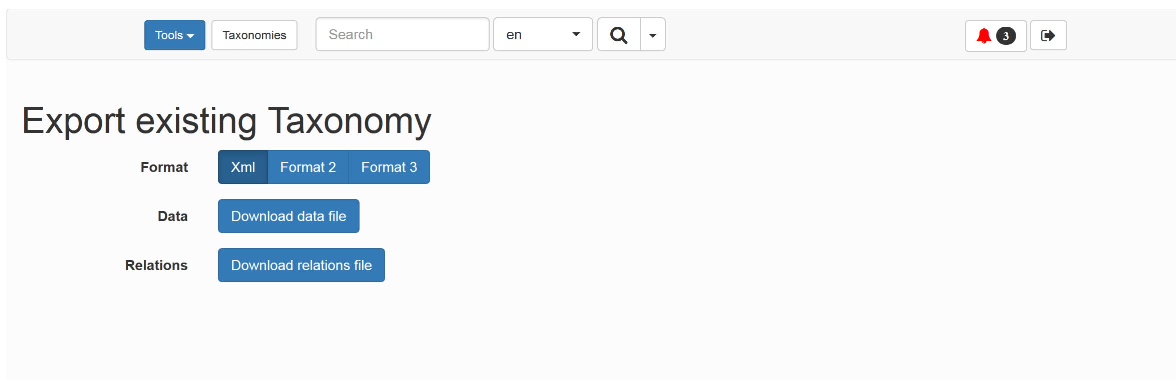



**Abbildung 5.16.:** Screenshot der View zum Exportieren einer Taxonomie

zwar der Download startet, dem Benutzer nach dem Start des Downloads allerdings eine leere Seite angezeigt wird. Dieser hat nun nicht mehr die App vor sich und kann nur über den Zurück-Button des Browser zu dieser zurückkehren. Eine weitere Möglichkeit ist es dem Anchor-Tag (`<a>` Tag) den Downloadlink zuzuweisen. Nachdem ein Benutzer diesen Link benutzt, startet der Download, allerdings wird auch hier dem Benutzer eine leere Seite angezeigt. Um das Problem der leeren Seite zu umgehen, kann ein Inlineframe (`<iframe>` Tag) verwendet werden. Ein Inlineframe ermöglicht es beliebige andere URLs innerhalb der aktuellen Webseite anzuzeigen. An dieser Stelle soll keine andere Webseite angezeigt werden, stattdessen soll ein Downloadvorgang gestartet werden. Aus diesem Grund wird hier ein Inlineframe mit einer Größe von 1px mal 1px eingesetzt, welcher vom Benutzer nicht erkannt werden kann.

Durch die Verwendung des Inlineframes ergibt sich nun ein weiteres Problem – die Authentifizierung des Benutzers gegenüber dem Server. Diese findet dadurch statt, dass im Header der HTTP-Nachricht ein Authentifizierungstoken mitgesendet wird. Beim direkten Öffnen einer URL sind keine Möglichkeiten vorgesehen zusätzliche Informationen an den Header anzuhängen. Dies führt dazu, dass Downloads nur von ungeschützten URLs möglich sind. JavaScript im Gegenzug bietet die Möglichkeit Informationen an den Header der HTTP-Nachricht anzuhängen, wobei sich hier das zuvor genannte Problem ergibt. Eine weitere Möglichkeit ist es über eine asynchrone Anfrage (Ajax-Request) an den Server, den Download zu starten und am Ende die fertig heruntergeladenen Daten an den Browser zur Speicherung auf dem Datenträger zu übergeben. Diese Technik hat im Gegensatz zu den anderen vorgestellten den Nachteil, dass der Benutzer nicht den Fortschritt des Downloads nachverfolgen kann. Die anderen Techniken übergeben den Downloadvorgang an den Browser, wodurch dieser genau durch den Benutzer verfolgt werden kann. Ein weiterer Nachteil der JavaScript-Methode ist es, dass der komplette Inhalt der Taxonomie zuerst im Hauptspeicher des Clients zwischengespeichert wird. Hierdurch ergibt sich zwangsweise eine an dem verfügbaren Hauptspeicher gebundene Limitierung der Größe einer zu exportierenden Datei.

Um eine Limitierung des Hauptspeichers auszuschließen und eine durch das Anzeigen des genauen Verlaufs des Downloads verbesserte Benutzererfahrung, wird in dieser Arbeit auf die Methode mit dem Inlineframe gesetzt. Diese erfordert es allerdings, dass ein Downloadlink ohne eine Authentifizierung verwendet werden kann. Aus diesem Grund wurde der in Kapitel 5.2.7 beschriebene zweiteilige Vorgang entwickelt. Zuerst wird über den geschützten REST-Endpunkt `taxonomies/taxId/export`

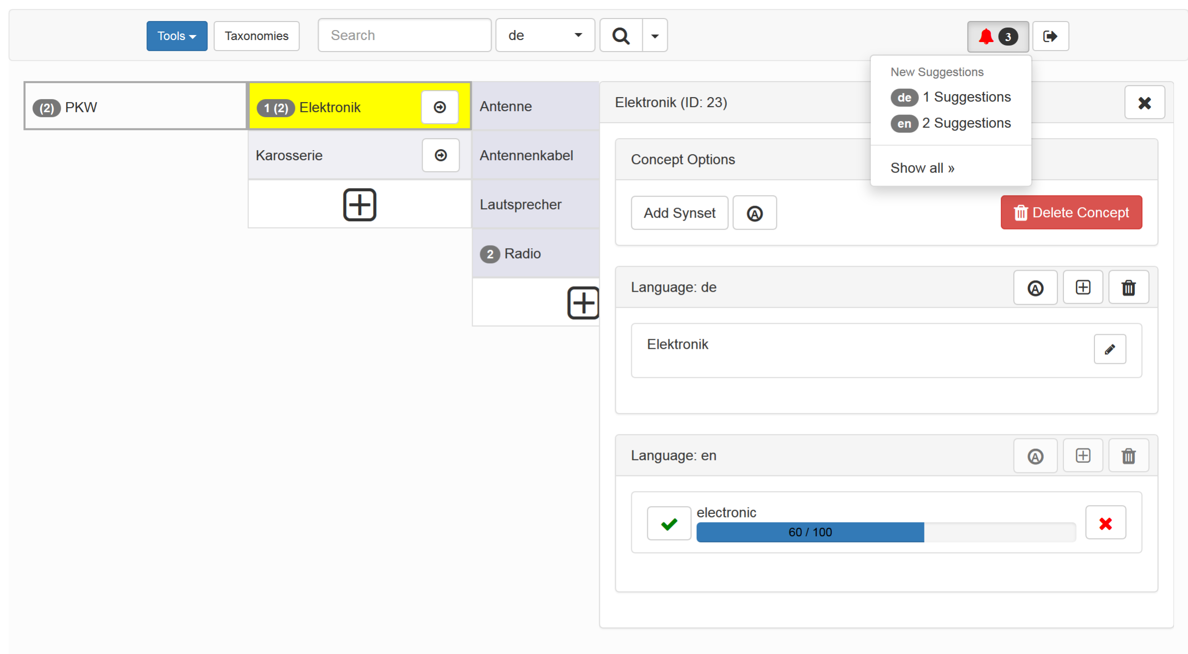



**Abbildung 5.17.:** Screenshot der Hauptview. Auf der linken Seite ist der Taxonomiebaum und auf der rechten die Details eines Konzepts zu sehen

ein Downloadlink angefordert, der nicht durch eine Authentifizierung geschützt ist. Dieser Link wird anschließend dem zuvor erwähnten Inlineframe zugewiesen, wodurch der Downloadvorgang beginnt. Nachdem der Download erfolgreich war, ist der zuvor generierte Link ungültig und es muss für den nächsten Downloadvorgang ein neuer Link generiert werden.

### 5.3.4. Visualisierung der Taxonomie

Ein wichtiger Bestandteil dieser Arbeit ist es eine Taxonomie auf einem Tablet anzuzeigen und ein Editieren dieser zu ermöglichen. Dies wurde in der View mit der Bezeichnung `View` umgesetzt. Zur Visualisierung des Taxonomiebaums wird die auf der linken Seite von Abbildung 5.17 gezeigte und in Kapitel 4.5.2 vorgestellte Side-by-Side Liste verwendet.

Folgend sollen der Aufbau und die Funktionsweise der View `View` anhand des in Abbildung 5.17 zu sehenden Screenshots näher betrachtet werden. Die Abbildung wird im Folgenden in drei verschiedene Teile unterteilt. Am oberen Ende befindet sich die Navigationsleiste, auf der linken Seite der Taxonomiebaum und rechts der im Folgenden als Detailansicht bezeichnete Teil.

Die Navigationsleiste wird in allen Views, mit Ausnahme der View `Login`, angezeigt. Von links nach rechts bietet diese folgende Möglichkeiten:

- Unter dem Menüpunkt *Tools* befinden sich Werkzeuge und Funktionen, die vergleichsweise selten benötigt werden. Hierzu zählt zum Beispiel die Umschaltung in den Vollbildmodus und das Importieren/Exportieren von Taxonomien.

- Über den Button *Taxonomies* lässt sich jederzeit auf die View `Selection` wechseln, um eine neue Taxonomie auszuwählen.
- Die nächsten drei Kontrollelemente gehören zu der logischen Einheit der Suche. In dem dargestellten Textfeld lässt sich der Suchbegriff eingeben. Die daneben liegende Combobox ermöglicht es die bei der Suche zu beachtenden Sprachen zu spezifizieren. Hierbei können beliebig viele oder keine bestimmte Sprache angegeben werden. Ist nichts ausgewählt, werden alle Sprachen bei der Suche berücksichtigt. Über den Button mit der abgebildeten Lupe lässt sich der Suchvorgang starten. Wurde zuvor eine Suche durchgeführt, so ist es über den kleinen Pfeil neben der Lupe möglich auf diese erneut zuzugreifen. Eine genauere Betrachtung der Suchfunktion ist in Kapitel 5.3.6 zu finden.
- Die rote Glocke signalisiert, dass für diese Taxonomien unbearbeitete Vorschläge existieren. Die Zahl neben der Glocke zeigt die Gesamtzahl der Vorschläge über alle Sprachen hinweg an. Wird der Button benutzt, so öffnet sich ein Popover-Element, das die Vorschläge nach Sprachen gruppiert anzeigt. In dem gezeigten Screenshot existieren beispielsweise insgesamt drei Vorschläge, wobei ein Vorschlag für die deutsche und zwei Vorschläge für englische Sprache existieren. Entweder können nun alle oder nach Bedarf auch nur die Vorschläge einer bestimmten Sprache in einer eigenen View angezeigt werden. Eine Betrachtung der View für neue Vorschläge ist in Kapitel 5.3.5 zu finden.
- Am rechten Rand der Navigationsleiste befindet sich ein Button, welcher zum Ausloggen aus dem System verwendet werden kann. Wird dieser Button benutzt, so wird der Benutzer zu der View `Login` weitergeleitet und hat erst nach einem erneuten erfolgreichen Login Zugriff auf Taxonomien.

Auf der linken Seite des Screenshots befindet sich der Taxonomiebaum. Nachdem eine Taxonomie in der View `Selection` ausgewählt wurde, wird nach dem Öffnen der hier vorgestellten View automatisch die ersten zwei Ebenen der Taxonomie vom Server angefordert und in vertikalen Listen dargestellt. Jedes Rechteck innerhalb einer Liste repräsentiert ein Konzept, wobei an dieser Stelle der vom Server ermittelte Anzeigename eines Konzepts (siehe Kapitel 5.2.4) verwendet wird. Zu diesem Zeitpunkt ist die rechts zu sehende Detailansicht geschlossen und der Baum kann, abgesehen von der Navigationsleiste, den kompletten verfügbaren Platz verwenden.

