



UNIVERSITÄT LEIPZIG  
FAKULTÄT FÜR MATHEMATIK UND INFORMATIK  
INSTITUT FÜR INFORMATIK

# **Integration von Daten der Semantic Web/Linked Data-Community in das ISOcat/RELcat-System**

**Masterarbeit**

Leipzig, 04.2013

vorgelegt von

Jochen Tiepmar

Studiengang Informatik

Betreuender Hochschullehrer : Prof. Dr. Gerhard Heyer



# Inhaltsverzeichnis

1 Einleitung, Motivation.....	3
1.1 CLARIN-D.....	3
1.2 ISOcat.....	4
1.3 RELcat.....	7
1.4 Motivation dieser Arbeit.....	9
2 Verwendete Techniken der Textverarbeitung .....	12
2.1.1 Terminologie-Extraktion.....	12
2.1.2 Porter-Stemming.....	12
2.1.3 Levenshtein-Distanz.....	13
3 Datenimport in ISOcat.....	15
3.1 Datenquelle.....	15
3.1.1 LOD-Cloud.....	17
3.2 Importmöglichkeiten von ISOcat/RELcat.....	19
3.2.1 ISOcat.....	19
3.2.2 RELcat.....	20
3.3 Umsetzung des Importes.....	21
3.4 Beispielimport.....	22
4 Finden von Kandidaten für Relationen zwischen Datenkategorien.....	25
4.1 Grundüberlegungen.....	25
4.2 Vergleichsmethoden.....	26
4.2.1 Namenssuche.....	26
4.2.2 Volltextsuche.....	28
4.2.3 Beispiel-Suche.....	32
4.2.4 WordNet.....	34
5 Online-Dienste.....	38
5.1 Webservices.....	38
5.2 Graphische Nutzeroberfläche.....	39
5.2.1 Kandidatensuche.....	39
5.2.2 Import von fremden Vokabularen.....	44
5.2.3 Beispieldurchlauf für <a href="http://www.lingvoj.org/lingvoj.rdf">http://www.lingvoj.org/lingvoj.rdf</a> .....	45
6 Evaluation.....	49
6.1 Evaluation der Datenaufbereitung.....	49
6.2 Evaluation des Imports.....	57
6.3 Evaluation der Suchmethoden.....	59
6.4 Recall, Precision und F-measure.....	65

7 Vorschläge für weiterführende Arbeiten.....	80
8 Zusammenfassung.....	81
9 Literaturverzeichnis.....	83
10 Anhang.....	84

|

# 1 Einleitung, Motivation

## 1.1 CLARIN-D

CLARIN-D<sup>1</sup> ist der deutsche Zweig von CLARIN, „einer web- und zentrenbasierten Forschungsinfrastruktur für die Geistes- und Sozialwissenschaften“[1], in der, gefördert vom Bundesministerium für Bildung und Forschung<sup>2</sup>, „Linguistische Daten, Werkzeuge und Dienste in einer integrierten, interoperablen und skalierbaren Infrastruktur für die Fachdisziplinen der Geistes- und Sozialwissenschaften bereitgestellt werden“[1] sollen. Dies umfasst Zentren in der Universität Tübingen, Universität Leipzig, BBAW Berlin, Universität Stuttgart, IDS Mannheim, LMU München, MPI Nijmegen, Universität Hamburg und der Universität des Saarlandes.



Abbildung 1: Zentren von CLARIN-D

Es werden unter anderem verschiedene Sprachressourcen, wie beispielsweise Korpora der Berlin-Brandenburgischen Akademie der Wissenschaften in Berlin und der englischen Sprach- und Übersetzungswissenschaft in Saarbrücken oder der Wortschatz der Universität Leipzig (und viele weitere) sowie teilweise webbasierte Tools zur Arbeit mit linguistischen Ressourcen zur Verfügung gestellt, um die digitale Arbeit mit Sprachen zu erleichtern.

1 <http://de.clarin.eu> [1]

2 <http://www.bmbf.de/>

Für die Arbeit mit digitalen Sprachressourcen ist es wichtig, einen Standard zu entwickeln, der vorgibt, wie die Daten aufgebaut sind. CMDI(Component MetaData Infrastructure)<sup>3</sup> bildet diesen Metadatenstandard für CLARIN und gibt den genauen Aufbau der Ressourcen vor. Je nach Anwendungsfall können dabei verschiedene Komponenten zusammengesetzt werden, um ein maßgeschneidertes Datenformat zu erhalten.

Folgendes Beispiel zeigt die Komponenten eines Beispieldatenformates namens Example\_Profile\_Instance, welche aus 3 Komponenten zusammengesetzt ist (example-component-photo, example-component-text und example-component-texttype) .

```
<CMD_ComponentSpec isProfile="true">
  <Header>
    <ID>clarin.eu:cr1:p_1290431694484</ID>
    <Name>Example_Profile_Instance</Name>
    <Description>Just a simple profile that combines several metadata components</Description>
  </Header>
  <CMD_Component CardinalityMax="1" CardinalityMin="1" name="Example_Profile_Instance">
  <CMD_Component CardinalityMax="1" CardinalityMin="1"
    ComponentId="clarin.eu:cr1:c_1290431694483"/>
  <CMD_Component CardinalityMax="1" CardinalityMin="1"
    ComponentId="clarin.eu:cr1:c_1290431694480"/>
  <CMD_Component CardinalityMax="1" CardinalityMin="1"
    ComponentId="clarin.eu:cr1:c_1290431694479"/>
  </CMD_Component>
</CMD_ComponentSpec>
```

Dieses Metadatenchema kann nun genutzt werden, um ein Datenschema für Daten anzulegen, die mit der Infrastruktur von CLARIN kompatibel sind.

Die Basis für dieses komponentenbasierte Metadatenmodell wird dabei von ISOcat bereitgestellt.

## 1.2 ISOcat

ISOcat<sup>4</sup> ist die Implementierung des ISO 12620:2009<sup>5</sup> Standards und wird verwendet, um dessen Konsistenz zu überprüfen und zu validieren. Dadurch soll ein allgemein akzeptierter Standard für die Definition von linguistischen Konzepten erreicht werden („Defining widely accepted linguistic concepts“[2]), indem für linguistische Konzepte Datenkategorien definiert werden, die als Webressource

---

3 <http://www.clarin.eu/CMDI>

4 <http://www.isocat.org/index.html>[2]

5 [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=37243](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37243)[3]

referenziert werden können. Datenkategorien selbst sind dabei nicht auf linguistische Konzepte beschränkt, beispielsweise existieren Datenkategorien für Dateiformate wie Video<sup>6</sup> oder Audio<sup>7</sup>.

Die Grundlage der Datenkategorien ist das DCR-Datenmodell<sup>8</sup>, welches in Form von dcif (**D**ata **C**ategory **I**nterchange **F**ormat) den Aufbau der Daten beschreibt. Einen Überblick über dieses Datenformat liefert Abbildung 2.

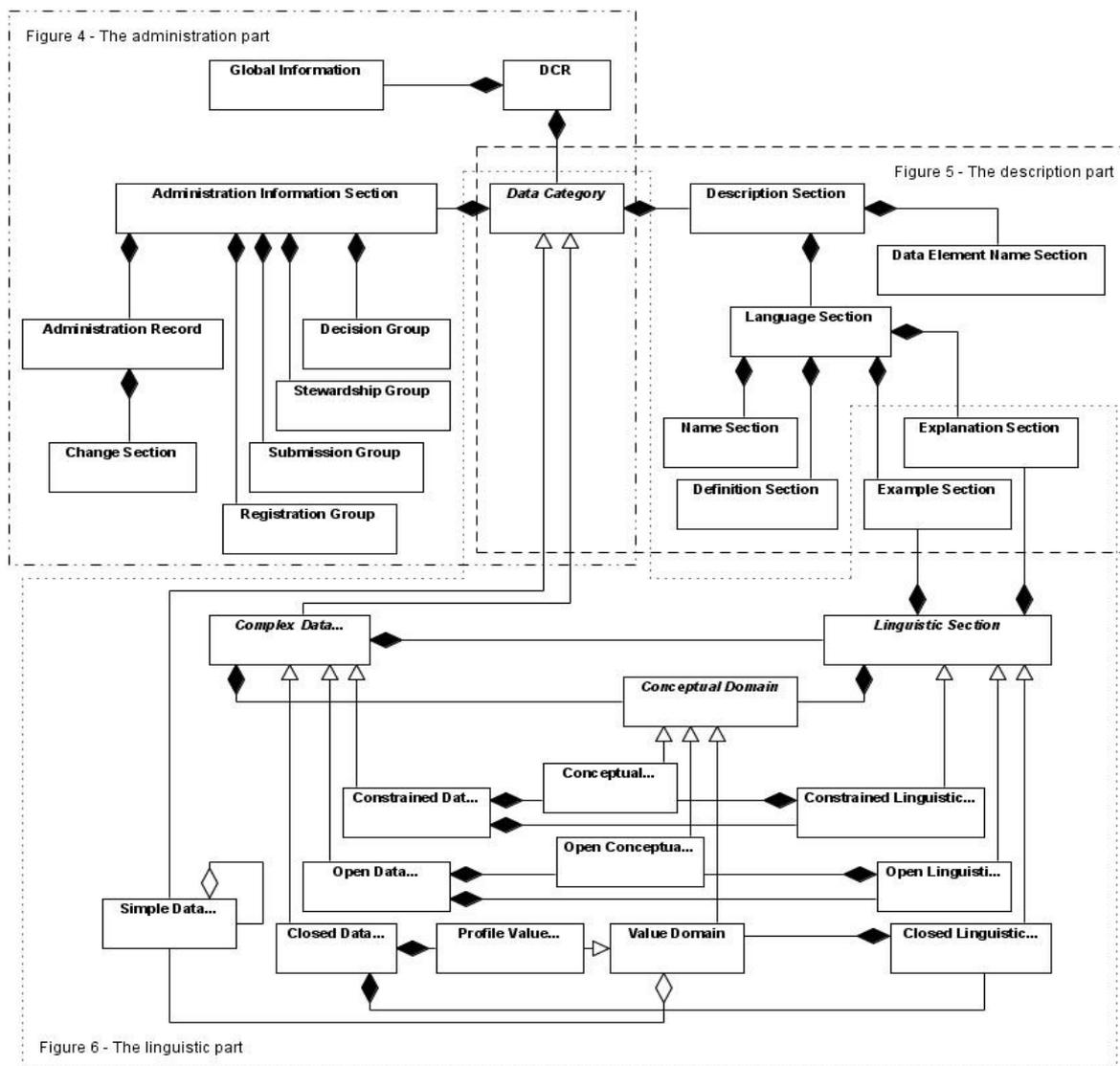


Abbildung 2: Überblick über das DCR-Datenmodell

6 <http://www.isocat.org/rest/dc/2654>

7 <http://www.isocat.org/rest/dc/2653>

8 Gesamtes Modell : [https://catalog.clarin.eu/isocat/12620/model/DCR\\_data\\_model.pdf](https://catalog.clarin.eu/isocat/12620/model/DCR_data_model.pdf)[4]

Datenkategorien sind dabei in 3 Abschnitte aufgeteilt:

- der administrative Abschnitt enthält Informationen, die die Datenkategorie aus organisatorischer Sicht beschreiben : Zu wem gehört sie?, Wann und warum wurde sie zuletzt geändert? (...)
- der deskriptive Abschnitt beschreibt das Konzept hinter dieser Datenkategorie anhand von Definitionen, Erläuterungen und Beispielen näher. Dabei wird nach verschiedenen Sprachabschnitte getrennt.
- der linguistische Teil einer Datenkategorie beschreibt Meta-Eigenschaften dieser Datenkategorie, wie die Unterscheidung zwischen komplexen und einfachen Datenkategorien. Dieser Teil spielt für diese Arbeit eine untergeordnete Rolle.

Eine Datenkategorie ist dabei über deren PID (Persistent IDentifier) eindeutig referenzierbar. Diese PID besteht immer aus 'DC-' und einer Zahl.

Beispielsweise existiert eine Datenkategorie 'Adjectival'<sup>9</sup>, die über die PID <http://www.isocat.org/datcat/DC-3066> (oder kurz DC-3066) eindeutig definiert und näher beschrieben ist.

Die in ISOCat enthaltenen Daten können öffentlich oder privat sein. Während die privaten Daten für diese Arbeit verständlicherweise nicht zur Verfügung stehen, sind alle öffentlichen Datenkategoriesets und Profile über den Gast-Zugang<sup>10</sup> abrufbar. Darin enthalten sind 3365 Datenkategorien in 71 Datenkategoriesets und 13 Profilen. Die Profile sind dabei Zusammenstellungen einzelner Datenkategoriesets, enthalten aber auch 1288 Datenkategorien, die nicht in den einzeln aufgelisteten Sets vorkommen<sup>11</sup>.

44 Datenkategoriesets enthalten dabei im Vergleich zu den jeweils anderen neue Datenkategorien. Eine Auflistung dieser Datenkategoriesets befindet sich auf der dieser Arbeit beiliegenden CD<sup>12</sup>.

Für das Erstellen oder Bearbeiten von Datenkategorien kann ein Webinterface<sup>13</sup> genutzt werden, über welches sich nach der Anmeldung Datenkategorien in einem

---

9 <http://www.isocat.org/rest/dc/3066>

10 <http://www.isocat.org/rest/user/guest/workspace>

11 [data/isocat/dc\\_in\\_profile.csv](data/isocat/dc_in_profile.csv)

12 <isocat/workspacecleaned.txt>

13 <https://catalog.clarin.eu/isocat/interface/index.html>

privaten Arbeitsplatz erstellen lassen. ISOcat ist ein offenes System, es kann sich also jeder anmelden und eigene Datenkategorien erstellen und nutzen oder öffentlich zugängliche Datenkategorie referenzieren.

Verschiedene Datenkategoriesets können dabei durchaus gleiche Konzepte mit unterschiedlichen Datenkategorien beschreiben oder Datenkategorien anderer Sets erweitern. In dem Set Edisyn<sup>14</sup> gibt es Datenkategorien, die das Konzept, welches in DC-3066 beschrieben ist, weiter differenzieren, indem sie für Geschlecht (f, m), Zahl (sg, pl) und Art (comparative, superlative, diminutive) des Adjektivs jeweils einzelne Datenkategorien definieren. Es erscheint schnell sinnvoll, Relationen zwischen Datenkategorien zu erstellen, die diese miteinander in Beziehung setzen können. ISOcat wurde allerdings ausdrücklich nicht für diesen Zweck geschaffen („On purpose ISOcat contains a shallow list of data categories, i. e. , only few relationships are allowed.“)[5]. Um dies dennoch zu ermöglichen, wurde RELcat zusätzlich zu ISOcat entwickelt.