Um die Kindkonzepte eines bestimmten Konzepts anzuzeigen, muss der Button mit dem Pfeil nach rechts verwendet werden, welcher sich in jedem Konzept-Rechteck befindet. Anschließend wird rechts des Buttons eine neue Liste mit den Kindkonzepten angezeigt. Befindet sich an dieser Stelle bereits ein Liste mit den Kindkonzepten eines anderen Elements, so werden alle rechts befindlichen Listen geschlossen und eine neue Liste mit Kindkonzepten erstellt. Alle Elternteile der gezeigten Konzepte sind mit einem grauen Rahmen markiert. Im Falle des Screenshots sind dies zum Beispiel die Konzepte mit der Bezeichnung *automotive* und *component*. Sollen mehr Listen mit Konzepten angezeigt werden, als die verfügbare Bildschirmfläche ermöglicht, so wird der komplette Block mit Listen horizontal scrollbar.

Viele Konzepte besitzen mehr als ein Synset, die jeweils wiederum mehrere Synonyme enthalten. Zudem besagen die Anforderung an die App, dass die Details eines Konzepts, sprich die Synsets, Synonyme, Tokens und Attribute, editierbar sein sollen. Hierzu wird innerhalb dieser View die

sogenannte Detailansicht eingeführt. Diese befindet sich im rechten Bereich des Screenshots. Die Detailansicht zeigt detaillierte Informationen eines Konzepts. Aufgerufen werden kann diese durch einen Klick auf das Rechteck eines Konzepts im Taxonomiebaum.

Die Detailansicht ist in mehrere Panels aufgeteilt. Das oberste Panel bietet allgemeine Aktionen und Optionen an, die das Konzept betreffen. So können neue Synsets hinzugefügt, Attribute editiert oder das Konzept gelöscht werden. Unterhalb dieses Panels befinden sich die Synsets des Konzepts, wobei jedes Synset in einem eigenen Panel untergebracht ist. Auf der rechten Seite des Kopfs des Synset-Panels befinden sich drei verschiedene Buttons. Über diese lassen sich die Attribute eins Synsets editieren, ein neues Synonym hinzufügen oder das Synset löschen. Im Körper des Synsets befinden sich in einer Liste alle Synonyme dessen. Über den rechts angebrachten Button mit dem Stiftsymbol, lassen sich die Tokens und somit der Namen des Synonyms verändern. Zudem ist es an dieser Stelle möglich die Attribute eines Synonyms, beziehungsweise dessen Tokens zu editieren.

Über den Taxonomiebaum ist es möglich auf einen Blick zu erfassen, wie viele Vorschläge für neue Synonyme es für ein Konzept, beziehungsweise dessen Kinder gibt. Eine eingeklammerte Zahl links neben dem Konzeptnamen gibt die Anzahl der Vorschläge für Kinder dieses Konzeptes an. In dem Screenshot besitzen beispielsweise die Kindkonzepte des Konzept *automotive* insgesamt drei Vorschläge. Ist die Zahl nicht eingeklammert, so bedeutet dies, dass das Konzept selbst die Vorschläge besitzt. Das Konzept *Maintenance and Serivce* zum Beispiel besitzt einen neuen Vorschlag. Dieser kann entweder direkt in der Detailansicht des Konzepts oder in der eigens dafür geschaffenen View (siehe Kapitel 5.3.5) bearbeitet werden.

Der Screenshot zeigt einen Vorschlag für das Konzept *Maintenance and Service* für die deutsche Sprache an. Die blaue Leiste gibt den *Score* des Vorschlags an. Der Score gibt an, wie hoch die Wahrscheinlichkeit ist, dass das vorgeschlagene Synonym zu dem angezeigten Konzept passt. Über den links neben der Leiste angebrachten Button, lässt sich das Konzept in die Taxonomie übernehmen. Über den Button rechts kann ein Vorschlag abgelehnt werden. Wurde ein dieser abgelehnt, so wird dieser in Zukunft nicht mehr in der App angezeigt.

### 5.3.5. Visualisierung der Synonymvorschläge

In diesem Unterkapitel wird die View `Suggestions` vorgestellt, welche verfügbare Vorschläge für neue Synonyme an einem Ort auflistet. Diese View kann ausschließlich über den zweiten Button von rechts in der Navigationsleiste erreicht werden. Hierbei ist es möglich Vorschläge einer bestimmten Sprache oder aller Sprachen anzuzeigen. Wird eine der beiden Optionen gewählt, so wird man in die in Abbildung 5.18 zu sehende View `Suggestions` weitergeleitet.

Diese ist in verschiedene Bereiche aufgeteilt, wobei jeder Bereich ein Konzept repräsentiert. Am oberen Ende eines jeden Bereichs befindet sich ein sogenannter *Breadcrumb*. Dieser visualisiert den Pfad des gezeigten Konzepts zum Wurzelkonzept. Besitzt ein Konzept mehr als ein Elternkonzept, so werden hier der Anzahl der Elternkonzepte entsprechend viele Breadcrumbs angezeigt. Darunter befinden sich die nach Sprachen gruppierten Vorschläge. Existieren bereits Synonyme in der Sprache, in der die Vorschläge vorliegen, so werden diese als Referenz und zum einfacheren Einordnen eines Vorschlags unter der Überschrift *Current Synonyms* aufgelistet.

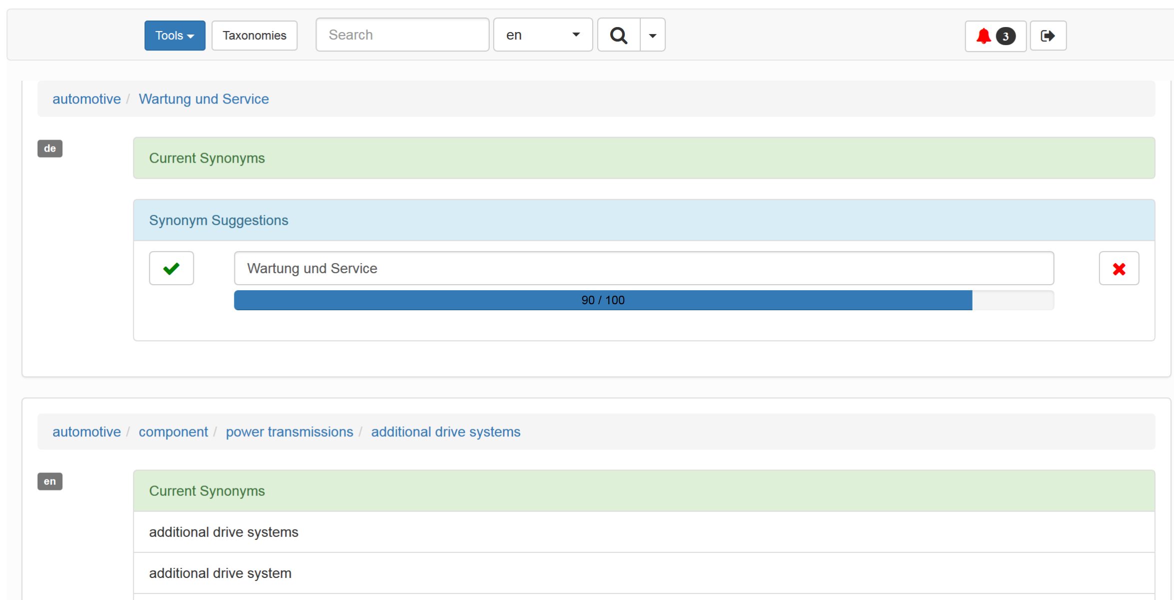



**Abbildung 5.18.:** Screenshot der View `Suggestions`, die unbearbeitete Vorschläge anzeigt

Unter der Auflistung der bereits existierenden Synonyme, werden die Vorschläge aufgelistet. Der Name eines Vorschlags wird in einer Textbox angezeigt, sodass dieser bei Bedarf angepasst werden kann. Unter der Textbox befindet sich eine Anzeige, die die Wahrscheinlichkeit darstellt, mit welcher ein Vorschlag zu dem Konzept passend ist. Der Wert kann zwischen 0 und 100 liegen, wobei höhere Werte eine bessere Konformität aufzeigen. Soll ein Vorschlag übernommen werden, so ist der Button links mit dem grünen Haken-Symbol zu verwenden. Passt der Vorschlag nicht zu dem Konzept, so kann dieser mit dem rechts befindlichen Button mit roten Kreuz-Symbol abgelehnt werden. Die beiden Buttons sind bewusst weit voneinander entfernt, sodass eine durch eine ungenaue Toucheingabe zustande kommende Fehlbedienung minimiert werden kann. In beiden Fällen wird dem Server das Ergebnis mitgeteilt und der Vorschlag aus der Ansicht entfernt.

### 5.3.6. Suchfunktion innerhalb einer Taxonomie

Das Finden eines bestimmten Konzepts, beziehungsweise Synonyms kann durch das manuelle Erkunden des Taxonomiebaums bewerkstelligt werden. Diese Vorgehensweise erfordert entweder eine sehr gute Kenntnis der Taxonomie oder benötigt viel Zeit, da viele Konzepte durchsucht werden müssen. Aus diesem Grund wurde eine Suchfunktion integriert, die über die Navigationsleiste zu erreichen ist. Ein Screenshot der View `Search`, die für das Anzeigen der Suchergebnisse zuständig ist, kann bei Abbildung 5.19 gefunden werden.

Der eingetragene Suchbegriff im Textfeld der Navigationsleiste wird dazu verwendet nach Synonymen zu suchen, die diesen im Namen enthalten. Der Begriff kann sich somit an beliebiger Stelle innerhalb eines Suchergebnisses befinden. In bestimmten Anwendungsfällen ist ausschließlich die ID eines Konzepts bekannt. Um nach einer ID zu suchen gibt es zwei verschiedene Möglichkeiten. Falls der Suchbegriff aus einer Zahl besteht, werden zusätzlich zu den Namen der Synonymen die IDs der

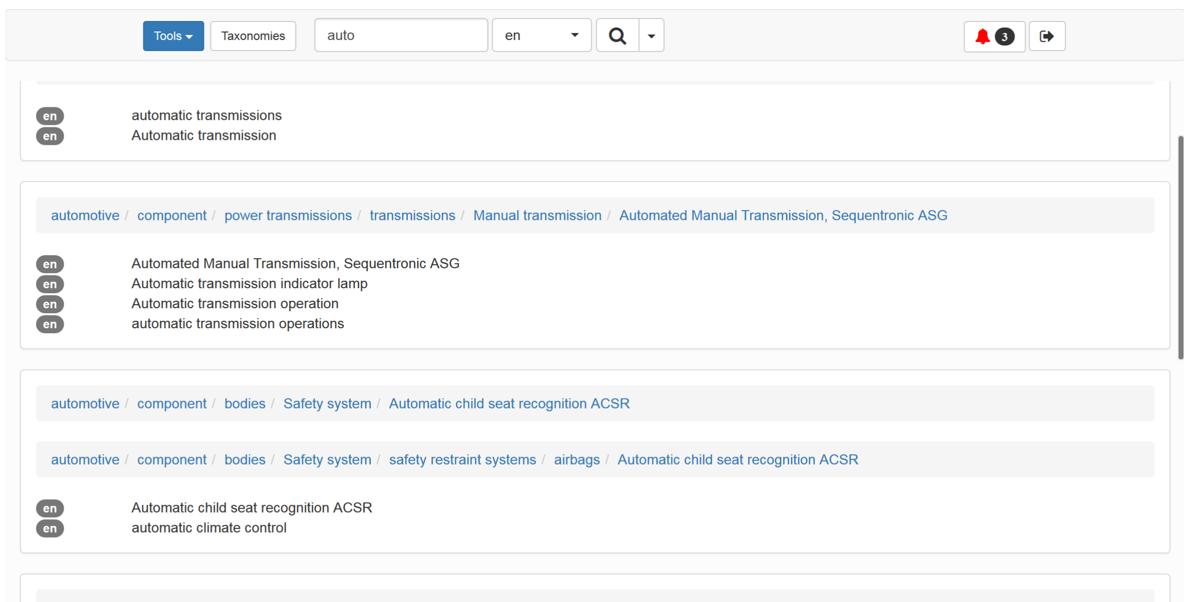



**Abbildung 5.19.:** Screenshot der View `Search`, die Suchergebnisse darstellt

Konzepte durchsucht. Sollen ausschließlich die IDs von Konzepten durchsucht werden, können die Suchmuster `@id:[id]`, `@id=[id]` oder `@id [id]` verwendet werden. `[id]` ist hierbei der Platzhalter für eine ID und muss durch diese ersetzt werden. Besteht der Namen eines Synonyms aus einer Zahl und es sollen keine Konzept-IDs bei der Suche miteinbezogen werden, können die Suchmuster `@name:[name]`, `@name=[name]` und `@name [name]` verwendet werden. `[name]` ist der Platzhalter für den Namen und muss durch diesen ersetzt werden.