### 1.3 RELcat

RELcat<sup>15</sup> ist ein ISOcat erweiternder Katalog von Relationen zwischen Datenkategorien, der es jedem Nutzer ermöglichen soll, Relationen zwischen Datenkategorien ISOcats oder von einer Datenkategorie zu einer externen Ressource zu definieren. Folgende Hierarchie von Relationen wurde dabei in RELcat als Standard definiert ([6], Seite 3):

1	rel:related	
1.1	rel:sameAs	
1.2	rel:almostSameAs	
1.3	rel:broaderThan	
1.3.1	rel:superClassOf	
1.3.2	rel:hasPart	
1.3.2.1		rel:hasDirectPart
1.4	rel:narrowerThan	
1-4-1	rel:subClassOf	
1.4.2	rel:partOf	
1.4.2.1		rel:directPartOf

Dadurch können beispielsweise die verschiedenen Datenkategorien, die in Edisyn unterschiedliche Arten von Adjektiven beschreiben, als Unterkategorien des Adjektivs aus Gold2010 eingeordnet werden oder Datenkategorien, die die gleichen Konzepte

14 <https://catalog.clarin.eu/isocat/rest/dcs/362>

15 <http://lux13.mpi.nl/relcat/site/index.html>[5]

beschreiben, mittels einer sameAs-Relation verknüpft werden.

Diese Hierarchie stellt einen Ausgangspunkt dar und kann erweitert werden, indem neue Relationen an passender Stelle eingefügt werden. In [6] werden beispielhaft Relationen von owl<sup>16</sup> und skos<sup>17</sup> folgendermaßen eingeordnet ([6], Seite 3).

- 1. 1 rel:sameAs
  - 1. 1. 1 owl:sameAs
  - 1. 1. 2 owl:equivalentClass
  - 1. 1. 3 owl:equivalentProperty
  - 1. 1. 4 skos:exactMatch
- 1. 2 rel:almostSameAs
  - 1. 2. 1 skos:closeMatch

Diese Flexibilität ermöglicht generische Algorithmen, die ohne genaue Kenntnis der verwendeten Vokabulare mit den Relationen arbeiten können, sowie die Verwendung von bereits existierenden linguistischen Ressourcen ohne diese anpassen zu müssen ([6], Seite 3).

Damit Nutzer sich bei Anfragen auf bestimmte Sets beziehen können, wird ein Quad-Store, namentlich OpenAnzo<sup>18</sup>, verwendet, der Fakten nach Subjekt, Prädikat, Objekt und dem Set unterscheiden kann, wobei das Set hier über den Graph differenziert wird. Als Abfragesprache wird SPARQL<sup>19</sup> (**SPARQL Protocol And RDF Query Language**) genutzt. Zum aktuellen Zeitpunkt werden dabei die Sets<sup>20</sup> CMDI, dc, relish, gold, edisyn, vlo und tds angeboten.

RELcat befindet sich aktuell im Alpha-Status, „RELcat is in its early stages of development. This means that anything can change at any time. It's also only available on a test server. This means that the server can go down anytime and data might get unrecoverably lost at any time. So this alpha release of RELcat is **not** meant and **not** ready for day to day work!“[5] Die Arbeit mit RELcat findet also noch auf einer experimentellen Ebene statt und es ist möglich, dass dies in Zukunft zu Anpassungsbedarf des in dieser Arbeit entwickelten Systems führt. Dies kann insbesondere die Datenabfrage betreffen, bei der die Queries im Moment per Http-Requests an einen Webservice gesendet werden<sup>21</sup>.

---

16 <http://www.w3.org/2002/07/owl#>

17 <http://www.w3.org/2004/02/skos/core>

18 <http://www.openanzo.org/>

19 <http://www.w3.org/TR/RDF-sparql-query/>

20 <http://lux13.mpi.nl/relcat/set/CMDI>, <http://lux13.mpi.nl/relcat/set/dc>

21 Eine Beispielanfrage für 1 Set [http://lux13.mpi.nl/relcat/set/gold/query?](http://lux13.mpi.nl/relcat/set/gold/query?query=SELECT+DISTINCT+?o+WHERE+{{isocat:DC-3055+rel:sameAs+?o}})

query=SELECT+DISTINCT+?o+WHERE+{{isocat:DC-3055+rel:sameAs+?o}}

Es ist laut Menzo Windhouwer zu erwarten, dass hier noch ein richtiger SPARQL-Endpunkt bereitgestellt wird „The API supports SPARQL but I should work on making it a real SPARQL endpoint.“<sup>22</sup>

Möchte man eine Anfrage über mehrere Sets stellen, kann man diese als Parameter in der URL<sup>23</sup> auflisten.

Die Anfragen an den Webservice sind dabei aktuell dahingehend beschränkt, dass vorhandene Daten abgefragt werden können, Einfüge- und Lösch-Operationen allerdings nicht durchführbar sind[5].

Aus Gründen der Übersichtlichkeit und der Kompatibilität mit zukünftigen Iterationen von RELcat ist diese Arbeit auf die von RELcat standardmäßig genutzte Relationshierarchie beschränkt.

## 1.4 Motivation dieser Arbeit

In dieser Arbeit sollen weitestgehend automatisierte Möglichkeiten geschaffen werden, mit denen man auf möglichst unkomplizierte Art und Weise neue Datenkategorien aus anderen Ressourcen in ISOcat importieren kann.

Durch einen einfachen Import von Vokabularen würden die Datenformate, die von ISOcat und CLARIN bereitgestellt werden, für eine breitere Anwendergruppe Relevanz finden und sich weiter als Standard für sprachwissenschaftliche Disziplinen durchsetzen. Idealerweise würden damit andere Standards von ISOcat mit übernommen werden können, was die Interoperabilität verschiedener wissenschaftlicher Institute fördern und die Arbeit derer, die diesen Instituten computergestützte Werkzeuge zur Verfügung stellen möchten, stark vereinfachen würde.

Ein zweites Ziel dieser Arbeit besteht darin, dabei Datenkategorien mit anderen Ressourcen über RELcat in Relation zu setzen. Neu erzeugte Datenkategorien können daraufhin untersucht werden, ob sie in ISOcat bereits enthalten sind um Dubletten zu vermeiden. Die Erkennung von Dubletten wird aktuell über das Webinterface der ISOcat ermöglicht. Dabei wird anhand des Identifiers einer neu

---

<sup>22</sup> references/email/Re import.eml

<sup>23</sup> Eine Beispielanfrage für alle Sets <http://lux13.mpi.nl/relcat/query?set=CMDI&set=gold&set=dc&set=relish&set=edisyn&set=vlo&set=tds&query=SELECT+DISTINCT+?o+WHERE+{{isocat:DC-3055+rel:sameAs+?o}}>

erzeugten Datenkategorie der bisherige Datenbestand dahingehend durchsucht, ob dieser Identifier schon vorhanden ist. Dies setzt allerdings wiederum die Verwendung des Webinterfaces voraus, was es erforderlich macht, die Datenkategorien einzeln abzarbeiten.

Weiter wäre dieses Feature dafür verwendbar, in der ISOcat „aufzuräumen“ – viele Datenkategorien beschreiben Konzepte, die bereits durch andere Datenkategorien beschrieben sind<sup>24</sup>. Durch das Auffinden von Kandidaten für Relationen ließen sich hier veraltete Einträge suchen und gegebenenfalls löschen oder eine allgemeine Strukturierung zwischen diesen Datenkategorien aufbauen, die das gesamte System übersichtlicher machen würde.

Dabei ist es wichtig hervorzuheben, dass das automatische Finden von Relationen ein Problem ist, welches vom Import unabhängig gestaltet werden kann, indem es auf der Ebene der Datenkategorien, und damit nach dem eigentlichen Import, umgesetzt wird.

Ein weiterer wichtiger Punkt sind die hohen Qualitätsansprüche, die an die in ISOcat und RELcat enthaltenen Informationen gestellt werden. Da diese das Fundament für weitere Forschungen bilden, ist es wichtig, dass keine unwahren Informationen enthalten sind. Dafür wird in Kauf genommen, dass nicht jede mögliche Information beschrieben ist, was allerdings generell nur sehr schwer umsetzbar wäre, da in den geisteswissenschaftlichen Disziplinen beschriebene Konzepte mitunter nicht frei von Interpretationsspielräumen sind. Um diesem Qualitätsanspruch gerecht zu werden, wird in dieser Arbeit darauf Wert gelegt, dass stets nur nach Kandidaten oder Vorschlägen für Relationen gesucht wird, die anschließend manuell bestätigt werden müssen.

Daraus ergeben sich die beiden Grundaufgaben dieser Arbeit:

- Importieren neuer Daten als Datenkategorien
- Finden von Kandidaten für Relationen zwischen Datenkategorien

Um die Qualität der Ergebnisse zu sichern und gleichzeitig die zweite Aufgabe vernünftig einzugrenzen, ist die praktische Relevanz der Kandidatensuche dabei auf

---

<sup>24</sup> Es gibt etliche Beispiele hierfür.

DC-1227 und DC-3064 beschreiben 'active voice'.

DC-331, DC-1418 und DC-2774 beschreiben 'abbreviation'

die in ISOcat enthaltenen Daten beschränkt. Dies folgt direkt daraus, dass Datenkategorien in RELcat mit externen Ressourcen verknüpft werden können. Würde man als Datenkategorie nun Ressourcen verwenden, die noch nicht in RELcat enthalten sind, wäre der Zweck der RELcat – Relationen mit Bezug zu Datenkategorien anzubieten – nicht mehr gesichert.

Das Finden von Kandidaten ist also in der Praxis nur nach dem abgeschlossenen Import und damit auf die in ISOcat enthaltenen Daten beschränkbar. Eine Erweiterung auf potentielle Datenkategorien wäre sogar kontraproduktiv, da die in der Relation angegebene PID einer Datenkategorie von ISOcat vergeben wird, also nur zur Verfügung steht, wenn die Datenkategorie in ISOcat importiert wurde.

*„What will be overwritten?”*

The DCIF is originally intended to export existing data categories so contains some fields which are managed by the system are also included and mandatory. For the following fields the system will use its default values instead of the ones provided by you:

- PIDs: just invent your own URI, e. g. , my:DC-1, and use them to relate DCs (closed DC conceptual domain to simple DCs and simple DCs with an is-a relation to another simple DC); these PIDs will be overwritten by ISOcat PIDs (unless you have ISOcat acceptable PIDs, contact the ISOcat system administrator to find out)“[7]

## 2 Verwendete Techniken der Textverarbeitung

Folgende Techniken der Textverarbeitung wurden in dieser Arbeit verwendet und seien deshalb noch einmal kurz erläutert.

### 2.1.1 Terminologie-Extraktion

Um Volltextbeschreibungen miteinander vergleichbar zu machen, ist es vorteilhaft, sich auf die Begriffe zu beschränken, die für die Beschreibung wichtig sind und unwichtige Begriffe, wie zum Beispiel Stoppwörter, zu ignorieren. Das Herausfiltern wichtiger Begriffe aus Volltext nennt man Terminologie-Extraktion. Die Toolbox der ASV Leipzig bietet dafür ein Verfahren an, welches eine auf Wortstatistik und Mustererkennung basierende Terminologie-Extraktion umsetzt und für diese Arbeit verwendet wurde. Dabei werden die Wörter aus einem Text extrahiert, die mit einer festgelegten Mindestsignifikanz im Vergleich zu einem Vergleichskorpus selten sind und somit für den Text signifikant –“The most important statistical method is the so-called *differential analysis* which measures the extent to which the frequency of a word *w* in the given text deviates from its frequency in general usage. The latter frequency is determined using a *reference corpus*, i.e. a large and well-balanced collection of documents in the given language.”([8], Introduction). Je öfter also ein Begriff im Text und dabei seltener im Referenzkorpus vorkommt, desto signifikanter ist er für den Text.

Zusätzlich werden Stoppwörter, also Wörter, die für sich genommen keine Bedeutung haben – wie beispielsweise 'und' oder 'oder' – aussortiert.

### 2.1.2 Porter-Stemming<sup>25</sup>

Im allgemeinen bezeichnet Stemming oder Grundformreduktion die Reduktion eines Wortes auf seinen Wortstamm. Dadurch erreicht man eine Standardisierung von idealerweise zusammengehörigen Wortformen.

Der von Martin Porter entwickelte Porter-Algorithmus<sup>26</sup> „is a process for removing the commoner morphological and inflexional endings from words in English“[9], ist also

<sup>25</sup> Verwendete Implementierung <http://alvinalexander.com/java/jwarehouse/lucene-1.3-final/src/java/org/apache/lucene/analysis/PorterStemmer.java.shtml>

<sup>26</sup> <http://tartarus.org/martin/PorterStemmer/index.html>[9]

ein Prozess, bei dem die für gewöhnlich auftretenden morphologischen und durch Flexion entstehenden Endungen eines Wortes entfernt werden sollen. Dadurch können Wörter normalisiert und für Information Retrieval Systeme nutzbar gemacht werden.

Der Algorithmus besteht aus 5 Schritten, in denen nach speziellen Regeln die Suffixe eines Wortes reduziert und ersetzt werden. Die einzelnen Schritte werden in Martin Porters Paper [10] ausführlich beschrieben und funktionieren seitdem unverändert.

Es kann passieren, dass es dabei zu Overstemming kommt, also Wörter auf einen gemeinsamen Stamm reduziert werden, die keinen gemeinsamen Stamm haben. Beispielsweise werden sowohl university als auch universe auf den Stamm univers reduziert. Da es sich bei den hier relevanten Texten aber um einen eingeschränkten Themenbereich handelt, ist dieses Problem vernachlässigbar.

Ein weiteres wichtiges Detail liegt darin, dass der Algorithmus nicht nur reduziert, sondern auch ersetzt. Schritt 1c in [10] liefert hierfür das Beispiel 'happy', welches auf 'happi' reduziert wird. 'happi' wird anschließend nicht weiter reduziert. Würde man davon ausgehen, dass durch Porter reduzierte Begriffe ausschließlich verkürzte Wörter sind, würde man einen Informationsverlust riskieren<sup>27</sup>.

### **2.1.3 Levenshtein-Distanz<sup>28</sup>**

Die von dem russischen Mathematiker Wladimir Iossifowitsch Lewenstein entwickelte Levenshtein-Distanz oder Editier-Distanz gibt die Mindestanzahl der Einfügungen, Ersetzungen und Löschungen an, um ein Wort in ein anderes Wort umzuformen und eignet sich zur Messung der Distanz – also dem Gegenteil von Ähnlichkeit – zweier Zeichenketten untereinander.

Die Levenshtein-Distanz hat 4 Eigenschaften<sup>29</sup>

- Sie beträgt mindestens den Unterschied der Längen beider Zeichenketten.
- Sie beträgt höchstens die Länge der längeren Zeichenkette.
- Sie ist dann und nur dann 0, wenn beide Zeichenketten identisch sind.
- Sie ist nicht größer als die Hamming-Distanz plus dem Längenunterschied.

---

<sup>27</sup> Dies wird für die Suche von Kandidaten anhand des Namens noch wichtig werden.

<sup>28</sup> Verwendete Implementierung : <http://mrfoo.de/archiv/1176-Levenshtein-Distance-in-Java.html>

<sup>29</sup> <http://de.wikipedia.org/wiki/Levenshtein-Distanz>[11]

der Zeichenketten.

## 3 Datenimport in ISOcat

### 3.1 Datenquelle

Es gibt mehrere Ansätze, um Wissen so zu beschreiben, dass es sowohl für einen Menschen als auch für einen Computer verständlich ist. In der Regel wird dabei versucht, Informationen für Menschen lesbar und „sinnvoll“ und andererseits genügend strukturiert für eine algorithmische Auswertung zu gestalten.