Durch die rechts neben dem Textfeld befindliche Combobox können die in die Suche einzubeziehenden Sprachen spezifiziert werden. Zu Beginn ist immer die Standardsprache einer Taxonomie vorausgewählt. Es ist mögliche mehrere oder keine Sprache auszuwählen. Wird keine spezifische Sprache bestimmt, so wird eine sprachunabhängige Suche durchgeführt. Über den Button mit dem Lupen-Icon lässt sich der Suchvorgang beginnen. Der Benutzer wird zeitgleich in die View `Search` weitergeleitet.

In der View `Search` werden Suchergebnisse nach Konzepten gruppiert und in optisch abgetrennten Bereichen präsentiert. Über einen am oberen Ende eines Bereiches befindlichen Breadcrumb wird der Pfad eines Konzepts zur Wurzel angezeigt. Besitzt ein Konzept mehr als ein Elternkonzept, so werden an dieser Stelle eine der Anzahl der Elternkonzepte entsprechende Menge an Breadcrumbs angezeigt. Durch einen Klick auf einen Teil des Breadcrumb wird der Benutzer zurück in die View `View` geleitet, es wird der Pfad zu dem angeklickten Konzept geöffnet und dieses wird in der Detailansicht angezeigt.

Unter dem Breadcrumb befinden sich die gefundenen Synonyme. Links befindet sich die Sprache des Synonyms, rechts dessen Namen. Durch einen Klick auf eines der Synonyme, wird der Benutzer in die View `View` geleitet und es wird der Pfad zu dem Konzept, welches die Synonyme enthält, geöffnet. Zusätzlich wird dieses Konzept in der Detailansicht angezeigt.

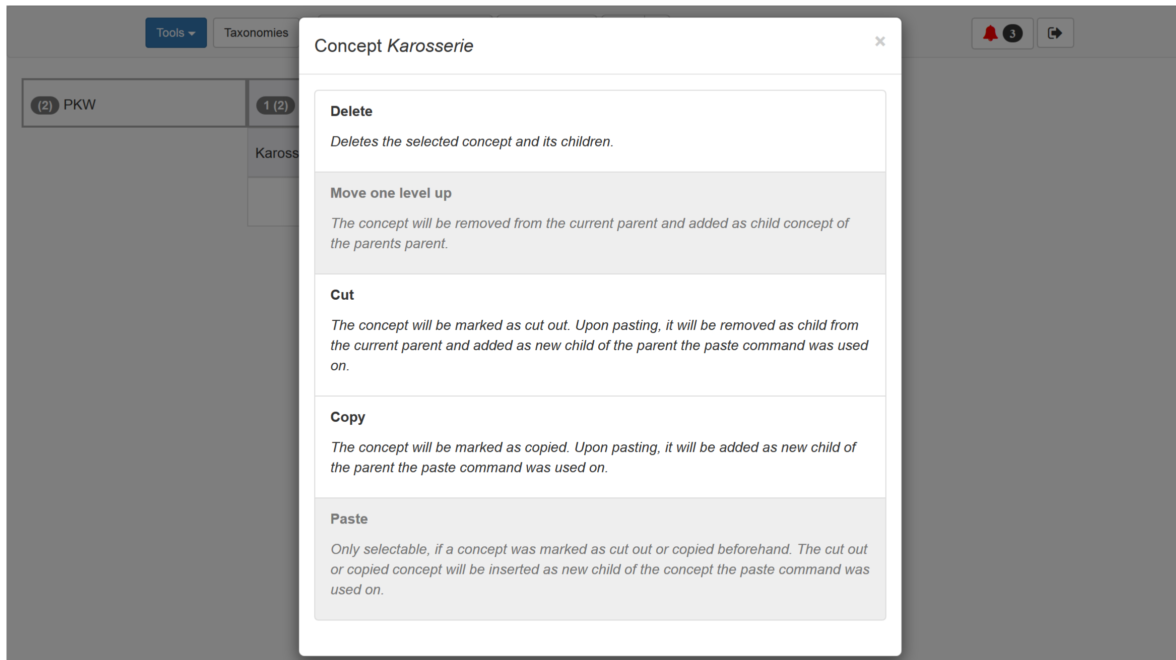



**Abbildung 5.20.:** Ausschnitt eines Screenshot der App, der das Kontextmenü zeigt. Dieses kann zum Kopieren/Verschieben von Konzepten verwendet werden

Für bestimmte Suchbegriffe können sehr viele Suchergebnisse gefunden werden. Um die Suche zu beschleunigen, den Datenverkehr gering zu halten und die Ressourcen des Clients zu schonen unterstützt die Suche die sogenannte *Pagination*. Hierbei werden nur Teile des Ergebnisses abgefragt und dem Nutzer wird es ermöglicht bei Bedarf weitere vom Server anzufordern. Um einen guten Kompromiss zwischen der Menge der Suchergebnisse und der benötigten Zeit für eine Suche zu erhalten, können maximal 100 Synonyme gleichzeitig angezeigt werden.

### 5.3.7. Kopieren und verschieben von Konzepten

Um die Wartung einer Taxonomie zu vereinfachen, wird eine Funktion zum Kopieren und Verschieben von Konzepten implementiert. Die App ist primär auf Touch-fähige Tablets ausgelegt, weswegen die Verwendung von Touch-Gesten möglich ist. Um ein Konzept kopieren oder verschieben zu können, muss der Benutzer einen sogenannten *langen Druck (long press)* im Taxonomiebaum auf ein Konzept ausführen. Dies bedeutet, dass er dieses für mindestens 400 ms berühren, beziehungsweise anklicken muss. Nach Ablauf der Zeit öffnet sich ein Kontextmenü, über welches verschiedene Aktionen ausgeführt werden können.

#### Funktionsweise

Ein Screenshot des Menüs ist in Abbildung 5.20 zu sehen. Folgend werden die einzelnen auswählbaren Aktionen anhand ihrer englischen Begriffe näher erläutert:

- **Delete.** Löscht das aktuelle Konzept und dessen Kinder. Alle diesen Konzepten zugeordneten Synsets, Synonyme, Tokens und Attribute werden gelöscht.
- **Move one level up.** Verschiebt ein Konzept und dessen Kindkonzepte auf die nächst höhere Ebene innerhalb des Taxonomiebaums. Diese Aktion kann nicht bei einem direkten Kind des Wurzelkonzepts ausgeführt werden, da dies zu zwei Wurzelkonzepten führen würde. Eine Taxonomie kann immer nur ein Wurzelkonzept enthalten.
- **Cut.** Umrandet ein Konzept mit einer gestrichelten Linie, wodurch es als ausgeschnitten markiert ist. Zu diesem Zeitpunkt wurden noch keine Änderungen an dem Konzept durchgeführt. Durch erneutes Aufrufen des Kontextmenüs bei einem anderen Konzept kann die *Paste* Aktion ausgewählt werden. Hierdurch wird das aktuell ausgewählte Konzept, das neue Elternkonzept des zuvor ausgeschnittenen Konzepts. Beim Verschieben werden Kindkonzepte mit verschoben.

  Ein Verschieben eines Konzepts zu einem Kindkonzept des zuvor ausgeschnittenen ist nicht möglich und wird mit einer Fehlermeldung quittiert. Durch diese Restriktion werden Kreise innerhalb des Graphen vermieden. Würde ein Kreis entstehen, wären die Bedingungen einer Hierarchie verletzt und somit die Voraussetzung einer validen Taxonomie nicht mehr gegeben.
- **Copy.** Markiert ein Konzept als kopiert. Diese Aktion weist ein ähnliches Verhalten wie das Ausschneiden (Cut) auf. Der einzige Unterschied besteht darin, dass anstatt dem Austauschen des Verweises auf ein Elternkonzept, ein zusätzlicher hinzugefügt wird. Hat ein Konzept zuvor ein Elternkonzept gehabt, so besitzt dieses nun zwei. Hierdurch erscheint es an zwei verschiedenen Stellen innerhalb des Taxonomiebaums. Da keine Kopie des Konzepts erzeugt, sondern ausschließlich eine Referenz erstellt wird, wirken sich Änderungen an einem Konzept mit mehreren Elternkonzepten auf mehrere Stellen innerhalb des Taxonomiebaums aus.
- **Paste.** Diese Aktion ist nur verfügbar, wenn zuvor ein Konzept ausgeschnitten (cut) oder kopiert (copy) wurde. Ist diese verfügbar, so wird ein Konzept als Kind des ausgewählten Konzepts eingefügt.

### Limitierungen

Ein zum Kopieren oder Verschieben markiertes Konzept kann nicht an beliebiger Stelle innerhalb des Taxonomiebaums eingefügt werden, um dessen Integrität zu wahren.

In Kapitel 2.1 wurde beschrieben, dass eine Taxonomie stets eine hierarchische Struktur ist. Diese hat als Voraussetzung, dass keine Kreise innerhalb des Graphen existieren dürfen. Aus diesem Grund ist es nicht erlaubt ein Konzept unterhalb sich selbst einzufügen, da dies zu einem Kreis führen würde.

Des Weiteren kann ein als kopiert/ausgeschnitten markiertes Konzept die Einfügeoperation nicht auf sich selbst anwenden. Dies würde erneut zu einem Kreis innerhalb des Graphen führen.

Bei jeder Kopier- und Verschiebeoperation wird serverseitig geprüft, ob diese einen Kreis innerhalb des Graphen erzeugen würde. Ist dies der Fall, wird dem Client eine Fehlermeldung zurückgeliefert.

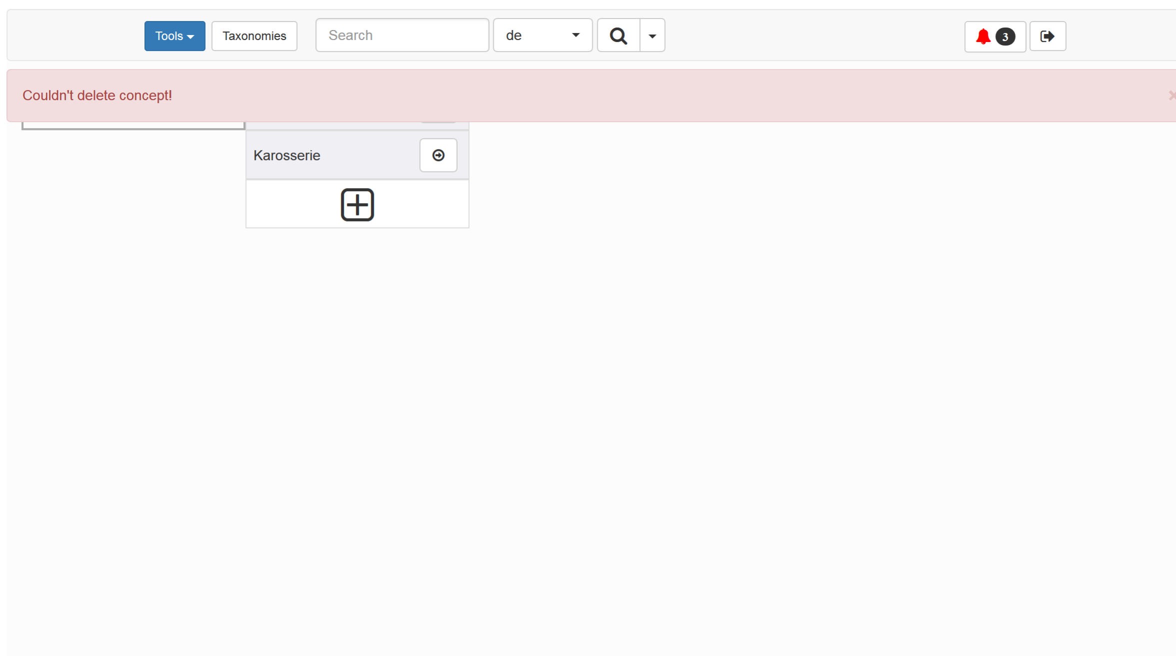



**Abbildung 5.21.:** Screenshot eines Alerts, der besagt, dass ein Konzept nicht gelöscht werden konnte. Die Gründe hierfür werden nicht genannt.

### 5.3.8. Behandlung von Fehlern

Während der Ausführung einer Software kann es jederzeit zu unerwarteten Fehlern kommen. Hierbei können sowohl interne, als auch externe Fehler auftreten. Interne Fehler charakterisieren sich dadurch, dass diese durch selbst entwickelten Programmcode ausgelöst werden. Externe Fehler hingegen treten außerhalb des Systems auf und können nicht von einem Entwickler der Applikation behoben werden. Hierzu zählen zum Beispiel Netzwerkfehler.

Egal durch welchen Grund ein Fehler auftritt, eine Applikation sollte entsprechend darauf reagieren. Bei Fehlern die nicht automatisch behoben werden können sollte der Benutzer durch eine entsprechende Fehlermeldung informiert werden. Der Inhalt und Detailgrad einer Fehlermeldung sollte sich nach der zu erwartenden Benutzergruppe richten.

Schlägt beispielsweise eine Datenbankabfrage fehl, so ist es bei einem Administrator sinnvoll diesem den genauen Grund des Fehlschlagens mitzuteilen. Hierdurch erhält dieser eine Hilfestellung bei der Lösung des Problems. Tritt derselbe Fehler allerdings bei einem normalen Benutzer auf, so genügt es auf eine Fehlfunktion des Systems hinzuweisen. Eine genaue Fehlermeldung hat unter Umständen folgende Nachteile:

- Fehlermeldungen, wie sie zum Beispiel von Datenbanken erzeugt werden, enthalten viele technische Begriffe. Ein normaler Benutzer kann diese im Allgemeinen nicht interpretieren.

- Durch genaue Fehlermeldungen können Schwachstellen und interne Vorgehensweisen des Systems an einen Benutzer offenbart werden. Einem potentiellen Angreifer werden hierdurch mögliche Angriffsvektoren geliefert.

In der aktuellen Implementierung wird nicht zwischen verschiedenen Benutzergruppen unterschieden, weshalb ausschließlich unspezifische Fehlermeldungen an den Benutzer weiter gegeben werden. Tritt ein unerwarteter Fehler auf, so wird ein sogenannter *Alert* angezeigt. Dieser wird als eine alles überlagernde Leiste dargestellt, die eine Meldung enthält und über einen Button geschlossen werden kann. In Abbildung 5.21 kann ein Alert gesehen werden, der Angibt, dass ein Konzept nicht gelöscht werden kann. Den genauen Grund hierfür erfährt der Benutzer nicht. Ein Administrator hingegen kann in der Log-Datei des Servers eine ausführliche Fehlermeldung begutachten.

Da die App wenig Programmlogik enthält, treten die meisten Fehler während der Kommunikation mit dem Server auf. Wie in Kapitel 5.3.1 erwähnt wickelt das AngularJS Plugin *Restangular* die REST Kommunikation mit dem Server ab. Zur Fehlerbehandlung setzt dieses auf sogenannte Versprechen (Promisses). Einem Versprechen können mit der Funktion `then(successHandler, errorHandler)` zwei verschiedene Funktionen übergeben werden. Die erste Funktion wird aufgerufen, wenn die Anfrage erfolgreich war. Als Parameter wird hier die Antwort des Servers geliefert. Die zweite wird verwendet, wenn während der Anfrage ein Fehler aufgetreten ist. An dieser Stelle wird die Fehlermeldung des Server als Parameter übergeben.

In dieser Arbeit befindet sich der Code zum Anzeigen der Fehler-Alerts grundsätzlich in der zweiten Funktion. Alerts dienen nicht nur dazu auf Fehler hinzuweisen. Sie werden auch verwendet um den Benutzer über erfolgreiche Aktionen zu informieren. Diese als Erfolg-Alerts bezeichneten Meldungen werden in der ersten Funktion eines Promiss-Handlers erzeugt. Listing 5.1 zeigt in Pseudo-Code, wie Promisses gehandhabt und verschiedene Alerts erzeugt werden

```
1 Restangular.RestAbfrage().then(function(antwort) {
2       // Wird aufgerufen, wenn die REST Abfrage erfolgreich war
3       // Zeige einen Erfolg-Alert
4 }, function(fehler) {
5       // Wird aufgerufen, wenn während der REST Abfrage ein Fehler aufgetreten ist
6       // Zeige einen Fehler-Alert
7 });
```

**Listing 5.1:** Verwendung von Promisses zur Fehlerbehandlung

# 6. Evaluation

In diesem Kapitel findet zuerst ein Vergleich zwischen dem alten Editor und der neu entwickelten App satt. Anschließend wird eine Performance-Evaluation des Systems durchgeführt. Hierbei wird getestet, wie sich das System bei einer großen Menge von Taxonomien und bei einer in der Größe ansteigenden Taxonomie verhält.

## 6.1. Vergleich zwischen neuem und altem Taxonomieeditor

In diesem Kapitel wird der alte und in Kapitel 4.3 vorgestellte Taxonomieeditor mit dem neu entwickelten App verglichen. Hierbei wird insbesondere auf wichtige Funktionen, die Visualisierung der Taxonomie, einen behobenen Fehler und die Benutzererfahrung eingegangen.

### 6.1.1. Funktionsvergleich

Der Vergleich der Funktionen zwischen dem alten Taxonomieeditor und der App ist in drei verschiedene Teile aufgeteilt. Zuerst werden die Funktionen vorgestellt, die von beiden Programmen unterstützt werden. Anschließend werden die Funktionen hervorgehoben, die ausschließlich in der App zu Verfügung stehen. Zum Schluss wird darauf eingegangen, welche Funktionen nicht in der App realisiert wurden und nur im alten Taxonomieeditor implementiert sind.

#### Funktionen die beide Applikationen bieten

- Ein wichtiger Punkt beider Applikationen ist die Visualisierung des Taxonomiebaums. Ein ausführlicher Vergleich findet im späteren Verlauf dieses Unterkapitels statt
- Die Details eines Konzepts können vom Benutzer betrachtet und editiert werden
- Eine Suchfunktion ermöglicht es nach Teilen eines Namens eines Synonyms zu suchen

### Funktionen die ausschließlich in der App verfügbar sind

- Die Zielplattform der App sind Tablets. Diese besitzen im Allgemeinen weniger nutzbare Bildschirmfläche, als dies ein Desktop-Computer bieten kann. Um die nutzbare Fläche zu maximieren, kann die App im Vollbildmodus verwendet werden
- Eines der Hauptziele dieser Arbeit ist es, gegebene Vorschläge zu bewerten und diese entweder in die Taxonomie zu übernehmen oder abzulehnen. Dies kann in der App entweder in der Detailansicht eines Konzepts oder in einer eigens dafür erstellten View durchgeführt werden
- In manchen Fällen möchte ein Benutzer wissen welches Konzept sich hinter einer bestimmten ID verbirgt. Hierzu wurde die Suche so erweitert, dass ein Konzept über dessen ID ermittelt werden kann
- Es ist möglich, dass verschiedene Benutzer zeitgleich mit mehreren Instanzen der App auf eine Taxonomie zugreifen. Jeder dieser Benutzer greift auf die gleichen Daten zu, wodurch diese für alle zeitgleich verfügbar sind
- Im Gegensatz zum alten Editor findet eine Kommunikation zwischen einem Client und einem Server statt. Im Allgemeinen werden diese nicht auf der gleichen Maschine ausgeführt, weshalb eine Netzwerkkommunikation stattfindet. Um das Mitschneiden des Netzwerkverkehrs und unautorisierten Zugriff zu vermeiden, wird jegliche Netzwerkkommunikation verschlüsselt und ein Verwenden des Systems ist nur nach erfolgreicher Authentifizierung möglich
- Die App ist ausschließlich mit Webtechniken implementiert, wodurch diese in aktuellen Browsern lauffähig ist. Aus diesem Grund ist diese auf nahezu jeder Plattform einsetzbar. Für eine optimale Benutzererfahrung sollte diese allerdings mit einem touchfähigen Tablet verwendet werden
- Um Taxonomien die in dem in Kapitel 4.2 vorgestellte Format vorliegen innerhalb des neu entwickelten Systems nutzen zu können, wurde eine Import-Funktion implementiert. Diese ermöglicht es, die im XML-basierte Format vorliegende Taxonomie an den Server zu senden und in die Datenbank zu importieren
- Eine Export-Funktion ermöglicht das exportieren der Taxonomie aus der Datenbank in das in Kapitel 4.2 vorgestellte XML-basierte Format. Dies ermöglicht es, auf dem alten Format basierende Werkzeuge und Techniken, mit einer vom neu entwickelten System verwalteten Taxonomie zu verwenden

### Funktionen die ausschließlich im alten Taxonomieeditor verfügbar sind

- In den gegebenen Taxonomien sind Konzepte und Synonyme enthalten, die an keiner Stelle innerhalb des Taxonomiebaums eingeordnet sind. Der alte Taxonomieeditor bietet hier eine Möglichkeit diese Taxonomien in einer gesonderten Liste anzuzeigen. Diese Funktion wurde aufgrund der knappen zur Verfügung stehenden Zeit nicht in der App implementiert

- Beim manuellen Editieren, sprich ohne die Hilfe eines Editors, kann es vorkommen, dass IDs doppelt vergeben werden. Um die hierdurch entstehenden Problem zu vermeiden, kann eine Taxonomie repariert werden. In diesem Fall bedeutet dies, dass kollidierende IDs neu vergeben werden. Innerhalb des in dieser Arbeit entwickelten Systems kann diese Art von Problem nicht auftreten, da dies von der Datenbank verhindert wird. Aus diesem Grund wurde auf die Implementierung einer solchen Funktion verzichtet

### 6.1.2. Verarbeitbare Taxonomiegröße

Eine Taxonomie wird auf verschiedene Weise in der alten und neuen Applikation persistent gespeichert. In dem neu entwickelten System wird diese in eine relationale Datenbank geschrieben. Zu keinem Zeitpunkt wird die komplette Taxonomie in den Hauptspeicher des Server, beziehungsweise des Clients geladen. Vielmehr werden ausschließlich die momentan benötigten Teile aus der Datenbank abgefragt und verarbeitet. Die Größe einer Taxonomie wird somit durch den verfügbaren Speicher der Datenbank festgelegt.

Die alte Applikation ist für Desktop-Computer ausgelegt und die komplette Taxonomie muss zum Öffnen lokal vorliegen. Beim Laden einer Taxonomie wird diese deserialisiert und komplett in den Hauptspeicher geladen. Hierdurch ergibt sich eine Limitierung der Taxonomiegröße anhand des verfügbaren Hauptspeichers.

### 6.1.3. Visualisierung einer Taxonomie

Die Visualisierung einer Taxonomie wird innerhalb der alten und neuen Applikation auf verschiedene Weise bewerkstelligt. In der Alten wird der Taxonomiebaum auf der linken Seite in eingerückter Form dargestellt. Diese Art der Visualisierung ist durch deren Nutzung in dem Dateimanager *Windows Explorer*, der mit jedem Microsoft Betriebssystem ausgeliefert wird, weithin bekannt. Die App hingegen verwendet für jede Ebene eine eigene vertikale Liste, die von links nach rechts horizontal dargestellt wird.