Einer dieser Ansätze ist das Semantic Web oder LOD (Linking Open Data). Dabei werden als Tripel dargestellte Fakten erstellt, die aus einem Subjekt, einem Prädikat und einem Objekt bestehen. Die so formulierten Fakten sind aufgrund ihrer festen Struktur für Maschinen auswertbar und durch die Ähnlichkeit zum grundlegenden Muster des menschlichen Satzbaus für den Mensch verständlich und lesbar.

Subjekt	Prädikat	Objekt
<ex:JochenTiepmar>	<dc:author>	„Integration von Daten der...“
<ex:JochenTiepmar>	<ex:student>	<uni:Leipzig>

<...> bezeichnet dabei eine Ressource, die durch eine URI<sup>30</sup> fest definiert ist. Subjekt und Prädikat müssen durch URIs spezifizierte Ressourcen sein, das Objekt kann sowohl Ressource als auch ein Literal – also beispielsweise eine Zeichenfolge oder eine Zahl – sein.

Die Abkürzung vor ':' steht für das Vokabular, in welchem das Prädikat definiert wurde. dc: steht beispielsweise für das Vokabular Dublin Core<sup>31</sup>, in dem ein Prädikat 'author' beschrieben ist, das hier verwendet wird, um den Autor eines Dokumentes zu kennzeichnen. Vokabulare bilden in der Regel<sup>32</sup> eine Menge von thematisch zusammengehörigen Prädikatsdefinitionen und können dabei frei definiert werden, das heißt, es steht jedem Anwender frei, ob er ein vorhandenes Vokabular nutzt oder ein eigenes erstellt. Die Verwendung eines etablierten Vokabulars bringt die Vorteile

30 [https://de.wikipedia.org/wiki/Uniform\\_Resource\\_Identifier](https://de.wikipedia.org/wiki/Uniform_Resource_Identifier)

31 [http://purl.org/metadata/dublin\\_core#](http://purl.org/metadata/dublin_core#)

32 Durch den offenen Charakter der Semantic Web können solche Dinge nicht zwingend vorausgesetzt werden.

der Standardisierung mit sich.

Vokabulare können für diese Arbeit grob in 2 Gruppen eingeteilt werden:

Sie können beschreibenden Charakter haben, wie das friend of a friend (foaf) Projekt, welches das Ziel hat, Informationen, die üblicherweise in sozialen Netzwerken verwendet werden, als ein solches Vokabular zu beschreiben („creating a Web of machine-readable pages describing people, the links between them and the things they create and do“)<sup>33</sup>. Diese Vokabulare beschreiben das Datenmodell für die darin formulierten Daten.

Die zweite wichtige Gruppe von Vokabularen umfasst solche, die Funktionalität bereitstellen, beispielsweise um Subklassenbeziehungen zu ermöglichen. Neben RDF und RDFS tritt hier vor allem OWL in den Vordergrund, ein vom World Wide Web Consortium<sup>34</sup> definierter Standard zur Beschreibung von Ontologien. OWL wurde dabei in 3 Abstufungen definiert: OWL Lite, OWL DL und OWL Full, die in dieser Reihenfolge komplexere Funktionen bereitstellen. OWL DL entspricht dabei der Beschreibungslogik (**Description Logic**) SHOIN(D) und ist bei maximaler Ausdrucksstärke in endlicher Zeit entscheidbar („maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time)“)[12]. OWL Full erweitert diese Ausdrucksstärke auf die komplette RDF-Syntax, kann aber die Berechenbarkeit nicht mehr garantieren („maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.“)[12] Diese Vokabulare bilden das Datenmodell zu der vorher beschriebenen Gruppe und beschreiben, welche Eigenschaften die darin enthaltenen Prädikate haben können, also dass beispielsweise gilt: <dc:author> <owl:subProperty> <foaf:person> – Autor ist eine Unterkategorie von Person.

Durch die Ausweitung dieser Art der Datenmodellierung kann ein Netzwerk aus Aussagen gebildet werden, die es einem Computer ermöglichen, Schlüsse zu ziehen und damit Verständnis vorzutauschen, indem er die Begriffe zueinander in Verbindung setzen kann. Im oben angegebenen Beispiel wäre zum Beispiel der Schluss, dass ein Student der Universität Leipzig besagte Arbeit verfasst hat, implizit

---

33 <http://www.foaf-project.org/>

34 <http://www.w3.org/>

folgerbar.

Eine Ontologie besteht mindestens aus einem Vokabular und enthält optional Instanzdaten, die in diesem Vokabular formuliert wurden.

Für diese Arbeit sind Definitionen aus der ersten Gruppe von Vokabularen relevant. Prädikate, die in der zweiten Gruppe definiert werden, stellen zwar Strukturinformationen dar, allerdings bringt die Vermischung des Importes mit dem Suchen nach Kandidaten für Relationen, wie in Kapitel 1.4 beschrieben, praktische Probleme mit sich, die zu unerwünschte Ergebnisse führen können.

Diese Beschreibung deckt das Semantic Web natürlich nur oberflächlich ab. Die Arbeitsgruppe AKSW<sup>35</sup>(Agile Knowledge Engineering and Semantic Web) der Universität Leipzig beschäftigt sich ausgiebig mit Anwendungsfällen und Technologien des Semantic Web und ist ein kompetenter Ansprechpartner für den interessierten Leser.

Eine in sich kompatible Sammlung solcher Ontologien bietet die LOD-Cloud, die für diese Arbeit als Datenquelle herangezogen wurde. Weiter Projekte mit ähnlichem Ziel sind LOD2<sup>36</sup> oder Datahub<sup>37</sup> und viele weitere. LOD ist dabei in dem Projekt Datahub enthalten, und es gibt generell eine Vielzahl von Verknüpfungen zwischen diesen Projekten.

### **3.1.1 LOD-Cloud**

LOD (Linking Open Data) steht für ein Projekt, welches sich zur Aufgabe gemacht hat, verschiedene Konzeptbeschreibungen zu sammeln, die untereinander kompatibel sind.

Die LOD-Cloud umfasst 355 Ontologien<sup>38</sup>, die sich folgendermaßen auf Wissensgebiete verteilen<sup>39</sup>:

---

35 <http://aksw.org/>

36 <http://aksw.org/Projects/LOD2.html>

37 <http://datahub.io/de/>

38 <http://datahub.io/group/lodcloud?tags=lod>

39 Die Angaben über die Mengen wurden widersprüchlich angegeben, aber geben trotzdem einen groben Einblick in die Verteilung.

Domain	Number of datasets	Triples	%	(Out-)Links	%
Media	25	1,841,852,061	5.82 %	50,440,705	10.01 %
Geographic	31	6,145,532,484	19.43 %	35,812,328	7.11 %
Government	49	13,315,009,400	42.09 %	19,343,519	3.84 %
Publications	87	2,950,720,693	9.33 %	139,925,218	27.76 %
Cross-domain	41	4,184,635,715	13.23 %	63,183,065	12.54 %
Life sciences	41	3,036,336,004	9.60 %	191,844,090	38.06 %
User-generated content	20	134,127,413	0.42 %	3,449,143	0.68 %
	295	31,634,213,770		503,998,829	

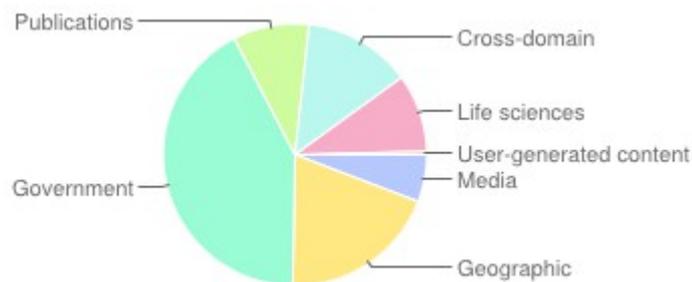
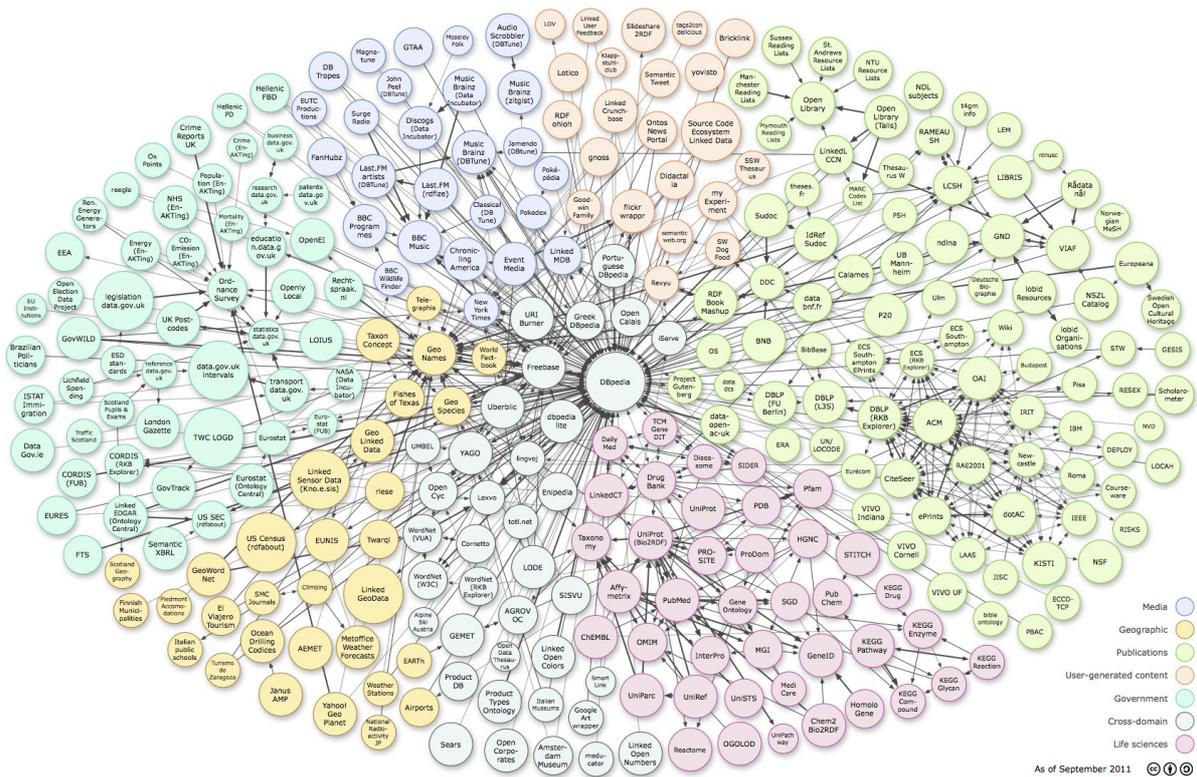


Abbildung 3: Themenverteilung in der LOD-Cloud

Die Themengebiete der in der LOD-Cloud enthaltenen Ontologien sind also sehr breit gefächert und nicht auf den Fokus von ISOcat zugeschnitten, was aber für diese Arbeit kein Nachteil ist, da es nicht zur Aufgabe gehört, passende Ontologien zu finden.

Abbildung 17 stellt alle Vokabulare, die in der LOD-Cloud enthalten sind, dar und soll den Umfang dieses Projektes verdeutlichen<sup>40</sup>:

<sup>40</sup> Quelle:[http://de.wikipedia.org/wiki/Datei:LOD\\_Cloud\\_Diagram\\_as\\_of\\_September\\_2011.png](http://de.wikipedia.org/wiki/Datei:LOD_Cloud_Diagram_as_of_September_2011.png)



Verbindungen zwischen den Punkten entsprechen dabei Verknüpfungen zwischen den Ontologien.

Eine weitere, sehr auf diese Arbeit zugeschnittene, Datenquelle steht unter der URL <http://datahub.io/dataset?q=linguistic&tags=linguistics> zur Verfügung. Zum Zeitpunkt des Schreibens dieser Arbeit werden 90 Ontologien mit dem Tag 'linguistics' gelistet. Unter anderem findet sich hier auch die Gold2010-Ontologie wieder, die in ISOcat ebenfalls gefunden werden kann.

## 3.2 Importmöglichkeiten von ISOcat/RELcat

### 3.2.1 ISOcat

Es gibt 2 Wege, um Datenkategorie in ISOcat zu erzeugen und zu bearbeiten.

Über ein Webinterface können nach Anmeldung Datenkategorien im eigenen Profil erstellt und bearbeitet werden. Da dabei jede Änderung manuell durchgeführt werden muss, ist dies keine Möglichkeit, die für die Arbeit weiter verfolgt wurde.

Besser geeignet für den Import von größeren Datenmengen ist die von Menzo

Windhouwer in einem Forumspost[7] beschriebene Methode. Dabei wird ein dcif - Dokument erzeugt, welches die Informationen über das zu importierende Datenkategorieset beschreibt und vom Systemadministrator der ISOcat validiert und gegebenenfalls eingefügt wird.

Ein dcif-Dokument beschreibt dabei genau ein Datenkategorieset und die darin enthaltenen Datenkategorien.

<http://www.isocat.org/12620/examples/dcif-example.dcif> beziehungsweise die Datei <references/httpbackups/dcif-example.dcif.xml> veranschaulicht beispielhaft den Aufbau eines dcif-Dokumentes.

Da diese Methode die einzige Möglichkeit darstellt, um größere Datenmengen effektiv zu importieren, wurde dieser Weg in dieser Arbeit weiterverfolgt.

### 3.2.2 RELcat

Für den Import der Relationen zwischen Datenkategorien genügt es, diese in einem trig-Dokument zu definieren und an den Systemadministrator von ISOcat zu schicken, der dieses ebenfalls validiert und einfügt. Folgendes trig-Dokument beschreibt das Set CMDI

```
@prefix relcat : <http://www.isocat.org/relcat/set/> .
@prefix rel   : <http://www.isocat.org/relcat/relations#> .
@prefix dc   : <http://purl.org/dc/elements/1.1/> .
@prefix isocat : <http://www.isocat.org/atacat/> .
relcat:CMDI
{
    isocat:DC-2573 rel:sameAs dc:identifier .
    isocat:DC-2482 rel:sameAs dc:language .
    isocat:DC-2484 rel:sameAs dc:language .
    isocat:DC-2545 rel:sameAs dc:title .
    isocat:DC-2512 rel:sameAs dc:creator .
    isocat:DC-2510 rel:sameAs dc:date .
    isocat:DC-2538 rel:sameAs dc:date .
    isocat:DC-2534 rel:sameAs dc:source .
    isocat:DC-2520 rel:sameAs dc:description .
    isocat:DC-2571 rel:sameAs dc:format .
    isocat:DC-2571 rel:distinct dc:type .
    isocat:DC-2522 rel:subClassOf dc:funder .
    isocat:DC-2542 rel:subClassOf dc:contributor .
    isocat:DC-2556 rel:subClassOf dc:contributor .
    isocat:DC-2502 rel:subClassOf dc:coverage .
}
```

Mit relcat:CMDI wird der Name des zu importierenden Sets festgelegt. Zum Zeitpunkt dieser Arbeit werden Sets mit gleichem Namen überschrieben: Menzo Windhouwer

via Email „It is the name which is used to retrieve the set, and if we reuse it the old set will be overwritten (...)I'm starting a new iteration of RELcat development, so this might all change in the future“<sup>41</sup>, dies kann also in absehbarer Zukunft noch überarbeitet werden.

Für diese Arbeit werden die Sets standardmäßig den Namen 'autoimport' erhalten.