Die alte Applikation lädt stets die komplette Taxonomie in den Hauptspeicher. Dies bietet den Vorteil, dass bei der Visualisierung der Taxonomie der komplette Baum ohne weiteres Laden von neuen Daten dargestellt werden kann. In der App liegen die Daten auf einem Server und es werden nur die benötigten angefordert und dargestellt. Hierdurch fällt die Anforderung an den Hauptspeicher geringer aus, allerdings benötigt das Erkunden des Taxonomiebaums mehr Zeit, da die jeweils benötigte Daten vom Server angefordert werden müssen.

Durch einen Klick auf ein Konzept im Taxonomiebaum wird bei beiden Applikationen dieses auf der rechten Seite in einer detaillierten Ansicht dargestellt. Der grundlegende Aufbau ist bei beiden Ähnlich. Alle vorhandene Synsets werden untereinander aufgelistet und enthalten die entsprechenden Synonyme. Es ist mögliche neue Synsets, Synome oder Tokens hinzufügen oder vorhandene zu editieren, beziehungsweise zu löschen. Beide bieten zudem die Möglichkeit Attribute hinzuzufügen, zu löschen oder zu editieren.

### 6.1.4. Behobener Fehler

Innerhalb der alten Applikation gibt es einen schwerwiegenden Fehler. Wie in Kapitel 4.2 beschrieben kann ein Konzept mehrere Elternkonzepte besitzen. Das erwartete Verhalten ist es, dass in diesem Fall ein Konzept unterhalb beider Elternkonzept im Taxonomiebaum angezeigt wird. Durch den Fehler wird dieses allerdings nur unter einem zufälligen Elternkonzept angezeigt. Dem Benutzer ist es somit nicht möglich zu erkennen, dass ein Konzept mehrere Elternkonzepte besitzt.

In der App wurde dieser Fehler behoben und ein Konzept mit mehreren Elternkonzepten wird unterhalb dieser angezeigt.

### 6.1.5. Benutzererfahrung

Um die Arbeit mit einer Taxonomie möglichst einfach zu gestalten, sollte eine dafür ausgelegte Applikation eine gute Benutzererfahrung bieten. In diesem Kontext bedeutet dies, dass verschieden Funktionen möglichst einfach zu erreichen und durchzuführen sind. Zudem sollte es eine klare Benutzerführung geben.

Der alte Taxonomieeditor verwendet für die Visualisierung des Taxonomiebaums stets den gleichen Platz. Die App hingegen verwendet nach Möglichkeit den kompletten verfügbaren Platz der Bildschirmfläche. Dies ermöglicht es den Taxonomiebaum übersichtlicher zu visualisieren.

Beide Applikationen bieten eine Detailansicht für Konzepte. Der alte Editor verwendet an dieser Stelle sehr viele beschriftete Buttons, die diesen überfüllt wirken lassen. In der App werden für Buttons hauptsächlich Icons verwendet, um eine bessere Übersicht zu erreichen. Zum Editieren und hinzufügen von Synsets, Synonymen, Tokens und Attributen werden in der alten Applikation jeweils eigene Fenster geöffnet. Dies führt dazu, dass der Bildschirm viele kleine Fenster beherbergen muss. Dies kann schnell unübersichtlich werden. Um dies zu vermeiden, wurde in der App weitestgehend auf Pop Ups verzichtet. Wenig verwendete Funktionen, wie zum Beispiel das Ändern des Namens eines Synonyms wurde in Bereiche ausgelagert, die sich dynamisch ein- und ausblenden lassen.

## 6.2. Performance-Evaluation

In diesem Unterkapitel findet eine Performance-Evaluation des erstellten Systems statt. Hierbei wird überprüft wie sich verschiedene Metriken zum einen bei sehr vielen und zum andern bei einer in der Größe anwachsenden Taxonomie verhält. Am Ende dieses Kapitels findet eine Bewertung der gefundenen Ergebnisse statt.

Ein Vergleich der Performance mit dem in Kapitel 4.3 vorgestellten Editor kann nicht stattfinden, da sich der Aufbau und die Funktionsweise der beiden Applikationen zu stark unterscheidet.

*Anmerkungen:*

1. In den folgenden Unterkapiteln wird des Öfteren der Begriff des *Durchschnitts* verwendet. Gemeint ist hiermit das Arithmetische Mittel.

2. In den untenstehenden Diagrammen sind jeweils zwei Linien eingezeichnet. Die graue nicht stetige Linie repräsentiert die Daten. Die schwarze stetige Linie ist eine auf den Daten basierende Trendlinie.

### 6.2.1. Verhalten bei einer großen Menge Taxonomien

Dieses Unterkapitel beschäftigt sich mit dem Verhalten des Systems bei vielen kleinen Taxonomien. Zuerst wird das Vorgehen dieses Teils der Evaluation erläutert. Anschließend werden die Ergebnisse präsentiert.

#### Vorgehen

Um das Verhalten des Systems bei sehr vielen kleinen Taxonomien testen zu können, wird eine Taxonomie mit folgenden Kennwerten verwendet:

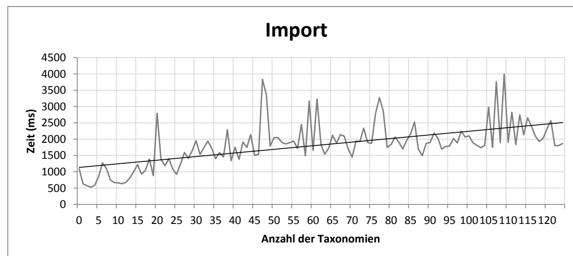
- 128 Konzepte
- 128 Synsets
- 220 Synonyme
- 286 Tokens
- 255 Attribute

Jedes Konzept besitzt genau ein Synset. Ein Synset enthält durchschnittlich 1,72 Synonyme. Ein Synonym besteht im Durchschnitt aus 1,3 Wörtern (Tokens). Bis auf das Wurzelkonzept, hat jedes Konzept zwei Attribute. Diese beschreiben die Relation zwischen Eltern- und Kindkonzepten.

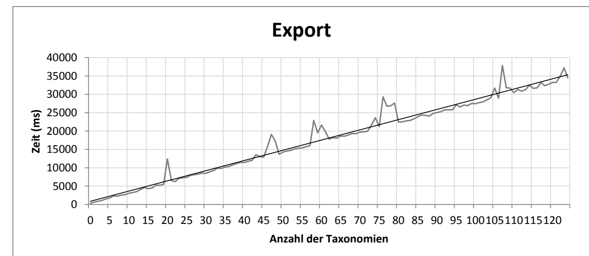
Die Daten der Taxonomie sind in einer Datei mit der Bezeichnung `klein_data.tax` gespeichert, die eine Dateigröße von 122 kB hat. Die Relationen sind in der Datei `klein_forschung.tax` mit einer Dateigröße von 76 kB zu finden.

Diese Taxonomie wird 125-mal importiert, womit am Ende 125 voneinander unabhängige Taxonomien im System vorhanden sind. Dieses enthält somit zum Schluss 16000 Konzepte, 16000 Synsets, 27500 Synonyme, 35750 Tokens und 31875 Attribute. Nach jedem einzelnen Importvorgang werden die folgenden Metriken betrachtet:

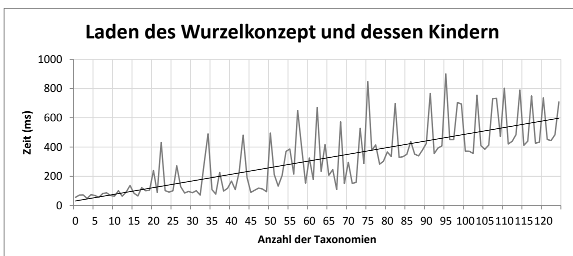
1. Die benötigte Zeit zum Anlegen und importieren der kompletten Taxonomie (Daten und Relationen)
2. Die benötigte Zeit zum Exportieren der letzten importierten Taxonomie
3. Die Zeit zum Laden vom Wurzelkonzepten und dessen Kindkonzepten
4. Die Zeit zum Laden der Details, sprich den Synsets, Synonymen, Tokens und Attribute, des Wurzelkonzepts



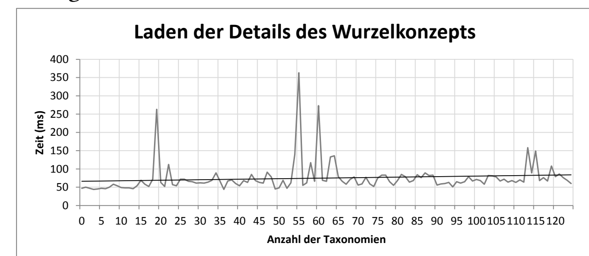

(a) Diagramm des Zeitverhaltens beim kontinuierlichen Importieren von Taxonomien. Mit zunehmender Anzahl von Taxonomien benötigt der Import mehr Zeit.




(b) Diagramm des Zeitverhaltens beim kontinuierlichen Exportieren von Taxonomien. Mit zunehmender Anzahl von Taxonomien benötigt der Export signifikant mehr Zeit.




(c) Diagramm des Zeitverhaltens beim kontinuierlichen Laden der Wurzelkonzepte und deren Kinder. Mit zunehmender Anzahl von Taxonomien benötigt dieser Vorgang signifikant mehr Zeit.




(d) Diagramm des Zeitverhaltens beim kontinuierlichen Laden der Details des Wurzelkonzepts. Mit zunehmender Anzahl von Taxonomien bleibt die benötigte Zeit konstant.

**Abbildung 6.1.:** Auswertung der Evaluation mit einer großen Menge Taxonomien

Im Optimalfall bleibt bei allen Kennwerten die für die Aktionen benötigt Zeit konstant. Erwartet wird hingegen eine langsame lineare Steigerung der Zeit, da die Datenbank mit wachsender Anzahl von Taxonomien eine immer größere Datenmenge verarbeiten muss.

### Ergebnisse

In diesem Unterkapitel werden die Ergebnisse der Evaluation mit einer großen Menge an Taxonomien vorgestellt. Eine Visualisierung der vorgestellten Ergebnisse ist in Abbildung 6.1 zu sehen. Eine allgemeine Bewertung dieser findet in Kapitel 6.2.3 statt.

Abbildung 6.1a zeigt, dass die benötigte Zeit zum Importieren der Taxonomien stark zwischen den einzelnen Durchläufen variiert. Minimal benötigt ein Importvorgang 527 ms und maximal 3999 ms. Durchschnittlich wurde eine neue Taxonomie in 1824 ms importiert. Die im Diagramm eingezeichnete Trendlinie zeigt einen langsamen Anstieg der benötigten Zeit mit einer zunehmender Anzahl von Taxonomien. Die ersten 10 Importvorgänge haben durchschnittlich 794 ms und die letzten 10 2148 ms benötigt. Importe finden im Allgemeinen selten statt, weswegen die ermittelten Zeiten einem Benutzer zugemutet werden können.

Die benötigte Zeit zum Exportieren einer Taxonomie steigt signifikant mit zunehmender Anzahl im System vorhandener Taxonomien an. Die in der Datenbank vorhandenen Taxonomie-Elemente

steigen durch das kontinuierliche Importieren derselben Taxonomie linear an. In Abbildung 6.1b ist durch die eingezeichnete Trendlinie ein mit wenigen Ausreißern stetiger linearer Anstieg der für den Export benötigten Zeit zu sehen. Minimal benötigt das Exportieren 326 ms und maximal 37853 ms. Somit wird für das Exportieren der 125. identischen Taxonomie rund 116-mal länger benötigt als für den ersten Export. Exportvorgänge, die unter einer Minute benötigen können dem Benutzer durchaus zugemutet werden. Dennoch sollte einer Prüfung von Möglichkeiten zum Verbessern des Verhaltens in späteren Arbeiten stattfinden.

Das Laden des Wurzelkonzepts und dessen Kinder weist stark variierende Werte auf. Dennoch ist in Abbildung 6.1c eine klarer linearer Anstieg der benötigten Zeit zu erkennen. Zu Beginn wird bei den ersten 10 Durchläufen durchschnittlich 68 ms benötigt. Die letzten 10 hingegen benötigen durchschnittlich 552 ms. Dies entspricht einer knapp 8-fachen Steigerung der benötigten Zeit. Minimal hat der Vorgang 49 ms, maximal 900 ms und durchschnittlich 314 ms in Anspruch genommen. Bei über 100 im System vorhandenen Taxonomien liegt die durchschnittlich benötigte Zeit bei $\approx$ 0,5 Sekunden, was einem Benutzer durchaus zugemutet werden kann.

Die letzte der überprüften Metriken stellt die Zeit zum Laden der Details eines Wurzelkonzepts dar. Die Größe dieser hat sich während der Evaluation nicht verändert. Die benötigte Zeit ist bis auf weniger Ausreißer konstant niedrig und wird von einem Benutzer kaum bemerkt. Minimal wurden 44 ms, maximal 363 ms und durchschnittlich 75 ms zum Laden der Details benötigt.

### 6.2.2. Verhalten bei einer in der Größe anwachsenden Taxonomie

Dieses Unterkapitel beschäftigt sich mit dem Verhalten des Systems bei einer sich stetig vergrößernden Taxonomie. Zu Beginn wird das Vorgehen dieses Teils der Evaluation erläutert und anschließend wird auf die Ergebnisse eingegangen.

#### Vorgehen

Im zweiten Teil dieser Evaluation soll überprüft werden, wie sich das System bei einer in der Größe ansteigenden Taxonomie verhält. Als Ausgangspunkt dient eine mit den folgenden Werten generierte Taxonomie:

- 500 Konzepte
- 1000 Synsets
- 1000 Synonyme
- 2000 Tokens

Das Wurzelkonzept enthält zu Beginn 50 Kindkonzepte. Die restlichen Konzepte werden zwischen den Kindeskindern des Wurzelkonzepts verteilt. Jedes Konzept besitzt zwei Synsets und jedes Synset ein Synonym. Jedes Synonym besteht aus zwei Tokens.

Um eine ansteigende Größe der Taxonomie zu erhalten, wird diese in mehreren Iterationen vergrößert. In jeder Iteration wird die Anzahl der Konzepte um 500, Synsets um 1000, Synonyme um 1000, Tokens

um 1000 und Attribute um 2000 erhöht. Durchgeführt werden 50 Iterationen, wodurch sich am Ende eine Taxonomie mit 25000 Konzepte, 50000 Snysets, 50000 Synonymen und 100000 Tokens ergibt. 2500 dieser Konzepte sind direkte Kinder des Wurzelkonzepts.

Pro Iteration werden dem Wurzelkonzept 50 neue Kindkonzepte hinzugefügt. Die restlichen werden den unterhalb der Kinder des Wurzelkonzepts liegenden Konzepten zufällig als Kindkonzepte zugewiesen. Synsets, Synonyme, Tokens und Attribute werden wie zuvor beschrieben verteilt.

In jeder Iteration werden die folgenden Metriken betrachtet:

1. Die benötigte Zeit zum Vergrößern der Taxonomie
2. Die benötigte Zeit zum Exportieren der Taxonomie
3. Die Zeit zum Laden des Wurzelkonzepts und dessen Kindkonzepten
4. Die Zeit zum Laden der Details, sprich den Synsets, Synonymen und Tokens, des Wurzelkonzepts

Im optimalen Fall bleiben 1. und 4. zeitlich konstant, da hier jeweils die gleichen Aktionen mit einer konstanten Datenmenge durchgeführt werden. Bei 2. und 3. wird ein linearer Anstieg der Zeit erwartet, da hier die entsprechende Datenmenge konstant zunimmt.

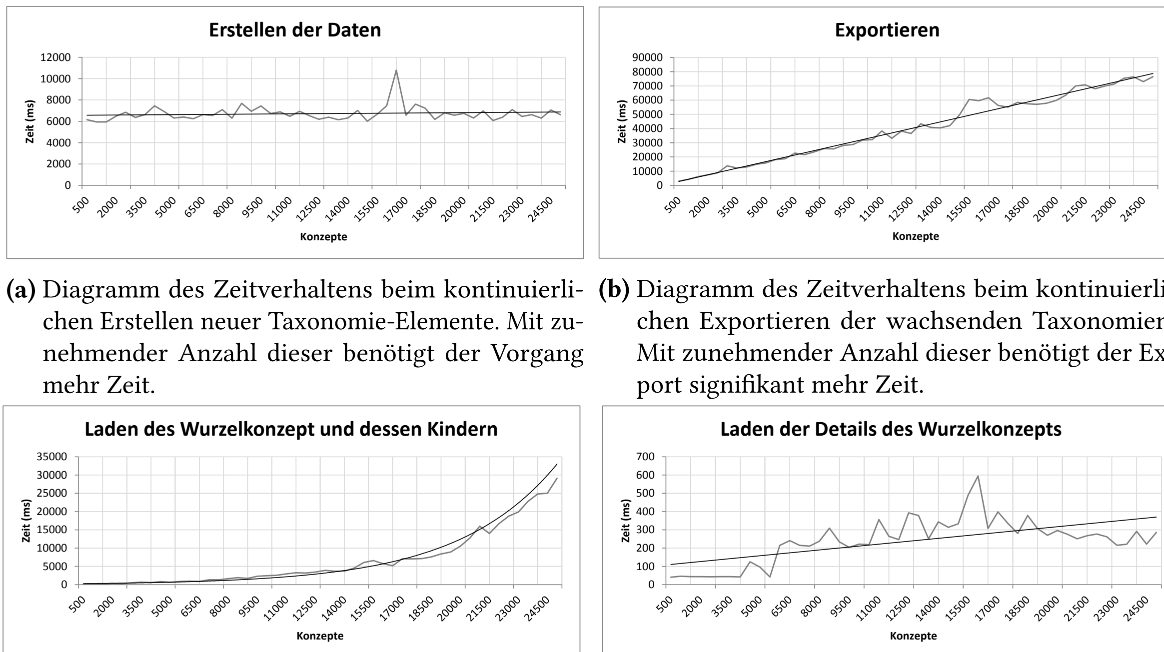
#### Ergebnisse

Folgend werden die Ergebnisse der mit dem zuvor beschriebenen Vorgehen durchgeführten Evaluation betrachtet. Eine allgemeine Bewertung der Ergebnisse findet in Kapitel 6.2.3 statt.

Die Zeit zum Erstellen der Daten ist wie in Abbildung 6.2a zu sehen nahezu durchgehend konstant. Minimal werden 5935 ms, maximal 10794 ms und durchschnittlich 6729 ms benötigt. Die Erstellung eines Konzepts und dessen Details benötigt somit im Durchschnitt 13,5 ms. Die benötigte Zeit steigt im Laufe der Durchführung der Evaluation nicht an und kann somit als konstant angesehen werden. Benutzer erstellen im Normalfall entweder einzelne Konzepte manuell oder importieren eine große Menge über die Importfunktion. Dieses Ergebnis deckt das manuelle Erstellen ab. Durch die geringe pro Konzept benötigte Zeit, geschieht dies für den Benutzer ohne Wartezeit.

Das in Abbildung 6.2b zu sehende Exportverhalten entspricht dem in Abbildung 6.2b zu sehenden Verhalten. Im Gegensatz zu den in Kapitel 6.2.1 verwendeten kleinen Taxonomien, wird hier eine Große verwendet. Minimal werden 3018 ms und maximal 7649 ms benötigt. Wie in Kapitel 6.2.1 beschrieben, können dem Benutzer Exportvorgänge unter einer Minute zugemutet werden. Dennoch sollte in späteren Arbeiten eine Prüfung der Möglichkeiten zur Verbesserung des Zeitverhaltens stattfinden.

Die Zeit zum Laden des Wurzelknotens und dessen Kindern steigt wie in Abbildung 6.2c exponentiell an. Um dies zu verdeutlichen wurde im Gegensatz zu den anderen Diagrammen eine exponentielle Trendlinie verwendet. Sowohl die zu ladende Anzahl an Konzepten (50 bis 2500), als auch die Gesamtzahl der Konzepte (500 bis 25000) steigt mit jeder Iteration linear an. Der Vorgang benötigt bei den ersten 10 Iterationen durchschnittlich 464 ms. Die letzten 10 Durchgänge hingegen benötigen




**(a)** Diagramm des Zeitverhaltens beim kontinuierlichen Erstellen neuer Taxonomie-Elemente. Mit zunehmender Anzahl dieser benötigt der Vorgang mehr Zeit.

**(b)** Diagramm des Zeitverhaltens beim kontinuierlichen Exportieren der wachsenden Taxonomien. Mit zunehmender Anzahl dieser benötigt der Export signifikant mehr Zeit.

**(c)** Diagramm des Zeitverhaltens beim kontinuierlichen Laden der Wurzelkonzepte und dessen Kinder. Mit zunehmender Anzahl der zu ladenden Konzepten benötigt dieser Vorgang signifikant mehr Zeit.

**(d)** Diagramm des Zeitverhaltens beim kontinuierlichen Laden der Details des Wurzelkonzepts. Mit zunehmender Anzahl von Taxonomie-Elementen bleibt die benötigte Zeit konstant.

**Abbildung 6.2.:** Auswertung der Evaluation mit einer sich stetig vergrößernden Taxonomie

19130 ms, was annähernd der 41-fachen Zeit entspricht. Durch den exponentiellen Anstieg der Benötigten Zeit, benötigt das System bei 25000 Konzepten annähernd 30 Sekunden, was keinem Benutzer zugemutet werden kann.

Die benötigte Zeit zum Laden der Details variiert wie in Abbildung 6.2d stark zwischen den einzelnen Iterationen. Über die gesamte Zeit hinweg gesehen, steigt diese leicht linear an. Minimal werden 41 ms, maximal 595 ms und durchschnittlich 240 ms zum Laden der Details des Wurzelkonzepts benötigt. Die benötigte Zeit liegt, bis auf einen Ausreißer, immer unter 500 ms. Diese Wartezeit kann jedem Benutzer zugemutet werden.

### 6.2.3. Bewertung der Ergebnisse

Aufgrund der für die Durchführung dieser Arbeit knapp bemessenen Zeit ist keine genau Bewertung der Ergebnisse möglich. Folgend werden nicht bestätigte Vermutungen zu den einzelnen Ergebnissen angestellt, die als Ausgangspunkt für weitere Arbeiten dienen können. Eine genauere Betrachtung dieser ist insofern interessant, da die zuvor angestellten Vermutungen über das Verhalten der einzelnen Metriken selten eingehalten werden konnten.

Viele der in Abbildung 6.1 und Abbildung 6.2 zu sehenden Diagramme weisen große Schwankungen zwischen den einzelnen Iterationen auf. Dies könnte am Testgerät und dessen Festplatte liegen. Es

liegt die Vermutung nahe, dass geringe Zeiten erreicht werden, wenn der Cache der Datenbank, beziehungsweise der Festplatte genügend Platz aufweist. Ist dieser voll und muss geleert werden, so ergeben sich zwangsweise Warte- und somit höhere Antwortzeiten.

Die zum Exportieren benötigte Zeit scheint mit der Gesamtanzahl von im System enthaltenen Konzepten zusammenzuhängen. Diese Vermutung wird angestellt, da das Exportieren der in Kapitel 6.2.1 verwendeten kleinen Taxonomie genau wie das Exportieren der in Kapitel 6.2.2 verwendeten großen Taxonomie stetig mehr Zeit benötigt. Um die für den Export benötigten Details eines Konzepts zu erhalten, werden viele große Tabellen der Datenbank miteinander gejoint. Hierdurch könnte sich die unabhängig von der Größe der zu exportierenden Taxonomie, benötigte Zeit erklären lassen. Durch verbesserte Datenbankabfragen oder ein anderes Datenbanklayout könnte dieses Verhalten eventuell verbessert werden.