Beim Import werden die PIDs überschrieben: „*What will be overwritten? (...) PIDs: just invent your own URI, e.g., my:DC-1, and use them to relate DCs (closed DC conceptual domain to simple DCs and simple DCs with an is-a relation to another simple DC); these PIDs will be overwritten by ISOcat PIDs (unless you have ISOcat acceptable PIDs, contact the ISOcat system administrator to find out)*“[7]. Weiter wird die Version auf 1:0 gesetzt, Registration status auf private und Creation date wird mit dem Datum des Imports ersetzt: „Version will become 1:0 Registration status will become private Creation date will become date of import“[7]

### 3.3 Umsetzung des Importes

Da die tripelbasierte Darstellung ein etabliertes Mittel zur Wissensdarstellung ist und strukturierte Daten in der Regel gut in dieses Format konvertiert werden können, wird diese als Ausgangspunkt gewählt.

Es gibt keine Möglichkeit, den Import großer Datenmenge automatisch durchzuführen, weshalb der Import darauf beschränkt ist, passende Importdateien zu erzeugen.

```
Foreach Tripel<Subjekt, Prädikat, Objekt>
  if(exists(Datenkategorie (Pid = Subjekt)))
    map(Datenkategorie, Prädikat, Objekt)
  else
    create Datenkategorie(Pid = Subjekt)
    map(Datenkategorie, Prädikat, Objekt)
```

map(Datenkategorie, Prädikat, Objekt) sucht dabei in der Datei mappings.properties nach dem ersten passenden dcif-Tag und gibt der Variable der übergebenen Datenkategorie den Wert des übergebenen Objektes.

Die unterstützten Datenformate richten nach der verwendeten Jena-API<sup>42</sup> und umfassen die üblichen Formate der tripelbasierten Darstellung.

---

41 Siehe references/email/Re autoimport.trig.eml

42 <http://jena.apache.org/>

Ein Grundproblem liegt dabei darin, dass dies eine Abbildung einer offenen Welt auf eine geschlossene Welt ist, die sich nicht vollständig durchführen lässt. Es ist unmöglich, alle möglichen xml-Tags auf ihre passenden dcif-Tags abzubilden, da man dafür jedes (zukünftige) xml-Tag kennen müsste. Deshalb benötigt der Import eine Konfigurationsdatei namens `mappings.properties`, in der jede Abbildung folgendermaßen definiert wird:

```
[ausgeschriebenes xml-tag ohne http://] = [Name der dcif - Eigenschaft]
```

```
www.w3.org/2004/02/skos/core#definition = definition
```

Diese Tags dürfen keine Zeichen enthalten, die für JAVA-Properties eine Bedeutung haben ( insbesondere ':' ).

Anschließend kann die Importdatei folgendermaßen erzeugt werden:

```
//Konstruktor mit Angabe der Datenquelle
RDF2Dcif worker = new RDF2Dcif("http://www.lingvoj.org/lingvoj.rdf");
worker.init();

//Nachbauen des Datenkategoriesets wie in mappings.properties vorgegeben
worker.CreateDcs();

//Importdatei schreiben
worker.Writedcif();
```

Dieser Code erzeugt eine Datei `lingvoj.rdf.dcif`<sup>43</sup>, die in ISOcat importiert werden kann. Für Tests kann diese Datei auch schon für die Kandidatensuche mit herangezogen werden, was allerdings nur auf Testsystemen zu empfehlen ist.

Als PID für eine Datenkategorie wird dabei der letzte Teil der URI angenommen.

In ISOcat ist eine Definition ein obligatorischer Bestandteil einer Datenkategorie. Aus diesem Grund wird, sofern keine Definition gefunden werden konnte, ein leerer String als Definition festgelegt. „I still get some Schematron error messages (see below). I think they mostly are due to the lack of a definition, which is obligatory.“(Menzo Windhouwer via Email)<sup>44</sup>.

### 3.4 Beispielimport

Für den Beispielimport wird die owl-Darstellung der GOLD2010-Ontologie<sup>45</sup> mit

43 [results/importfiles/lingvoj.RDF.dcif](#)

44 [references/email/Re import.eml](#)

45 <http://linguistics-ontology.org/gold-2010.owl>

folgender mappings.properties-Datei verwendet, die auch standardmäßig beiliegt.

```
www.w3.org/2004/02/skos/core#prefLabel = dataElementName
www.w3.org/2004/02/skos/core#altLabel = name
www.w3.org/2000/01/RDF-schema#label = name
xmlns.com/foaf/0.1/name = name
www.w3.org/2004/02/skos/core#definition = definition
www.w3.org/2004/02/skos/core#example = example
www.w3.org/2000/01/RDF-schema#comment = explanation
www.w3.org/2004/02/skos/core#editorialNote = note
www.w3.org/2004/02/skos/core#note = note
www.w3.org/2004/02/skos/core#changeNote = changeDescription
www.w3.org/2002/07/owl#sameAs = sameAs
```

In ISOcat wird Englisch über die Abkürzung 'en' angegeben, aus Gründen der Kompatibilität mussten alle 'lang="eng"' durch 'lang="en"' ersetzt werden. Sonst wurde nichts an dem ursprünglichen owl-Dokument geändert. Die vollständige 1152 Ressourcen umfassende Datei befindet sich auf der CD<sup>46</sup>.

```
<owl:Class rdf:about="http://purl.org/linguistics/gold/Auxiliary">
<rdfs:label xml:lang="eng">Auxiliary</rdfs:label>
<rdfs:subClassOf rdf:resource="http://purl.org/linguistics/gold/Verbal"/>
<rdfs:isDefinedBy
  rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://purl.org/linguistics/gold
</rdfs:isDefinedBy>
<rdfs:comment xml:lang="en">
  Auxiliary is a term used to describe verbs which are used in coordination with another verb to express
  mood, tense or aspect of the action denoted by the main verb. Marginal auxiliaries share some of these
  properties, but not all. [Crystal 1985: 28; Pei
</rdfs:comment>
</owl:Class>
```

Nach Aufruf von

```
RDF2Dcif worker = new RDF2Dcif("file:///[/PATH2WORKINGDIR]/gold-2010.owl");
worker.init();
worker.createDcs();
worker.writedcif();
```

befindet sich im Arbeitsverzeichnis eine Datei gold-2010.owl.dcif<sup>47</sup>, in der die Ressourcen folgendermaßen beschrieben werden.

---

46 references/httpbackups/gold2010.owl  
47 results/importfiles/gold2010.owl.dcif

```

<dcif:dataCategory pid="my-Auxiliary" type="simple">
<dcif:administrationInformationSection>
  <dcif:administrationRecord>
    <dcif:identifier>Auxiliary</dcif:identifier>
    <dcif:version>1:0</dcif:version>
    <dcif:registrationStatus>private</dcif:registrationStatus>
    <dcif:origin>gold-2010.owl</dcif:origin>
    <dcif:creation><dcif:creationDate>2013-03-18</dcif:creationDate>
    <dcif:changeDescription xml:lang="en">Automatic Import
    </dcif:changeDescription>
    </dcif:creation>
  </dcif:administrationRecord>
</dcif:administrationInformationSection>
<dcif:descriptionSection>
  <dcif:profile>Private</dcif:profile>
  <dcif:dataElementNameSection>
    <dcif:dataElementName>Auxiliary</dcif:dataElementName>
    <dcif:source>[FILEPATH]/gold-2010.owl</dcif:source>
  </dcif:dataElementNameSection>
  <dcif:languageSection>
    <dcif:language>en</dcif:language>
    <dcif:nameSection><dcif:name
      xml:lang="en">Auxiliary</dcif:name>
      <dcif:nameStatus>admitted name</dcif:nameStatus>
    </dcif:nameSection>
    <dcif:definitionSection>
      <dcif:definition xml:lang="en"></dcif:definition>
      <dcif:source>[FILEPATH]/gold-2010.owl</dcif:source>
    </dcif:definitionSection>
    <dcif:explanationSection>
      <dcif:explanation xml:lang="en">Auxiliary is a term used
      to describe verbs which are used in coordination with
      another verb to express mood, tense or aspect of the
      action denoted by the main verb. Marginal auxiliaries
      share some of these properties, but not all. [Crystal
      1985: 28; Pei
      </dcif:explanation>
      <dcif:source>[FILEPATH]/gold-2010.owl</dcif:source>
    </dcif:explanationSection>
  </dcif:languageSection>
</dcif:descriptionSection>
</dcif:dataCategory>

```

## 4 Finden von Kandidaten für Relationen zwischen Datenkategorien

Die zweite Aufgabe dieser Aufgabe besteht darin, für eine Datenkategorie, die als Vorlage dient, mögliche Kandidaten für Relationen zu finden und, wenn möglich, Relationen vorzuschlagen.

### 4.1 Grundüberlegungen

Datenkategorien bestehen aus 3 unterschiedlichen Arten von Attributen, die diese beschreiben und jeweils unterschiedliche Methoden zum Vergleich ermöglichen:

Namen : <dcif:name> <dcif:dataElementName>  
Volltext: <dcif:definition> <dcif:explanation> (<dcif:justification>) <dcif:note>  
Instanzdaten, Beispiele : <dcif:example>

Ähnlichkeiten zwischen Namen können ein Hinweis sein auf Ähnlichkeiten der von diesen Namen bezeichneten Konzepte. Insbesondere Spezialisierungen und Generalisierungen in wissenschaftlichen Konzepthierarchien neigen dazu, in irgendeiner Art von Substring-Beziehung zueinander zu stehen, wie die Beispiele 'adjective' und 'comparative adjective'. Namen wie adjective und adjectival bezeichnen ähnliche Konzepte<sup>48</sup> und haben einen starken semantischen Zusammenhang.

Die Ähnlichkeit zweier Konzepte müsste sich auch in einer Ähnlichkeit der diese Konzepte beschreibenden Texte widerspiegeln. Ein Text, der um seine Stoppwörter bereinigt wird und dessen Begriffe jeweils auf ihren Wortstamm reduziert wurden, müsste eine „Begriffswolke“ für das beschriebene Konzept ergeben. Diese Begriffswolke würde Ähnlichkeiten mit anderen Begriffswolken aufzeigen, wenn diese eine Schnittmenge an Begriffen gemeinsam haben.

Beispiele für ein linguistisches Konzept können aus einer Grundmenge gewählt werden, die viel zu groß ist, als dass man sie abbilden könnte. Deshalb ist es unmöglich, auf der Ebene von Beispielen direkt zu suchen. Beispiele selbst haben allerdings wieder Eigenschaften, die sie miteinander vergleichbar machen, nämlich die Anzahl der Buchstaben, Leerzeichen, Ziffern und Sonderzeichen. Von dieser

<sup>48</sup> <http://dictionary.reference.com/browse/adjective>

Abstraktionsebene aus betrachtet spielt es keine Rolle, ob als Beispiel für ein Konzept der 12.02.1985 oder der 10.10.2005 ausgewählt wurde. Beide Beispiele bestehen aus 8 Ziffern und 2 Sonderzeichen, was die Konzeptmenge schon auf eine Datumsart, eine Versionsnummer oder Ähnliches einschränken würde.

## 4.2 Vergleichsmethoden

### 4.2.1 Namenssuche

Diese Methode sucht Kandidaten für Relationen aufgrund von Ähnlichkeiten im Namen. Dadurch sollen einerseits Kandidaten gefunden werden, die unter ähnlichem Namen bereits beschrieben sind – beispielsweise 'adjective' und 'adj' – und andererseits ähnliche Konzepte, die durch ihre Namen weiter differenziert werden – beispielsweise adjective und comparative adjective.

#### **a** *Datenaufbereitung*

Die für diese Methode relevante Tabelle – pid\_name\_dcs wird während des Datenbankupdates mit den zusammengehörigen PIDs, Namen und dem Datenkategorieset gefüllt. Namen, die länger als 49 Zeichen sind, werden abgekürzt<sup>49</sup>. Diese Informationen werden nicht weiter bearbeitet.

#### **b** *Kandidatensuche*

Die Kandidatensuche startet man mit der Methode findNameCandidates von DCComparator. Zu Beginn wird der Name der Vorlage in seine durch Sonderzeichen getrennten Bestandteile zerlegt.

„indefinite article (sg,f)“ → indefinite,article,sg,f

Weiter werden für die Abkürzungen 'sg', 'pl', 'f', 'm', 1, 2, 3, 'pres', 'fut' und 'fin' die Begriffe 'singular', 'plural', 'feminine', 'masculine', 'first', 'second', 'third', 'future' und 'finite' als Bestandteil mit aufgenommen und umgekehrt die jeweiligen Abkürzungen für die Begriffe.

indefinite,article,sg,f → indefinite,article,sg,f,singular,female

Diese Ersetzungsregeln basieren auf den Namen der Datenkategorien, die nie als

---

<sup>49</sup> DC-2711 von nkjp enthält die Beschreibung als Namen.

Kandidaten gefunden wurden, aber in RELcat mit Datenkategorien verknüpft sind, und offensichtliche Abkürzungen in ihren Namen haben, wie beispielsweise DC-3562 namens 'v(fin,pres,1,sg)'.

Anschließend wird mittels LIKE-Operator nach passenden Einträgen für jeden Bestandteil des Namens gesucht. Dieser Operator ermöglicht die Suche nach Teilstrings, indem er eine Wildcard anbietet, die im Ergebnis durch beliebige Zeichenketten ersetzt wird. Es wird also nach allen Datenkategorien gesucht, die in ihrem Namen einen der Bestandteile des Namens der Vorlage enthalten.

Dies wird für die Porter-reduzierten Bestandteile wiederholt<sup>50</sup>.

Hier entstand ein Problem durch die Verwendung von Abkürzungen wie 'f' oder 'pl'. Da diese sowohl beim Kandidaten als auch bei der Vorlage verwendet werden können, ist es sinnvoll, die Begriffe und deren Abkürzungen sich gegenseitig ergänzen zu lassen. Für Abkürzungen wie 'f' sorgen dabei allerdings für starken Noise<sup>51</sup>, da damit jedes Wort mit einem 'f' als Kandidat in Frage kommt.

Deshalb werden Namensbestandteile nur für die Suche verwendet, wenn diese länger als 1 Zeichen sind. Damit wird immer noch Noise erzeugt – 'pl' würde 'plastic' als Kandidat liefern. Dieser sollte aber im erträglichen Maß liegen und dem Nutzen stark unterliegen.

Betroffen von dieser Regelung sind 152 Datenkategorien, wobei bei 83 ausschließlich Buchstaben als einzelne Zeichen vorkommen und 46 Datenkategorien nicht für die Namenssuche in Betracht kommen weil ihr Name ausschließlich aus einzelnen Zeichen besteht, die auch nicht durch andere Begriffe ergänzt werden. Eine vollständige Auflistungen der jeweiligen Namen befinden sich auf der CD<sup>52</sup>

Diese Suche ist nicht bijektiv, wenn also eine Datenkategorie x als Kandidat für eine Datenkategorie y in Betracht kommt, gilt dies nicht umgekehrt. 'Pronoun' ist ein Kandidat für 'noun', aber 'noun' ist kein Kandidat für 'pronoun'

---

50 Der erste Schritt mag redundant erscheinen, aber einige Regeln des Porter-Algorithmus reduzieren nicht, sondern ersetzen durch einen anderen Suffix (ies → y), sodass es hier zu einem Informationsverlust kommen kann, wenn nur nach dem reduzierten Wort gesucht wird.

51 Die Datei results/candidates/nameSim/singular.f.txt enthält die 610 Einträge zählenden Kandidatenmenge einer Datenkategorie 'singular, f', die dieses Problem veranschaulichen sollte.

52 results/tests/manuell/single\_letters\*.txt

### **c** *Evaluationsmaß*

David Pansch beschreibt in seiner Bachelorarbeit[13] eine Formel zur Berechnung der Namensähnlichkeit genormt auf die Länge. Diese lässt sich auch für diese Aufgabe anwenden und wurde ohne Änderungen übernommen.

Dieses Maß kann im Gegensatz zu denen der anderen Suchmethoden auch genutzt werden, um deren Ergebnisse zu bewerten.

*name* = Name der Datenkategorie

*v* = Vorlage    *k* = Kandidat

$$\text{Sim}(v,k) = 1 - \frac{\text{levenshtein}(name_v, name_k)}{\max(\text{length}(name_v), \text{length}(name_k))} \quad (1)$$

#### Beispielrechnung

*v* = adjective    *k*<sub>1</sub> = adjective    *k*<sub>2</sub> = adjectival    *k*<sub>3</sub> = noun    *k*<sub>4</sub> = comparative adjective

$\text{length}(name_v) = 9$      $\text{length}(name_{k_1}) = 9$      $\text{length}(name_{k_2}) = 10$      $\text{length}(name_{k_3}) = 4$

$\text{length}(name_{k_4}) = 21$

$$\text{Sim}(v, k_1) = 1 - \frac{0}{9} = 1$$

$$\text{Sim}(v, k_2) = 1 - \frac{2}{10} = 0,8$$

$$\text{Sim}(v, k_3) = 1 - \frac{9}{9} = 0$$

$$\text{Sim}(v, k_4) = 1 - \frac{12}{21} = 0.42857142857$$

### **d** *Weitere Ansätze*

Kandidat *k*<sub>4</sub> zeigt eine Substring-Beziehung zwischen Kandidat und Vorlage, die einen Informationsgehalt hat. Diese findet keinen expliziten Einfluß im Evaluationsmaß und könnte für eine Erweiterung genutzt werden, da sie auf einen starken Zusammenhang hindeutet.

#### **4.2.2 Volltextsuche**

Datenkategorien können in folgenden Attributen per Volltext näher beschrieben

werden: <dcif:definition>, <dcif:explanation>, <dcif:note>, <dcif:justification>.

<dcif:justification> beschreibt dabei allerdings nicht die Datenkategorie selbst, sondern begründet, warum diese in ISOcat enthalten sein sollte.

Da es für diese Aufgabe keine Rolle spielt, ob ein Begriff in der Definition oder in der Erklärung gefunden wurde, werden alle diese Texte zu einer Volltextbeschreibung zusammengefügt.

### **a Datenaufbereitung**

Während des Datenbankupdates wird für jedes Datenkategorieset eine Tabelle mit Namen des aktuell bearbeiteten Sets in der Datenbank angelegt. Für jede enthaltene Datenkategorie werden die PID und die aus den Volltexten extrahierten und per Porter-Algorithmus reduzierten Begriffe gespeichert.

Um die Wortmenge auf für das beschriebene Konzept relevante Begriffe einzuschränken, wurde die Terminologie-Extraktion der ASV-Toolbox verwendet. Die Toolbox bietet einige Parameter an, mit denen die Terminologie-Extraktion angepasst werden kann und arbeitet mit einem Sprachkorpus, der statistische Informationen für Wörter der jeweiligen Sprache enthält.

Da die Sprache sich hier sehr gut auf für Sprachwissenschaftler relevante Themen eingrenzen lässt, ist es sinnvoll, das Korpus entsprechend anzupassen. Um dies umzusetzen, wurden alle Datenkategoriesets der ISOcat eingelesen und als mit der Terminologieextraktion kompatiblen Korpusdateien abgespeichert. Diese Dateien liegen dieser Arbeit bei<sup>53</sup> und müssen wie in der Dokumentation der ASV Toolbox beschrieben als eigenes Sprachkorpus für die Terminologie-Extraktion verfügbar gemacht werden<sup>54</sup>. Der Quellcode zum Erzeugen des ISOcatkorpus liegt als jar-Paket und als Quellcode dieser Arbeit bei<sup>55</sup>.

Eine höher gewählte **Mindestfrequenz innerhalb des Korpus** führt zur Extraktion allgemeinsprachlicherer – also öfter im gemischten Sprachgebrauch auftretenden – Begriffe. Da hier fachspezifische Begriffe gesucht sind, ist es sinnvoll, diesen Wert so gering wie möglich – also 1 – zu wählen.

Für die **Mindestfrequenz im Text** wurde ebenfalls 1 gewählt, da es sich bei

---

53 data/isocatcorpora

54 Installationsanleitung Siehe Anhang A

55 Implementations/ISOcat2Wordlist

Definitionen oder Beschreibungen von Konzepten in der Regel um kurze Texte handelt, bei denen es zu erwarten ist, dass Begriffe innerhalb der Beschreibung nicht oft wiederholt auftreten.

Während Wörter mit Frequenz 1 in der Praxis oft auf Neologismen oder Rechtschreib – bzw. Kodierungsfehler zurückzuführen sind, handelt es sich bei dem hier verwendeten Daten um Texte, die von einer registrierten Nutzergruppe mit umfangreichem Fachwissen erstellt und kontrolliert werden. Deshalb kann man davon ausgehen, dass Auftreten von niederfrequenten Begriffen hier nicht durch solche Einflüsse verursacht werden.

Das verwendete **Signifikanzmaß** und die **Mindestsignifikanz** wurden mit dem in Kapitel 6.1 beschriebenen Terminologie-Extraktions-Tests ermittelt : Likelihood mit Mindestsignifikanz 30.

Im folgenden wird der Ablauf der Datenaufbereitung am Beispiel der in Gold-2010 befindlichen Datenkategorie DC-3055 veranschaulicht.

#### Volltextbeschreibung

The GOLD ontology was developed as part of the E-MELD project funded by the NSF (BCS0729644 and BCS0094934). The concepts were reviewed by 50 noted typologists and documentary linguists at the E-MELD 2005 workshop (<http://emeld.org/workshop/2005/proceeding.html>) at Harvard University. They have since been refined by the GOLD Community (<http://linguistics-ontology.org>).

An adjectival, or 'adjective', is a part of speech whose members modify nouns. An adjectival specifies the attributes of a noun referent. Note: this is one case among many. Adjectivals are a class of modifiers. An adjectival may be inflected as comparative or superlative [Crystal 1997: 8; Payne 1997: 63].

This concept is part of the General Ontology for Linguistic Description (GOLD). It is a child concept of <http://purl.org/linguistics/gold/Predicator>.

For other relationships among the concepts see: <http://linguistics-ontology.org/gold>.

To make suggestions with regard to the entire ontology or individual concepts, please visit the GOLD Community website at <http://linguistics-ontology.org>.

#### Extrahierte Begriffe

adjectival, adjectivals, bcs0094934, bcs0729644, community, crystal, funded, gold, harvard, meld, ontology, org, part, proceeding, reviewed, workshop

#### Auf Porter-Stamm reduzierte Begriffe

adjectiv, adjectiv, bcs0094934, bcs0729644, commun, crystal, fund, gold, harvard, meld, ontolog, org, part, proceed, review, workshop

Es fällt auf, dass die extrahierten Begriffe nicht zwangsläufig das Konzept selbst näher beschreiben, sondern auch das Umfeld der Datenkategorie. Es wird sich in der Auswertung dieser Arbeit zeigen, dass diese Methode trotzdem sinnvolle Ergebnisse liefert. Dabei sollte man aber daran denken, dass Datenkategorien bei dieser Suche potentiell Kandidaten mit gemeinsamen Quellen favorisieren, da diese (in diesem Fall 'gold' oft als signifikanter Term erkannt und darüber Übereinstimmungen gefunden werden.

Es ist wichtig, dass für die Datenaufbereitung und den Vergleich die gleichen Methoden verwendet werden. Deshalb ist es erforderlich, bei einer Änderung der Terminologie-Extraktion oder deren Parameter beziehungsweise einer Änderung der Methode der Grundformreduktion das Datenbankupdate zu wiederholen.

In der Datei blacklist.txt können Begriffe aus der Bearbeitung ausgeschlossen werden. Standardmäßig enthält diese das Wort 'contain'.

Es ist nicht ungewöhnlich, dass die Datenaufbereitung mehrere Stunden in Anspruch nimmt. Mit den Daten, die im ISOcat-Backup hinterlegt sind dauert das Update inklusive des Updates der Beispieldaten auf einem Laptop HP-625 mit 4GB Arbeitsspeicher, AMD V160 Prozessor (2,40GHz) unter Windows 7 64Bit von 22:05:02 bis 02:05:06 des Folgetages.<sup>56</sup> Insbesondere die Terminologie-Extraktion und das Sammeln der Daten von ISOcat sind sehr zeitaufwändig.

## **b**      **Vergleich**

Analog zur Datenaufbereitung werden die reduzierten Begriffe der Volltextbeschreibung der Vorlage extrahiert. Anschließend wird in der Datenbank nach Kandidaten gesucht, wobei Datenkategorien als Kandidat gelten, wenn sie mindestens einen Begriff mit der Vorlage gemeinsam haben.

## **c**      **Evaluationsmaß**

Für die Berechnung der Termüberdeckung zweier Datenkategorien eignet sich der Dice-Koeffizient mit

*# term = Anzahl der Begriffe*

*v = Vorlage      k = Kandidat*

---

<sup>56</sup> Siehe CD results/tests/logs/Log.txt

$$\text{Sim}(v, k) = \frac{(2 * \# \text{term}_{(v \wedge k)})}{(\# \text{term}_v + \# \text{term}_k)} \quad (2)$$

#### **d** *Weitere Ansätze*

Die Begriffsmenge für eine Datenkategorie kann erweitert werden, indem zusätzlich statistische Kookkurrenzen der Begriffe beachtet werden. WordNet® bietet ebenfalls Möglichkeiten zur Erweiterung der Begriffsmenge (beispielsweise können Synonyme mit in die Begriffsmenge aufgenommen werden). Da die Ergebnisse dieser Vergleichsmethode aber schon ohne diese Erweiterungen sehr ungenau sind und eine Verbesserung eher dahingehend sinnvoll wäre, die Präzision zu erhöhen – also weniger, aber dafür stärker differenzierende Begriffe zu extrahieren um die Überschneidung der Begriffsmenge aussagekräftiger zu machen – wurde dies im Rahmen dieser Arbeit nicht weiter verfolgt.

### **4.2.3 Beispiel-Suche**

Um die Datenkategorien auf Ebene ihrer Beispiele zu vergleichen, ist es nötig, ein möglichst generisches Verfahren zu verwenden, welches die Beispiele soweit abstrahiert, dass es nicht mehr auf das Beispiel selbst, sondern auf dessen Eigenschaften ankommt.

David Pansch beschreibt in seiner Bachelorarbeit[13] ein Verfahren, bei dem XPath-Ausdrücke verglichen werden, indem der Anteil der Zahlen und Buchstaben miteinander verglichen wird. Dieser Ansatz wurde für diese Aufgabe aufgegriffen und um Leer- und Sonderzeichen erweitert.

#### **a** *Datenaufbereitung*

Während des Updates der Datenbank wird die Tabelle pid\_example mit den für die jeweilige PID hinterlegten Beispielen gefüllt. Nachdem das Update abgeschlossen wurde, werden mittels ExampleUpdateHandler→updateExamples() in der Tabelle pid\_example\_stats für jede PID die durchschnittliche Anzahl der in einem Beispiel enthaltenen Buchstaben, Zahlen, Leerzeichen und Sonderzeichen gespeichert.

#### **b** *Vergleich*

Aus den vorberechneten Daten wird für jede Datenkategorie der Anteil jedes Zeichentyps an den Gesamtzeichen berechnet. Dies wird ebenfalls für die

Datenkategorie, die als Vorlage dienen soll, durchgeführt.

Eine Datenkategorie zählt nun als Kandidat, wenn mit jedem Zeichenanteil in einem Bereich liegt, der durch den jeweiligen Zeichenanteil der Vorlage  $\pm$  eines Toleranzintervalls liegt. Dieses Intervall gibt dann die Genauigkeit an, mit der die durchschnittlichen Zeichenanteile übereinstimmen und wird als Evaluationsmaß verwendet.

Um ähnlich des Volltextvergleiches die optimale Konfiguration für den Vergleich zu bestimmen, müsste die optimale Ergebnismenge anhand der in RELcat enthaltenen Informationen berechnet werden. Dafür benötigt man die Schnittmenge zwischen den Datenkategorien, bei denen Beispiele angegeben wurden und den Datenkategorien, die in RELcat vorkommen. Diese Schnittmenge<sup>57</sup> umfasst folgende 8 Datenkategorien:

DC-221  
DC-3720  
DC-3727  
DC-3051  
DC-3581  
DC-3580  
DC-4648  
DC-216

Dies ist für eine Bestimmung des optimalen Wertes zu wenig, weshalb die Bestimmung des optimalen Toleranzwertes für einen späteren Zeitpunkt empfehlenswert ist. Für diese Arbeit wird die maximale Varianz der Beispiele willkürlich auf 1% gesetzt. Damit ist gewährleistet, dass das Verfahren greift und zukünftige Auswertungen mit einem normierten Basiswert – 1% – verglichen werden können.

### **c**      **Evaluationsmaß**

$$\begin{aligned} letter &= \frac{\text{count}(\text{Buchst})}{\text{count}(\text{alle Zeichen})} & num &= \frac{\text{count}(\text{Ziffern})}{\text{count}(\text{alle Zeichen})} \\ space &= \frac{\text{count}(\text{Leerzeichen})}{\text{count}(\text{alle Zeichen})} & spec &= \frac{\text{count}(\text{Sonderzeichen})}{\text{count}(\text{alle Zeichen})} \end{aligned}$$

$v = \text{Vorlage}$      $k = \text{Kandidat}$

$$\text{Sim}(v,k) = 1 - \max(|letter_v - letter_k|, |num_v - num_k|, |space_v - space_k|, |spec_v - spec_k|) \quad (3)$$

---

57 Quellcode: implementations/snippets/dc\_with\_example\_in\_RELcat.txt

Beispielrechnung für ein Datum, eine Versionsnummer und einen Satz:

$$v = 12.02.1985$$

$$k_1 = 22.10.1985 \quad k_2 = v.1002.8 \quad k_3 = \text{Ein Männlein steht im Walde.}$$

$$letter_v = 0 \quad num_v = \frac{8}{10} \quad space_v = 0 \quad spec_v = \frac{2}{10}$$

$$letter_{k_1} = 0 \quad num_{k_1} = \frac{8}{10} \quad space_{k_1} = 0 \quad spec_{k_1} = \frac{2}{10}$$

$$letter_{k_2} = \frac{1}{8} \quad num_{k_2} = \frac{5}{8} \quad space_{k_2} = 0 \quad spec_{k_2} = \frac{2}{8}$$

$$letter_{k_3} = \frac{20}{25} \quad num_{k_3} = 0 \quad space_{k_3} = \frac{4}{25} \quad spec_{k_3} = \frac{1}{25}$$

$$\text{Sim}(v, k_1) = 1 - \max(0, 0, 0, 0) = 1$$

$$\text{Sim}(v, k_2) = 1 - \max\left(\frac{1}{8}, \frac{3}{8}, 0, 0\right) = 0.625$$

$$\text{Sim}(v, k_3) = 1 - \max\left(\frac{20}{25}, \frac{8}{10}, \frac{4}{25}, \frac{4}{25}\right) = 0.2$$

#### **d**      **Weitere Ansätze**

Ein großer Fundus an Beispielen lässt sich in den Instanzdaten finden, die in CMDI-Metadaten gespeichert sind. In diesen ist gespeichert, welche Textfragmente mit welchen Datenkategorien kategorisiert werden, also welche Textfragmente beispielsweise als Datumsangabe klassifiziert wurden.

Diese Informationen können zurückverfolgt werden und damit potentiell eine große Anzahl von Beispielen für jede Datenkategorie bereitstellen.

#### **4.2.4 WordNet**

WordNet<sup>58</sup> ist eine von George A. Miller 1980 begonnene Datenbank, die englische Substantive, Verben, Adjektive und Adverbien in kognitiven Synonymen (Synsets) zusammenfasst. „WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-

---

58 <http://WordNet.princeton.edu/>

semantic and lexical relations.“[14]. Diese Synsets sind durch semantische und lexikalische Relationen miteinander verknüpft, wodurch ein Begriffsnetzwerk – oder Wortnetz – entsteht, welches sich für diese Aufgabe als hilfreich erweist.

Eine Besonderheit liegt hier in der Qualität der Daten, die WordNet® bereitstellt und den angegebenen Relationen. Diese wurden bereits von einer Person manuell bestätigt, was es ermöglicht, nicht nur Kandidaten für Relationen zu suchen, sondern zusätzlich verlässliche Vorschläge für die zugehörige Relation selbst anzugeben.

### **a Datenaufbereitung**

WordNet® arbeitet auf einem eigenem Datensatz. Dieser liegt bei, wenn man den WordNet®-Browser installiert und der Speicherort muss in der Konfigurationsdatei angegeben werden. Eine Kopie des Ordners befindet sich auf der beiliegenden CD<sup>59</sup>.

Als Kommunikationsschnittstelle für die Implementierung dient die Klasse WordNetAccess, die für die jeweiligen Relationen in RELcat passenden Daten liefert. Damit ist gewährleistet, dass, sofern Änderungen nötig sind, diese unkompliziert und unabhängig von der restlichen Implementierung durchgeführt werden können.

### **b Vergleich**

In ersten Durchlauf wird für den Namen der Datenkategorie, die als Vorlage dient, nach Objekt-Einträgen im Datenbestand von WordNet® gesucht. Werden Objekt-Einträge gefunden, deren Relation auf eine der RELcat-Relationen passen, wird in der Datenbank nach Datenkategorien gesucht, deren Namen entweder auf den Objekt-Eintrag selbst oder dessen Porter-Stamm passen. Namen, die aus mehreren Teilen bestehen, wie 'comparative adjective', werden anfangs – im Gegensatz zur Namenssuche – als Ganzes betrachtet. Je nachdem, über welche Relation der Kandidat in WordNet® gefunden wurde, wird für diesen eine Relation vorgeschlagen.

Folgende Relationen aus Wordnet werden auf die entsprechenden RELcat-Relationen abgebildet:

<b>RELcat</b>	<b>WordNet®</b>
rel:sameAs	Synonymy
rel:superClass	Instance-Hypernymy Hypernymy

---

<sup>59</sup> data/WordNet/dict

<b>RELCat</b>	<b>WordNet®</b>
rel:subClass	Instance-Hyponymy Hyponymy
rel:hasPart	Member - Holonymy
rel:partOf	Member - Meronymy
rel:hasDirectPart	Substance-Holonymy Part - Holonymy
rel:directPartOf	Substance-Meronymy Part - Meronymy

Dabei gilt immer :

Subjekt – Prädikat – Objekt entspricht Vorlage – Relation – Kandidat

Wurden keine Kandidaten gefunden, wird die Suche für die durch Leerzeichen getrennten Bestandteile des Namens durchgeführt. Für die in diesem zweiten Durchlauf gefundenen Kandidaten wird keine Relation vorgeschlagen.

### **c** *Evaluationsmaß*

Jedes Ergebnis, welches von dieser Methode zurückgeliefert wird, basiert auf Fachwissen, welches vorher manuell in WordNet® festgehalten wurde. Kommt dabei ein falsches Ergebnis zustande, liegt es nicht an der Suchmethode, sondern daran, dass der Zusammenhang in diesem Fall nicht stimmt.

Beispielsweise könnte ein mehrdeutiges Wort x in einem nicht-linguistischen Kontext mit einem anderen Wort y verbunden sein, welches zwar ebenfalls im linguistischen Kontext existiert, dort aber nicht mit x verbunden ist. Dieser Kontextbezug und die große Menge an Fachwissen, die in die Verknüpfung zweier Konzepte in WordNet® einfließt, macht eine algorithmische Bewertung der Richtigkeit dieses Kandidaten mit Bezug zur hier beschriebenen Methode wenig sinnvoll.

Namensähnlichkeit, Termüberdeckung (ggf.) und die Ähnlichkeit der Beispiele (ggf.) können genutzt werden um die Ergebnisse zu bewerten, haben aber keinen Bezug zur eigentlichen Suchmethode.

### **d** *Weitere Ansätze*

Außer WordNet® gibt es noch andere Wissensdatenbanken, die wertvolle Informationen mit Bezug zu dieser Arbeit bereitstellen können, wie beispielsweise der

UNESCO-Thesaurus<sup>60</sup>,

Diese hätten in der Regel – wie auch WordNet® – einen Vorteil gegenüber den generischen Verfahren, da sie manuell erstellt wurden und deshalb nahezu sicheres Wissen enthalten.

---

60 <http://databases.unesco.org/thesaurus/>

## 5 Online-Dienste

### 5.1 Webservices

Da die Installation und vor allem die Vorbereitung der Datenbank mit einem nicht zu unterschätzendem Aufwand verbunden ist, wurden, basierend auf dem Spring-Framework<sup>61</sup>, Webservices erstellt, die in Form einer war-Datei und Quellcode dieser Arbeit beiliegen und zum Zeitpunkt der Abgabe auf dem Server 139.18.2.147:8080/clarin/ fertig benutzbar installiert sind. Die Quellcodedateien können gleichzeitig als Code-Beispiele angesehen werden.

Eine Installationsbeschreibung für diese Services befindet sich ist in Anhang A.

Aufruf (Link)	Beschreibung
<a href="#">describedc/?pid=[pid]</a>	Liefert eine Volltextbeschreibung der Datenkategorie, die durch [pid] definiert ist
<a href="#">namedc/?pid=[pid]</a>	Liefert den Namen der Datenkategorie, die durch [pid] definiert ist
<a href="#">listdc/?dcs=[dcs]</a>	Listet alle vorhandenen Datenkategorien im Datankategorieset [dcs] auf.
<a href="#">listdcs/</a>	Listet alle vorhandenen Datenkategoriesets auf
<a href="#">termsofdc/?pid=[pid]</a>	Listet alle Porter-reduzierten Begriffe der Datenkategorie, die durch [pid] definiert ist
<a href="#">namesim/?pid=[pid]&amp;name=[name]</a> <a href="#">namesim/?pid1=[pid1]&amp;pid2=[pid2]</a> <a href="#">namesim/?name1=[name1]&amp;name2=[name2]</a>	Liefert die in 4.2.1c definierte Namensähnlichkeit
<a href="#">termdice/?pid1=[pid1]&amp;pid2=[pid2]</a> <a href="#">termdice/?pid=[pid]&amp;termlist=[termlist]</a> <a href="#">termdice/?</a> <a href="#">termlist1=[termlist1]&amp;termlist2=[termlist2]</a>	Liefert die in 4.2.2c definierte Termüberdeckung. termlist enthält mit '(-M-)' getrennten Porter-reduziert Begriffe und kann auch per POST-Request gesendet werden. Wenn keine termlist angegeben wird, wird diese aus der Datenbank geholt.
<a href="#">dc_comparator/?name=[name]</a> <a href="#">dc_comparator/?</a> <a href="#">WordNetname=[WordNetname]</a> <a href="#">dc_comparator/?examples=[examples]</a> <a href="#">dc_comparator/?fulltext=[fulltext]</a>	Liefert die Kandidaten für die über Variablenname und Variablenwert definierte Kandidatensuche. Es kann nur eine Variable pro Request angegeben werden. fulltext enthält die unbearbeitete Volltextbeschreibung und examples die durch

61 [www.springsource.org/spring-framework](http://www.springsource.org/spring-framework)

Aufruf (Link)	Beschreibung
	'(-M-)' Beispiele. Beide können auch per POST-Request angegeben werden. Zusätzlich zu examples kann tolerance mit angegeben werden, was dem in 4.2.3c beschriebenen Evaluationsmaß entspricht und die Ergebnismenge einschränkt.
<a href="relcat/?pid=[pid]">relcat/?pid=[pid]</a>	Liefert alle in RELcat gefundenen Einträge der Datenkategorie, die durch [pid] definiert ist. Einträge bestehen dabei aus Relation und Objekt.

## 5.2 Graphische Nutzeroberfläche

Die für die Demonstration der einzelnen Funktionen auf Basis des Vaadin-Frameworks erstellte Weboberfläche ist zum Zeitpunkt der Abgabe über die URL 139.18.2.147:8080/RELcatFinder\_gui zugänglich. Obwohl diese Weboberfläche bereits eingeschränkt produktiv einsetzbar ist, soll sie nur der Illustration der Funktionen dienen und aufzeigen, wie diese – beziehungsweise die Webservices – zusammen eingesetzt werden können, um eine Arbeitsfläche bereitzustellen.

### 5.2.1 Kandidatensuche

Hierbei handelt es sich um den Bereich der Weboberfläche, der bereits produktiv nutzbar ist. Über das Eingabefeld in Abbildung 4 kann eine beliebige Kandidatenvorlage erstellt werden.



The image shows a web form for searching candidates. It contains the following elements from top to bottom:
 

- A text input field labeled "PID".
- A text input field labeled "Name".
- A slider control labeled "Exampletolerance" with a small input box and a minus sign button below it.
- A plus sign button below the slider.
- A text input field labeled "Description".
- A "Search" button at the bottom left.

Abbildung 4: Eingabefeld

Mit den Buttons '+' und '-' lässt sich die Beispielmenge erweitern oder ein Beispiel entfernen. Über den Schieberegler lässt sich die für die Beispiel-Suche zu verwendende Toleranz festlegen. Hierbei gilt, dass bei 0 die Beispiele der Kandidaten genau der Vorgabe entsprechen müssen und bei 1 alle Beispiele als passend gelten.

Je nachdem, welche Informationen ausgefüllt wurden, werden mit einem Klick auf 'Search' die jeweiligen Suchmethoden gestartet. Diese ergänzen sich. Wenn also eine Vorlage namens 'Noun' mit den Beispielen '12.02.1985' und '31.03.2010' erstellt wird, werden sowohl Datenkategorien, die bei der Namenssuche für 'Noun' gefunden wurden, als auch Datenkategorien, die bei der Beispiel-Suche gefunden wurden, in die Ergebnismenge mit aufgenommen.

Abbildung 5 zeigt die Datenkategorieauswahl, die alle öffentlichen Datenkategoriesets mit ihren enthaltenen Datenkategorien enthält.

DATACATEGORYSELECTION	DATACATEGORY	NAME
▶ 2012020119011033_dcif_import		
▶ 2012091914384360_dcif_import		
▶ 2012091917330138_dcif_import		
▶ 2012092009335901_dcif_import		
▶ abbreviations_test1		
▶ asr		
▶ ataframework		
▶ athenscorediscussion2		
▶ audio		
▶ bas_metadata_120131		
▶ basics		

Import from rdf

Abbildung 5: Datenkategorieauswahl

Die Datenkategorien werden dabei sichtbar, wenn die jeweiligen Sets expandiert werden. Nachdem ein Set expandiert wurde, werden alle Datenkategorien mit Namen und PID aufgelistet.

DATACATEGORYSELECTION	DATACATEGORY	NAME
▶ cases		
▶ ckcc_dans		
▶ edisyn		
▶ formrelated		
▼ gold_2010		
	DC-3055	abessive case
	DC-3056	abilitative mode
	DC-3057	ablative case
	DC-3058	absolute antiq
	DC-3059	absolute case
	DC-3060	abstract

Aufgelistete Datenkategorien lassen sich mit einem Klick auf die jeweilige Tabellenspalte sortieren und per Rechtsklick → 'Choose' werden die Informationen, die zu dieser Datenkategorie bekannt sind, in das Eingabeformular eingefügt.

PID

Name

Exampletolerance  
 -

+

Description

**Search**

*Abbildung 7: DC-3055 ausgewählt*

Die Beschreibungen werden in der Datenbank gesucht. Dies bedeutet, dass diese nur für bei dem letzten Update bekannten Datenkategorien eingefügt werden, nicht bei neu importierten.

Nach einem Klick auf 'Search' werden in der Ergebnistabelle nun die Ergebnisse der Namensähnlichkeit zwischen 'abessive case' und den jeweiligen Kandidaten und die Termüberdeckungen zwischen den zugehörigen Beschreibungen angezeigt.

APPROVED	RELATION	NAME	DATA CATEG	NAME SIMILIARIT	DICE (TERMS)
<input type="checkbox"/>	rel:sameAs	abessive case	DC-3055	1.0	1.0
<input type="checkbox"/>	rel:related	distributive aspect	DC-3169	0.31578947368	0.92307692
<input type="checkbox"/>	rel:related	ablative case	DC-3057	0.76923076923	0.88888888
<input type="checkbox"/>	rel:related	orthographic syster	DC-3365	0.15789473684	0.88888888
<input type="checkbox"/>	rel:related	clause	DC-3114	0.15384615384	0.88888888
<input type="checkbox"/>	rel:related	verbal adjective	DC-3545	0.1875	0.86666666
<input type="checkbox"/>	rel:related	lative case	DC-3293	0.61538461538	0.85714285
<input type="checkbox"/>	rel:related	delative case	DC-3152	0.61538461538	0.85714285
<input type="checkbox"/>	rel:related	subordinate clause	DC-3496	0.444444444444	0.85714285
<input type="checkbox"/>	rel:related	auxiliary	DC-3095	0.23076923076	0.85714285
<input type="checkbox"/>	rel:related	enclitic	DC-3177	0.15384615384	0.85714285
<input type="checkbox"/>	rel:related	mora	DC-3323	0.07692307692	0.85714285
<input type="checkbox"/>	rel:related	superlative adjectiv	DC-3505	0.333333333333	0.82758620
<input type="checkbox"/>	rel:related	comparative adjecti	DC-3122	0.28571428571	0.82758620
<input type="checkbox"/>	rel:related	plain adjective	DC-3396	0.19999999999	0.82758620

Abbildung 8: Suchergebnis für DC-3055

Dieses Beispiel zeigt eine Sonderregelung, die für die Weboberfläche getroffen wurde: Wenn die Namensähnlichkeit zwischen Kandidat und Vorlage 1 ist, wird die Relation rel:sameAs vorgeschlagen.

'Approved' gibt an, ob dieser Kandidat mit dieser Relation verwendet werden soll oder nicht. Standardmäßig ist dies deaktiviert, wodurch sichergestellt ist, dass jeder gefundene Kandidat manuell bestätigt werden muss, bevor er für RELcat vorgeschlagen wird.

Über die Auswahlbox in 'Relation' kann die passende Relation ausgewählt werden. Standardmäßig werden hier die Relationen aktiviert, die von den Suchalgorithmen für

den Kandidaten empfohlen werden.

Wird die Maus über eine PID oder einen Namen bewegt, wird die Beschreibung der Datenkategorie angezeigt.

Die Tabellenspalten 'Namesimilarity' und 'Dice(Terms)' ermöglichen ein Sortieren der Ergebnismenge, wobei letztere nur bei Verwendung der Vollextanalyse wichtig ist.

Mit dem button 'Create Importfile' öffnet man ein 2. Fenster, welches die bestätigten Kandidaten im trig-Format beschreibt.

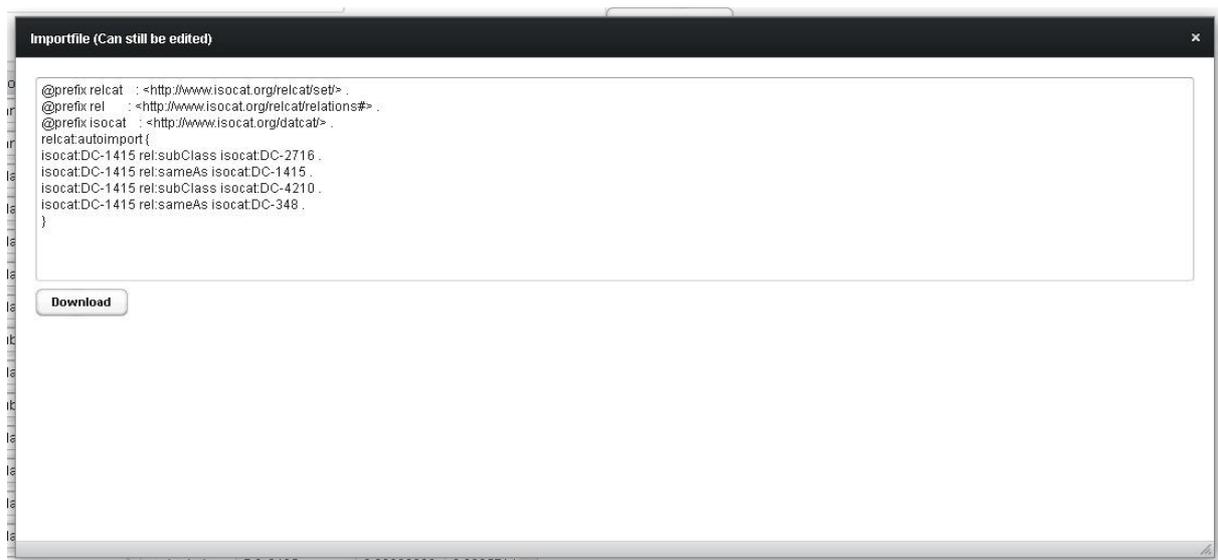


Abbildung 9: Importdatei für RELcat

Diese Datei kann hier noch editiert werden, um beispielsweise den Namen des Datenkategoriesets zu ändern<sup>62</sup>. 'Download' öffnet schließlich das Downloadfenster des Browsers für diese Datei. Die hier erzeugte Datei befindet sich auf der beiliegenden CD<sup>63</sup>.

### 5.2.2 Import von fremden Vokabularen

<sup>62</sup> Wird in RELcat ein Relationsset mit bereits enthaltenen Namen importiert, wird die vorherige Version überschrieben.

<sup>63</sup> results/importfiles/relcat/autoimport.trig

Unterhalb der Datenkategorieauswahl befindet sich ein Button 'Import from rdf'.

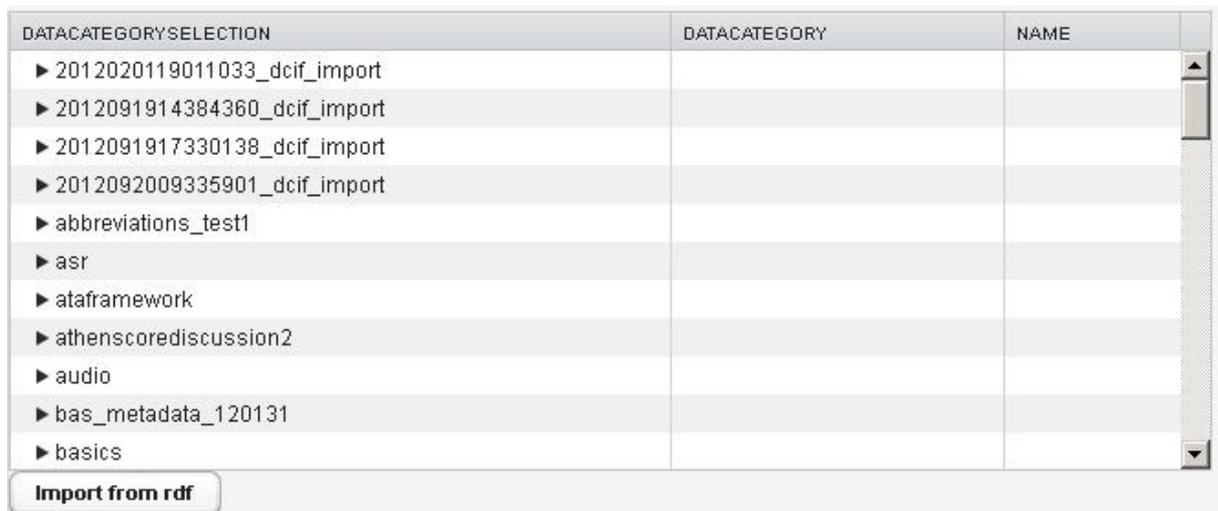


Abbildung 10: ImportButton

Dieser Button öffnet das Fenster in Abbildung 11

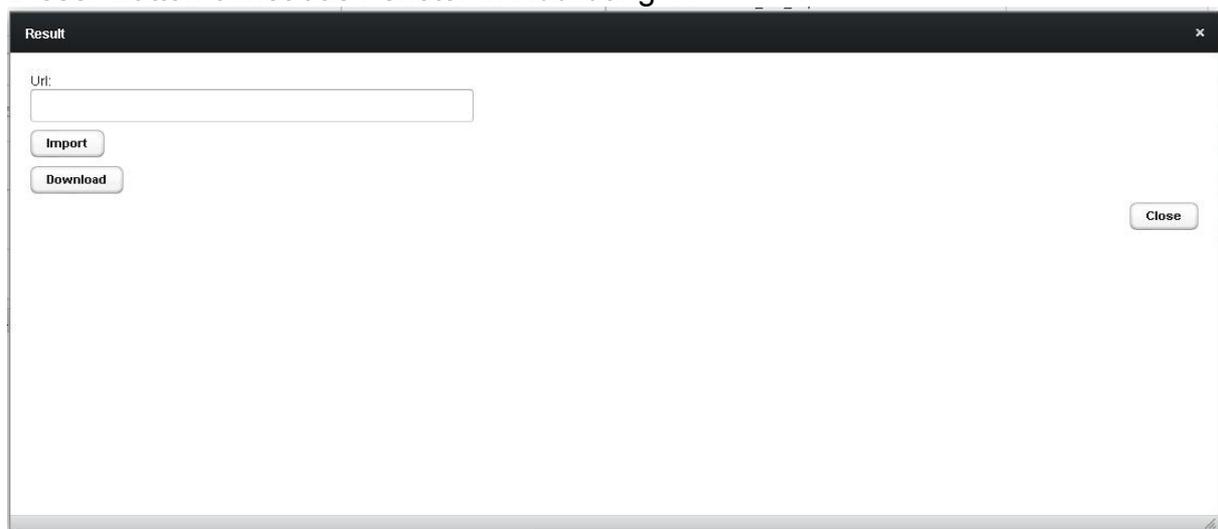


Abbildung 11: Import from rdf

Hier kann der Pfad zu der Datei angegeben werden, die für den Import als Datenquelle dienen soll. Nach einem Klick auf 'Import' erscheint diese in der Datenkategorieauswahl und kann wie beschrieben für die Suche verwendet werden. 'Download' erzeugt die passende dcif-Datei, die in ISOcat importiert werden kann.

### 5.2.3 Beispieldurchlauf für <http://www.lingvoj.org/lingvoj.rdf>

Es ist wichtig, dass dieser Durchlauf einen Fall darstellt, der als Funktionalität nicht erwünscht ist: der Import eines fremden Vokabulares und die gleichzeitige

Kandidatensuche. Extrahierte Informationen, die die Volltextbeschreibung bilden, werden hier ebenfalls nicht verarbeitet, auch wenn diese in die dcif-Datei geschrieben wurden. Die Volltextbeschreibungen für die Datenkategorien stammen, sofern sie automatisch eingefügt werden, aus der Datenbanktabelle pid\_descr beziehungsweise dem zugehörigen Webservice, werden also mit dem Datenbankupdate aus ISOcat extrahiert.

Das Vokabular Lingvoj<sup>64</sup> beschreibt, wie Sprachen in jeweils anderen Sprachen genannt werden. Die jeweiligen ISO 639-3 Sprachcodes bilden dabei den Identifier der Sprachressource und damit die PID der potentiellen Datenkategorie. Die dcif-Datei für diesen Beispieldurchlauf liegt dieser Arbeit bei<sup>65</sup>.

DATACATEGORYSELECTION	DATACATEGORY	NAME
▶ uby_2012		
▼ lingvoj.rdf		
	my-jv	Javanese language
	my-lol	Mongo language
	my-kok	Konkani language
	my-kos	Kosraean language
	my-pnt	Pontic Greek
	my-gor	Gorontalo language
	my-kg	Kongo language
	my-got	Gothic language
	my-ki	Gikuyu language

*Abbildung 12: Beispieldurchlauf Import*

Die URIs der Webressourcen werden als PIDs für die Datenkategorien und der englische Name als Name der Datenkategorie. Mit Rechtsklick → Choose können diese Informationen in das Eingabefeld als Vorlage für die Kandidatensuche kopiert werden.

Abbildung 13 und 14 zeigen die Ergebnismengen für die Suche nach zwei der importierten Datenkategorien. Abbildung 13 zeigt ein sehr präzises Beispiel.

64 <http://ontorule-project.eu/parrot/parrot?documentUri=http://www.lingvoj.org/ontology>

65 <results/importfiles/lingvoj.RDF.dcif>

APPROVED	RELATION	NAME	DATACATEGORY	NAME SIMIL
<input type="checkbox"/>	rel:sameAs	scottish gaelic	DC-684	1.0
<input type="checkbox"/>	rel:related	creoles and pidgins, english b	DC-627	0.1999999

Abbildung 13: Beispieldurchlauf hohe Präzision 'scottish gaelic'

Es wurde die korrekte Entsprechung der bereits enthaltenen Datenkategorie gefunden. Da die Namensähnlichkeit 1 beträgt, wurde sameAs als Relation vorgeschlagen. 'creoles and pidgin' ist das Ergebnis der Wordnetsuche.

Die hohe Präzision folgt hier aus der Seltenheit der Begriffe Scottish und Gaelic.

Abbildung 14 zeigt ein ungenaueres Ergebnis für 'West Frisian Language'.

APPROVED	RELATION	NAME	DATACATEGORY	NAME SIMILIAR
<input type="checkbox"/>	rel:related	australian languages	DC-566	0.66666666
<input type="checkbox"/>	rel:related	sorbian languages	DC-989	0.61904761
<input type="checkbox"/>	rel:related	written language	DC-3558	0.61904761
<input type="checkbox"/>	rel:related	salishan languages	DC-893	0.61904761
<input type="checkbox"/>	rel:related	sign language	DC-4667	0.57142857
<input type="checkbox"/>	rel:related	meta language	DC-2489	0.57142857
<input type="checkbox"/>	rel:related	metadata language	DC-2543	0.57142857
<input type="checkbox"/>	rel:related	village sign language	DC-4672	0.57142857
<input type="checkbox"/>	rel:related	otomian languages	DC-859	0.57142857
<input type="checkbox"/>	rel:related	wakashan languages	DC-984	0.57142857
<input type="checkbox"/>	rel:related	siouan languages	DC-908	0.57142857
<input type="checkbox"/>	rel:related	athapascan languages	DC-565	0.57142857
<input type="checkbox"/>	rel:related	father's language	DC-4515	0.57142857
<input type="checkbox"/>	rel:related	mother's language	DC-4516	0.57142857
<input type="checkbox"/>	rel:related	national language	DC-3702	0.57142857

Abbildung 14: Beispieldurchlauf niedrige Präzision 'West Frisian Language'

Das Wort 'language', kommt in vielen Namen vor, was wiederum zu vielen

Kandidaten führt. Die Termüberdeckung wird aufgrund der fehlenden Volltextbeschreibung nicht berechnet, was eine Auswertung nur für die (ohnehin verlässlichere) Namensähnlichkeit zulässt. Die Ergebnisse wurden nach dieser sortiert und es ist offensichtlich, dass keiner der Kandidaten in diesem Wert besonders hervorsticht. Dies lässt es sinnvoll erscheinen, ISOcat um diese Datenkategorie zu erweitern. Es wäre also zu überprüfen, ob die dcif-Datei des importierten Vokabulares in ISOcat eingebunden werden sollte.

Abbildung 15 zeigt das Fenster, welches sich mit 'Create Exportfile' öffnen lässt. Hier wurden die für 'Scottish Gaelic' gefundenen Relationen in der Ergebnistabelle bestätigt und damit übernommen. Die Datei kann nun editiert, das Set mit einem möglichst aussagekräftigen Namen versehen und anschließend heruntergeladen werden.



Abbildung 15: Beispieldurchlauf Exportdatei

Dieser beispielhafte Durchlauf macht deutlich, warum es sich noch nicht um eine vollständig produktiv einsetzbare Oberfläche handelt. Während die Suche nach Kandidaten für in ISOcat enthaltene Datenkategorien einwandfrei funktioniert, bringt der Import von neuen Vokabularen zwei Probleme mit sich, die im Rahmen dieser Arbeit nicht gelöst wurden.

Es wird nicht geprüft, ob eine Datenkategorie in ISOcat enthalten ist, wenn die Exportdatei erzeugt wird. Stattdessen wird angenommen, dass isocat:[pid] existiert. Diese Überprüfung lässt sich über mehrere Wege einfach umsetzen: /listdc der Webservices liefert alle bekannten Datenkategorien genau wie DBAccess →

getAllDc()), wenn eine lokale Datenbasis vorhanden ist. Über diese kann dann die PID validiert werden.

Relationen der RELcat können auch Datenkategorien mit externen Ressourcen verbinden. Beispielsweise enthält RELcat die Relation

```
isocat:DC-3055    rel:sameAs    http://purl.org/linguistics/gold/AbessiveCase
```

Diese Möglichkeit ist nicht durch diese Arbeit abgedeckt und aufgrund des Open World-Charakters der Semantic Web- / Linked Data-Technologie nur sehr schwer umsetzbar, da dafür das gesamte Semantic Web nach Kandidaten durchsucht werden müsste. Über RELcatComparator können aber bereits in RELcat vorhandene Informationen hierfür genutzt werden, um für gefundene Kandidaten deren externe Webressourcen in die Kandidatenmenge mit aufzunehmen.

Da die Oberfläche ursprünglich nur zu Demonstrationszwecken für die einzelnen Funktionen beziehungsweise als Beispielimplementierung entworfen wurde, ist es jedoch erfreulich, dass sie zumindest für die Suche von Kandidaten innerhalb der ISocat bereits praktisch einsetzbar ist.

Sämtliche Funktionen rufen ihre Informationen dabei von den Webservices ab, außer 'Import from rdf' und 'create Importfile'.

## 6 Evaluation

### 6.1 Evaluation der Datenaufbereitung

Da sich sowohl ISOcat als auch RELcat noch in der Entstehungsphase befinden, ist davon auszugehen, dass die Datenbasis sich in Zukunft ändern wird. Auf der beiliegenden CD befinden sich Dateien, die die momentane Datenbasis widerspiegeln und Textdateien mit detaillierteren Testergebnissen.

#### **a Statistik**

Hierbei handelt es sich um eine statistische Auswertung der Daten, die im Zuge des Datenbankupdates vorbereitet werden. Damit lassen sich beispielsweise die effizientesten Parameter der Terminologie-Extraktion ermitteln, was die Präzision der Begriffsanalyse sowie die Sparsamkeit der Datenbank beeinflusst.

Die Statistik umfasst dabei folgende Punkte:

<b>Statistik</b>	<b>Erklärung</b>
DataCategories	Anzahl der Datenkategorien
DataCategorySelections	Anzahl der Datenkategoriesets
DC per DCS Min/Max/Avg	Minimale/Maximale/Durchschnittliche Anzahl der Datenkategorie eines Datenkategoriesets
Terms	Anzahl der extrahierten Begriffe
Unique Terms	Anzahl der distinkten extrahierten Begriffe
Terms per DCS Min/Max/Avg	Minimale/Maximale/Durchschnittliche Anzahl der extrahierten Begriffe pro Datenkategorieset
Terms per DC Min/Max/Avg	Minimale/Maximale/Durchschnittliche Anzahl der extrahierten Begriffe pro Datenkategorie
Candidates per DC Min/Max/Avg	Minimale/Maximale/Durchschnittliche Anzahl der für eine Datenkategorie gefundenen Kandidaten
Candidate - Relations	Gesamtanzahl der Kandidatenrelationen
DCs as candidates	Anzahl der Datenkategorien, die irgendwann als Kandidaten in Frage kamen

Statistik	Erklärung
DCs not candidates for anything	Anzahl der Datenkategorien, die nie als Kandidaten in Frage kamen

Die Ergebnisse können abhängig von der Reihenfolge der Abarbeitung variieren, da die Zugehörigkeit von Datenkategorien nach dem ersten Auftreten zugeordnet wird und sich die statistischen Werte damit verschieben können. Für das Funktionieren der in dieser Arbeit entwickelten Suchmethoden spielt die Zugehörigkeit von Datenkategorien zu Datenkategoriesets aber keine Rolle, weshalb das vernachlässigbar ist.

### **b** *RELcat-Test*

Da es bei der Hauptaufgabe dieser Arbeit darum geht, Kandidaten für Relationen für RELcat zu finden, ist es auch sinnvoll, bereits vorhandene Daten der RELcat zur Evaluierung heranzuziehen.

Dieser Test sucht für jede Datenkategorie x aus der Datenbank die Datenkategorien y, sodass x mit y in irgendeiner Relation in RELcat in Verbindung gebracht wird. Anschließend werden diese mit den gefundenen Kandidaten verglichen. Wenn eine Datenkategorie in RELcat nicht in der Kandidatenmenge erscheint, gilt dieser Test als nicht bestanden. Andernfalls gilt der Test als bestanden.

Dabei kann nur bestätigt werden, dass ein Kandidat richtig ist, es kann kein Kandidat widerlegt werden. Auch kann aus den Ergebnisdaten kein Schluss auf die Richtigkeit der Kandidatenmenge gezogen werden, da RELcat nicht alle zwischen Datenkategorien herrschenden Relationen enthält<sup>66</sup>.

Der Test erzeugt 3 Arten von Dateien:

**RELcatTest\_found.txt** enthält eine Auflistung aller korrekt vorhergesagten Relationen mit dem zu diesem Zeitpunkt geltenden Verhältnis zwischen Treffern zur Gesamtzahl.

**RELcatTest\_not\_found.txt** enthält analog zu found.txt die Informationen über nicht vorhergesagte Relationen.

**RELcatTest\_stats.txt** enthält folgende Übersicht:

---

<sup>66</sup> Wäre dem so, wäre diese Arbeit nicht zu rechtfertigen.

- Relationen
  - Anteil erfolgreich vorhergesagter
  - Anteil nicht vorhergesagter
- Datenkategorien
  - Anzahl in Datenbank
  - Anzahl in RELcat
  - Anzahl Datenkategorien, deren RELcat-Relationen vorhergesagt wurden
  - Anzahl Datenkategorien, deren RELcat-Relationen nur teilweise vorhergesagt wurden
- Auflistung aller nicht vorhergesagten Relationen mit PID und Namen
- (...)
- Auflistung aller Datenkategorien, die fälschlicherweise irgendwann nicht vorhergesagt wurden
- (...)

### **c Terminologie-Extraktion-Test<sup>67</sup>**

Dieser Test dient dazu, die optimale Konfiguration des Parameters Mindestsignifikanz und das passende Signifikanzmaß für die Terminologie-Extraktion bestimmen zu können.

Die Toolbox unterstützt die beiden Signifikanzmaße likelihood- und frequency-ratio, für die laut der ihr beiliegenden Dokumentation jeweils unterschiedliche Mindestsignifikanzen sinnvoll sind. „The frequency ratio usually needs a higher value of the minimum significance (around 60) than does the likelihood ratio (around 20, as reflected by the default).“[8]

Zur Bestimmung des für diesen Fall besser passenden Signifikanzmaßes und der optimalen Mindestsignifikanz werden alle Datenkategoriesets und anschließend die Profile jeweils für die beiden Signifikanzmaße mit der in der Dokumentation empfohlenen Mindestsignifikanz +/- 20 in 5er-Schritten reduziert und mittels Statistik und RELcatTest evaluiert. Dabei wird die Suche nach Kandidaten auf die Suche anhand der Volltextbeschreibungen eingeschränkt, um Einflüsse der anderen Suchmethoden zu vermeiden.

Diese Auswertung dauert aufgrund der Dauer des Updates mehrere Tage. Dies kann zu Problemen führen, wenn der Webservice der ISOcat beispielsweise aus Wartungsgründen zeitweise nicht erreichbar ist. Aus diesem Grund und zur Gewährleistung der Reproduzierbarkeit wird dieser Test auf den dieser Arbeit beiliegenden ISOcat-Backups ausgeführt. Dies ermöglicht insbesondere eine

---

<sup>67</sup> Bei der Programmierung dieses Tests ist ein kleiner Fehler passiert – die Begriffe werden bereits vollständig bearbeitet, also auch auf ihren Porter-Stamm reduziert. Dies hat allerdings keine Auswirkungen auf das Ergebnis, da es für jeden Durchlauf durchgeführt wurde. Es führt nur zu der unschönen Beobachtung, dass die Ergebnisse hier den Ergebnissen, die im BaseTePortTest unter 'tepo' aufgeführt sind, entsprechen, obwohl sie eigentlich den Ergebnissen unter 'te' entsprechen müssten.

Anpassung der Testmenge, sodass Schnelltests auf einem Subset der gesamten Daten durchgeführt werden können.

Für den Ablauf des Tests werden verschiedene Datenbanken TeTest[Methode] [Signifikanz] angelegt – dies ist explizit im Quellcode festgelegt, um zu verhindern, dass versehentlich die aufbereiteten Daten gelöscht werden. Diese Datenbanken werden nicht gelöscht, um unerwartete Ergebnisse nachprüfen zu können. Die Exportdateien der Datenbanken, die für diese Arbeit angelegt wurden, liegen im Ordner results/tests/tetest/db\_files/.

Zwingend benötigt werden diese Datenbanken nach dem Test nicht mehr und können gelöscht werden.

Für jeden Durchlauf werden die in 6.1.1a und 6.1.1b beschriebenen Dateien mit Namen der Messmethode und der Mindestsignifikanz angelegt. Die zugehörigen Ergebnisdateien befinden sich im Ordner results/tetest/logs/.

Die Diagramme in Abbildungen 16 und 17 geben Aufschluss darüber, wie die Änderung der Mindestsignifikanz und des Signifikanzformulares sich auf die Menge der Begriffe auswirkt.

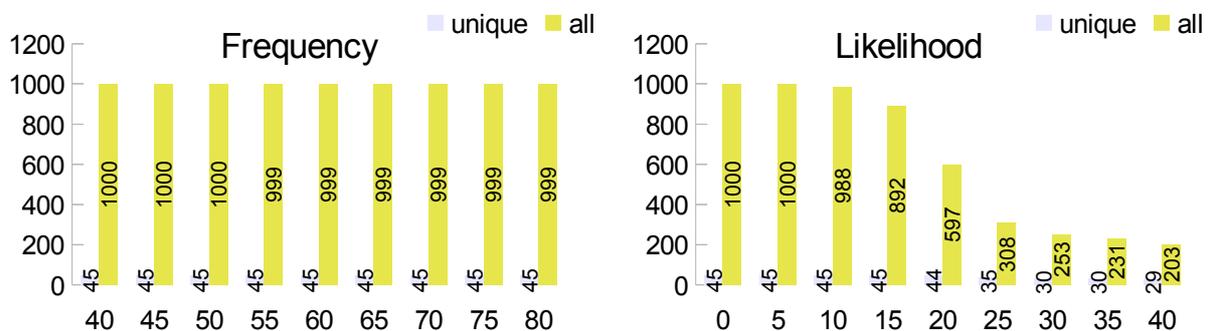


Abbildung 16: Anzahl der Begriffe insgesamt (in hunderten)

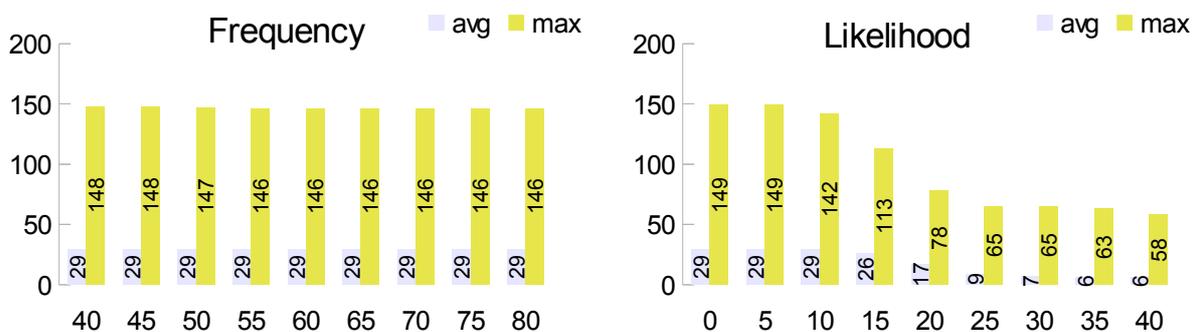


Abbildung 17: Anzahl der Begriffe pro Datenkategorie

Die Begriffsmenge, die bei höheren Mindestsignifikanzen extrahiert wird, ist dabei immer eine Untermenge oder gleich der Begriffsmenge, die bei niedrigeren Mindestsignifikanzen extrahiert wird, da jeder Begriff, der mit einer Signifikanz  $x$  relevant ist, natürlich auch mit einer Signifikanz  $<x$  relevant ist.

Da eine Datenkategorie als Kandidat gilt, wenn deren Begriffsmenge eine Schnittmenge mit der Begriffsmenge der Vorlage besitzt, ist es wenig überraschend, dass diese Verteilung sich direkt auf die Verteilung der Kandidaten auswirkt. Abbildung 18 zeigt die durchschnittliche und maximale Anzahl der für jede Datenkategorie gefundenen Kandidaten.

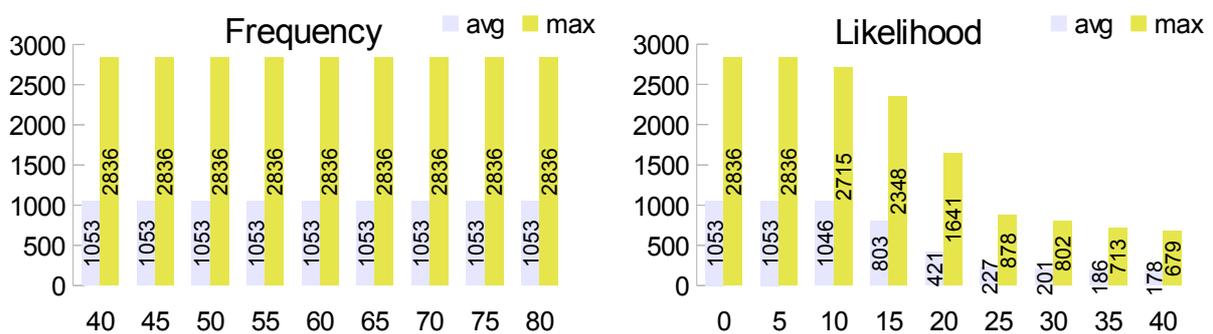


Abbildung 18: Anzahl der Kandidaten

Da die Begriffsmengen höherer Signifikanzen in den Begriffsmengen niedrigerer Signifikanzen enthalten sind und Kandidatenmengen das Ergebnis der Schnittmenge zweier Begriffsmengen sind, gilt diese Teilmengenbeziehung für diese ebenfalls. Wenn ein Kandidat mit einer Mindestsignifikanz  $x$  gefunden wurde, dann wird er auch mit einer Mindestsignifikanz  $<x$  gefunden.

Die Verteilung dieser Werte lässt vermuten, dass das Häufigkeitsmaß keinen deutlichen Einfluss auf die Terminologieextraktion hat. Ein weiterer Hinweis hierauf und außerdem ein Hinweis darauf, dass ein minimaler Einfluss messbar ist – und damit kein Implementierungsfehler vorliegt, findet sich in der Differenz der Gesamtanzahlen der geknüpften Verbindungen für die maximalen und minimalen Signifikanzwerte des jeweiligen Signifikanzmaßes. Ein hoher Wert für diese Differenz zeigt, dass die Mindestsignifikanzen der Maße Einfluss auf das letztendliche Endergebnis – die Menge der gefundenen Kandidaten – haben.

Während dieser Wert für das Häufigkeits-Maß bei  $3'549'495 - 3'548'195 = 300$  liegt,

ist eine Veränderung beim Likelihood-Maß mit  $3'549'543 - 601'816 = 2'947'727$  wesentlich deutlicher zu spüren.

Dies lässt sich damit begründen, dass aufgrund der Mindestfrequenz 1 im Text, die aufgrund der potentiell knappen Formulierungen sinnvoll ist, das Häufigkeitsmaß für diese Aufgabe nicht sinnvoll ist. Eine stichprobenhafte Überprüfung der zugehörigen Datenbanken bestätigte dies, da im wesentlichen nur die Stoppwörter aus