Beim Laden des Wurzelkonzepts und dessen Kindern muss für jedes angezeigte Konzept ein Anzeigenamen ermittelt werden. Hierzu werden wie beim Exportieren viele große Tabellen miteinander gejoint. Bleibt die Menge der zu ladenden Konzepte konstant (Abbildung 6.1c) und die Gesamtdatenmenge steigt konstant an, so ergibt sich ein linearer Anstieg der benötigten Zeit. Steigt hingegen die Menge der zu ladenden Konzepte (Abbildung 6.2c) und die Gesamtdatenmenge konstant an, so ergibt sich insgesamt ein exponentieller Anstieg der benötigten Zeit.

In nahezu jeder der vorgestellten Auswertungen steigt die benötigte Zeit einer Aktion mit der Anzahl der im System enthaltenen Daten an. Durch den Einsatz anderer Datenbanktechnologien, wie beispielsweise Graphen-Datenbanken, könnte das Verhalten unter Umständen signifikant verbessert werden.

# 7. Zusammenfassung

Das Ziel dieser Arbeit war es eine App für ein Tablet zu präsentieren, die es ermöglicht eine Taxonomie anzuzeigen, von mehreren Benutzern gleichzeitig nutzbar und durch gegebene Vorschläge erweiterbar zu machen. Um dieses Ziel zu erreichen wurde ein komplettes Client-Server-System entwickelt, das über einen REST basierten Webservice kommuniziert.

Der Serverteil wurde in der Programmiersprache Java entwickelt, wobei hier als Grundtechnologie die Java Enterprise Edition (JEE) gewählt wurde. Der geschriebene Code wird in dem Application Server GlassFish ausgeführt. Eine persistente Speicherung der Daten wird durch die relationale Datenbank MySQL gewährleistet. Intern ist der Serverteil logisch in vier verschiedene Teile aufgeteilt. Hierzu zählen das Benutzer- und Rechtemanagement, die Taxonomieverwaltung, Grundlagen der Datenbankkommunikation und grundlegende Funktionen für die Implementierung der REST Schnittstelle. Der Hauptfokus lag hierbei auf der Entwicklung der Taxonomieverwaltung, weshalb ein Großteil des Codes für die Benutzer- und Rechteverwaltung aus einem anderen Arbeit übernommen wurde [SFB+15]. Ein Zugriff von außen ist nur über die entwickelte REST Schnittstelle möglich.

Genutzt wird diese Schnittstelle von der entwickelten Web-App, die als Client des Systems dient. Die App verwendet als Grundlage das JavaScript-Framework AngularJS und das CSS-Framework Bootstrap. Diese Technologien wurden gewählt, da diese eine schnelle Entwicklung ermöglichen und viele Standardprobleme, wie zum Beispiel das Binden von Daten an eine View, lösen. Die entwickelte App ermöglicht es eine beliebige Anzahl an Taxonomien aus dem zuvor genutzten XML-Format zu importieren. Durch den Importvorgang übernommene Taxonomien können innerhalb der App angezeigt, erkundet und editiert werden. Manche Anwendungsfälle und Vorgänge sind auf das alte Taxonomieformat angewiesen und können das neue nicht verarbeiten. Aus diesem Grund wurde eine Möglichkeit entwickelt, eine in das System importierte Taxonomie zurück in das alte Format zu wandeln.

Außerhalb des Systems werden neue Vorschläge für Synonyme generiert, die eine halbautomatische Erweiterung der Taxonomie ermöglichen [Est15]. Um einen Vorschlag endgültig in eine Taxonomie zu übernehmen, soll dieser von einem Experten begutachtet werden. In der App wurden Funktionen implementiert, die es dem Experten ermöglichen die Vorschläge zu sichten und passende zu übernehmen.

Eines der Hauptprobleme dieser Arbeit war es eine passende Visualisierung von Taxonomien für Tablets zu finden. Um die Beste für das gegebene Problem zu finden, wurden insgesamt sechs verschiedene Techniken miteinander verglichen. Als am besten geeignet wurde die sogenannte Side-by-Side Liste identifiziert. Bei dieser existiert für jede momentan angezeigte Ebene des Taxonomiebaums eine eigene vertikale Liste. Diese Listen wiederum sind horizontal angeordnet. Hierdurch kann durch eine einfache Markierung der aktuell gewählte Pfad nachverfolgt werden.

# 8. Ausblick

In diesem Kapitel wird ein Ausblick auf mögliche zukünftige Funktionen gegeben, die in nachfolgenden Arbeiten behandelt werden können. Zuerst wird ein Ausblick auf die für den Server und anschließend die für den Client, sprich die App, vorstellbaren Funktionen gegeben. Der letzte Abschnitt befasst sich mit Funktionen, die systemübergreifend sind.

## 8.1. Server

Momentan wird zur persistenten Speicherung der Daten eine relationale Datenbank eingesetzt. Die gegebene Taxonomie liegt in Form eines gerichteten azyklischen Graphen vor. Diese Art von Datenstruktur lässt sich nur auf Umwegen sinnvoll in einer relationalen Datenbank speichern. Um eine bessere Performance zu erreichen, könnte der Einsatz einer NoSQL Datenbank geprüft werden. Insbesondere die sogenannten Graphen-Datenbanken sind für Handhabung von Graphen ausgelegt und versprechen eine Verbesserung der Performance und Vereinfachung des Quellcodes.

Im Datenbankmodell sind Benutzer mit Taxonomien assoziiert. Dies ermöglicht es eine Taxonomie bestimmten Benutzern zuzuordnen und deren Zugriff nur auf die zugeordneten Taxonomien zu beschränken. Dies wäre eine sinnvolle Erweiterung des Systems, die momentan nicht verwendet wird.

In bestimmten Anwendungsfällen ist es sinnvoll Änderungen an einer Taxonomie nachzuverfolgen. Um eine Taxonomie zu verwenden muss ein Benutzer authentifiziert sein. Hierdurch ist dem Server bekannt wer momentan auf eine Taxonomie zugreift. Um eine Nachverfolgung zu ermöglichen, könnte der Server bei jeder Änderung einer Ressource den entsprechenden Benutzer und die Änderungszeit speichern.

## 8.2. Client

Die aus dem Projekt RIoT [SFB+15] übernommene Benutzer- und Rechteverwaltung implementiert eine REST Schnittstelle zur Verwaltung von Benutzern, Gruppen und Rechten. Hierfür könnte eine Benutzeroberfläche implementiert werden, die diese Verwaltung ermöglicht.

Wenn die Kindkonzepte eins Konzepts vom Server abgefragt werden, um diese im Taxonomiebaum anzuzeigen, so werden momentan alle vorhandenen auf einmal geladen und angezeigt. Bei sehr vielen Kindern benötigt dieser Vorgang eine gewisse Zeit. Um das Laden der Kindkonzepte zu beschleunigen, wäre es denkbar zu Beginn eine feste Anzahl an Kindern zu laden. Gelangt der Benutzer durch Scrollen am Ende der geladenen Konzepte an, so werden bei Bedarf weitere vom Server abgefragt. Diese

Funktion wird von dem verwendeten Framework *iScroll 5* unterstützt und trägt dort die Bezeichnung *Infinite Scrolling*.

Die Detailansicht verwendet momentan fest codiert 50% der Breite des Bildschirms. Wird die App mit hochkant gehaltenen Tablets oder auf Geräte mit kleineren Bildschirmen, wie beispielsweise Smartphones, verwendet, so wäre eine dynamische Anpassung je nach Bildschirmgröße denkbar.

## 8.3. Systemübergreifend

Die aus dem Projekt RIoT [SFB+15] übernommene Benutzer- und Rechteverwaltung ermöglicht es Benutzern verschiedene Rechte zuzuweisen. So könnte es nur bestimmten Benutzern erlaubt sein eine Taxonomie zu editieren. Andere könnten zwar das Recht besitzen eine Taxonomie zu betrachten, allerdings soll es diesen nicht möglich sein diese zu editieren. Zusätzlich können Benutzer Gruppen zugewiesen werden um die Rechteverwaltung zu vereinfachen. Momentan sind keine bestimmten Rechte für den Zugriff auf eine Ressource nötig.

Zum jetzigen Zeitpunkt kann eine Taxonomie ausschließlich in das in Kapitel 4.2 vorgestellte alte XML-basierte Format exportiert werden. Diese Format ist proprietär und kann somit nicht von außenstehenden Verstanden werden. Die View `Export` im Client ist für die Unterstützung mehrerer Export-Formate vorbereitet. Der Serverteil hingegen muss hierfür erweitert werden. Als mögliches Format kommt zum Beispiel das vom W3C vorgestellte *Simple Knowledge Organization System (SKOS)* [W3C09] in Frage.

Treten unerwartete Fehler auf, werden auf der Serverseite momentan Fehlermeldungen mit wenig technisches Details generiert. Durch das Einführen von verschiedenen Benutzergruppen wäre es denkbar an diese angepasste Fehlermeldungen zu generieren. Einem Administrator könnten somit Meldungen mit mehr Details als einem normalen Benutzer präsentiert werden.

In der von [ST10] erstellten Taxonomie gibt es Konzepte und Synonyme, die an keiner Stelle innerhalb des Taxonomiebaums eingeordnet sind. Momentan gibt es in dem entwickelten System keine Möglichkeit diese zu identifizieren. Durch eine Erweiterung könnten diese identifiziert und dem Benutzer präsentiert werden.

# A. Anhang

## A.1. Komplettes ER-Diagramm

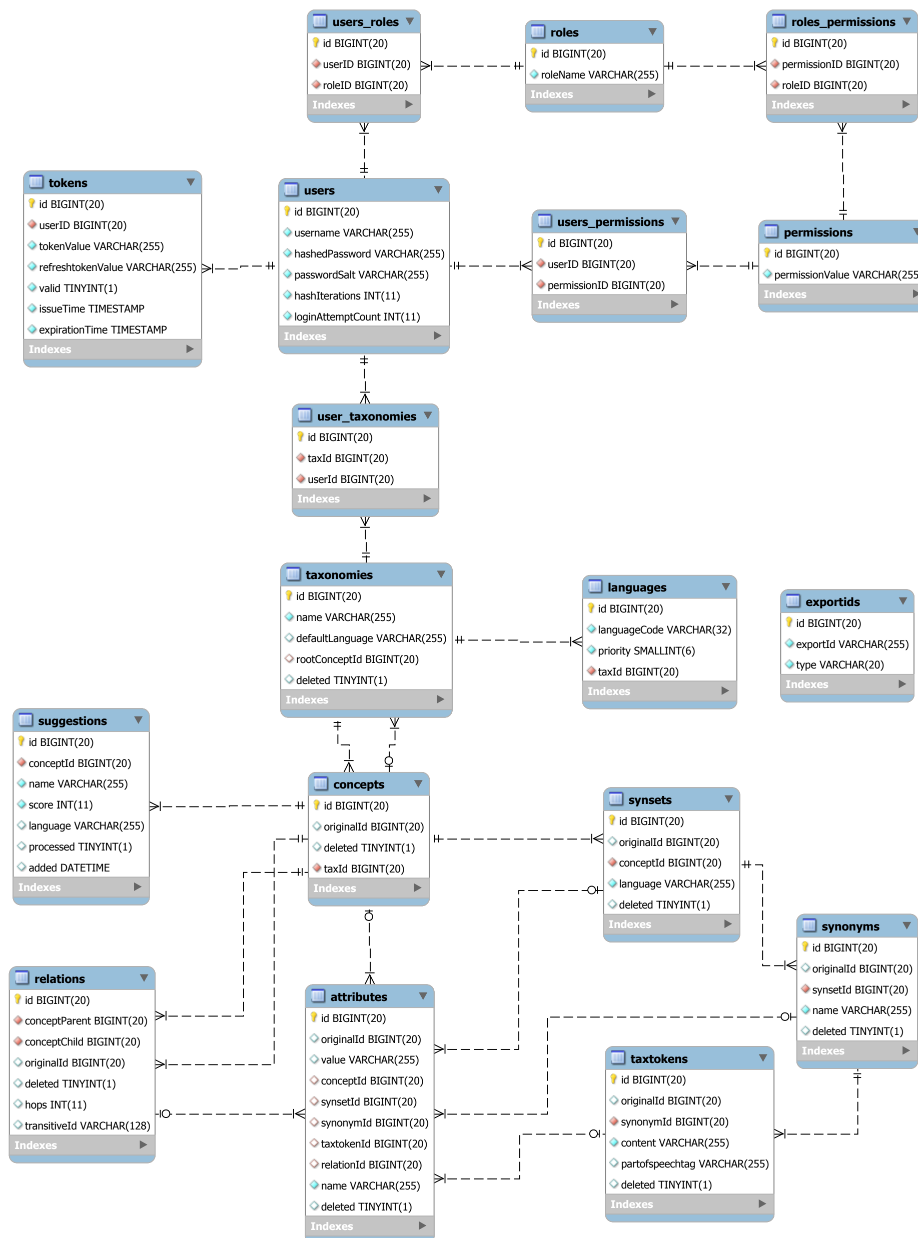



**Abbildung A.1.:** Komplettes ER-Diagramm (Exportiert aus MySQL-Workbench)

## A.2. Schritt zwei und drei des Importierens einer Taxonomie

Tools Taxonomies Search en

# Import new Taxonomy

**Step 1: Select or create taxonomy** (Taxonomy with name *automotive* selected)

**Step 2: Import taxonomy data**

Select the file with the data. In the legacy taxonomy format the file has the suffix _*data.tax*.

The file must be encoded with *UTF-16 BE (Big Endian)*.

Warning: The import process may take a few minutes!

**Data file** Datei auswählen Keine ausgewählt Upload

**Step 3: Import taxonomy relations**

**Abbildung A.2.:** Screenshot der App, der den zweiten Schritt des Importvorgangs zeigt

Tools Taxonomies Search en

# Import new Taxonomy

**Step 1: Select or create taxonomy** (Taxonomy with name *automotive* selected)

**Step 2: Import taxonomy data**

**Step 3: Import taxonomy relations**

Select the file with the relations. In the legacy taxonomy format the file has the suffix _*forschung.tax*.

The file must be encoded with *UTF-16 BE (Big Endian)*.

Warning: The import process may take a few minutes!

**Relations file** Datei auswählen Keine ausgewählt Upload

**Abbildung A.3.:** Screenshot der App, der den dritten Schritt des Importvorgangs zeigt

# Literaturverzeichnis


[Aab15] L. Aaberg. Sql2o: Easy database query library, 2015. URL `http://www.sql2o.org/`. (Zitiert auf Seite 51)

[Ado15] Adobe Systems Inc. PhoneGap, 2015. URL `http://phonegap.com/`. (Zitiert auf Seite 38)

[Ang] AngularUI Mitwirkende. AngularUI Router. URL `https://github.com/angular-ui/ui-router`. (Zitiert auf Seite 25)

[Ban13] M. Bank. *AIM - a Social Media Monitoring System for Quality Engineering*. Dissertation, Universität Leipzig, Leipzig, 2013. URL `http://nbn-resolving.de/urn:nbn:de:bsz:15-qucosa-115894`. (Zitiert auf Seite 15)

[BLFW+05] T. Berners-Lee, R. Fielding, W3C/MIT, Day Software, Larry Masinter, Adobe Systems. Uniform Resource Identifier (URI): Generic Syntax, 2005. URL `https://www.ietf.org/rfc/rfc3986.txt`. (Zitiert auf Seite 19)

[Chi06] L. Chittaro. Visualizing Information on Mobile Devices. *Computer*, 39(3):40–45, 2006. (Zitiert auf Seite 16)

[CJB99] B. Chandrasekaran, J. R. Josephson, V. R. Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, 1999. (Zitiert auf Seite 14)

[CZJ13] A. P. Chhetri, K. Zhang, E. Jain. ERELT. *VINCI '13*, S. 54–63, 2013. (Zitiert auf den Seiten 17 und 45)

[DLSW99] G. Dong, L. Libkin, J. Su, L. Wong. Maintaining the transitive closure of graphs in SQL. *In Int. J. Information Technology*, 5, 1999. (Zitiert auf den Seiten 17 und 64)

[EBB+14] K. Etemad, D. Baur, J. Brosz, S. Carpendale, F. F. Samavati. PaisleyTrees: A Size-Invariant Tree Visualization. *EAI Endorsed Transactions on Creative Technologies*, 1(1):e2, 2014. (Zitiert auf Seite 16)

[Erd08] K. Erdogan. A Model to Represent Directed Acyclic Graphs (DAG) on SQL Databases - CodeProject, 2008. URL `http://www.codeproject.com/Articles/22824/A-Model-to-Represent-Directed-Acyclic-Graphs-DAG-o`. (Zitiert auf Seite 17)

[Est15] M. Estel. Taxonomy extension through synonym discovery in integrated data. 2015. (Zitiert auf den Seiten 11, 60 und 97)

[Fie99] R. Fielding. Hypertext Transfer Protocol - HTTP/1.1, 1999. URL `https://www.ietf.org/rfc/rfc2616.txt`. (Zitiert auf Seite 21)

[Fie00] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. (Zitiert auf Seite 19)

[Fir15] M. Firtman. Mobile HTML5 compatibility, 2015. URL `http://mobilehtml5.org/`. (Zitiert auf Seite 37)

[Gar04] L. M. Garshol. Metadata? Thesauri? Taxonomies? Topic Maps! Making Sense of it all. *Journal of Information Science*, 30(4):378–391, 2004. (Zitiert auf Seite 13)

[Gar05] J. J. Garett. Ajax: A New Approach to Web Applications, 2005. URL `https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php`. (Zitiert auf Seite 22)

[GD13] J. Goll, M. Dausmann. *Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java*. SpringerLink : Bücher. Imprint: Springer Vieweg, Wiesbaden, 2013. (Zitiert auf Seite 23)

[Goo15a] Google Inc. AngularJS — Superheroic JavaScript MVW Framework, 06.06.2015. URL `https://angularjs.org/`. (Zitiert auf Seite 22)

[Goo15b] Google Inc. AngularJS: Developer Guide: Unit Testing, 2015. URL `https://docs.angularjs.org/guide/unit-testing`. (Zitiert auf Seite 26)

[Gos05] J. Gossman. Introduction to Model/View/ViewModel pattern for building WPF apps, 2005. URL `http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx`. (Zitiert auf Seite 23)

[HF97] B. Hamp, H. Feldweg. GermaNet - a Lexical-Semantic Net for German. In *In Proceedings of ACL workshop Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, S. 9–15. 1997. (Zitiert auf Seite 15)

[HGZ+10] J. Hao, C. A. Gabrysch, C. Zhao, Q. Zhu, K. Zhang. Visualizing and Navigating Hierarchical Information on Mobile User Interfaces. *International Journal of Advanced Intelligence*, (2):81–103, 2010. (Zitiert auf den Seiten 17 und 45)

[HS10] C. Hänig, M. Schierle. Relationsextraktion aus Fachsprache - ein automatischer Ansatz für die industrielle Qualitätsanalyse. *eDITion*, 2010. (Zitiert auf Seite 15)

[HZ] J. Hao, K. Zhang. A Mobile Interface for Hierarchical Information Visualization and Navigation. In *2007 IEEE International Symposium on Consumer Electronics*, S. 1–7. (Zitiert auf den Seiten 17 und 45)

[MBF+90] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, K. J. Miller. Introduction to WordNet: An On-line Lexical Database *. *International Journal of Lexicography*, 3(4):235–244, 1990. (Zitiert auf den Seiten 15 und 30)

[MX98] L. Masinter, Xerox. Returning Values from Forms: multipart/form-data: RFC 2388, 1998. URL `https://www.ietf.org/rfc/rfc2388.txt`. (Zitiert auf Seite 65)

[NS13] V. Nastase, M. Strube. Transforming Wikipedia into a Large Scale Multilingual Concept Network. *Artif. Intell.*, 194:62–85, 2013. (Zitiert auf Seite 15)

[OPC15] Oracle Inc., Project Kenai, Cognisync. JAX-RS, 2015. URL https://jax-rs-spec.java.net/. (Zitiert auf Seite 51)

[Ora14] Oracle Inc. GlassFish Server, 2014. URL https://glassfish.java.net/. (Zitiert auf Seite 51)

[Ora15] Oracle Inc. Java SE Technologies - JDBC, 2015. URL http://www.oracle.com/technetwork/java/javase/jdbc/index.html. (Zitiert auf Seite 51)

[OT15] M. Otto, J. Thornton. Bootstrap: The world's most popular mobile-first and responsive front-end framework., 2015. URL http://getbootstrap.com/. (Zitiert auf Seite 72)

[RRD07] L. Richardson, S. Ruby, T. Demmig. *Web-Services mit REST: [frischer Wind für Web-Services durch REST].* O'Reilly, Beijing and Cambridge and Farnham and Köln and Paris and Sebastopol and Taipei and Tokyo, dt. ausg., 1. aufl Auflage, 2007. (Zitiert auf Seite 19)

[SBRZ10] M. Spitters, R. Bonnema, M. Rotaru, J. Zavrel. Bootstrapping information extraction mappings by similarity-based reuse of taxonomies. *Reuse and Adaptation of Ontologies and Terminologies Workshop*, 2010. (Zitiert auf Seite 15)

[Sch11] M. Schierle. Language Engineering for Information Extraction. 2011. (Zitiert auf Seite 15)

[Sch13] L. Schmitz. Monty Widenius im Gespräch: "Die Zukunft von MySQL gestalten wir", 2013. URL http://www.computerwoche.de/a/die-zukunft-von-mysql-gestalten-wir,2533344,2#. (Zitiert auf Seite 51)

[SFB+15] N. Schnabel, Franco da Silva, Ana, D. Braunschweiger, P. Keck, C. Bäumlisberger, S. Schnaible, J. Tangermann, F. Gänßlen, R. Schulz, M. Lehwald, B. Giesel. RIoT: Revolutionary Internet of Things. 2015. (Zitiert auf den Seiten 8, 50, 53, 55, 60, 63, 97, 99 und 100)

[SPH11] H. Shin, G. Park, J. Han. Tablorer - An Interactive Tree Visualization System for Tablet PCs. *Computer Graphics Forum*, 30(3):1131–1140, 2011. (Zitiert auf den Seiten 16 und 44)

[ST08] M. Schierle, D. Trabold. Extraction of Failure Graphs from Structured and Unstructured Data. *Machine Learning and Applications, 2008. ICMLA '08*, S. 324–330, 2008. (Zitiert auf Seite 15)

[ST10] M. Schierle, D. Trabold. Multilingual Knowledge-Based Concept Recognition in Textual Data. In *Advances in Data Analysis, Data Handling and Business Intelligence*, S. 327–336. 2010. (Zitiert auf den Seiten 11, 15, 16, 30, 49 und 100)

[Til15] S. Tilkov. *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web.* dpunkt, Heidelberg, 3., aktualisierte und erw. aufl. Auflage, 2015. (Zitiert auf Seite 19)

[UG96] M. Uschold, M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136, 1996. (Zitiert auf Seite 14)

[W3C09] W3C. SKOS: Simple Knowledge Organization System Reference, 2009. URL http://www.w3.org/TR/2009/REC-skos-reference-20090818/. (Zitiert auf Seite 100)

[YC06] H. Y. Yoo, S. H. Cheon. Visualization by Information Type on Mobile Device. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60*, APVis '06, S. 143–146. Australian Computer Society, Inc, Darlinghurst, Australia, Australia, 2006. (Zitiert auf Seite 16)

Alle URLs wurden zuletzt am 23. 09. 2015 geprüft.

## Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift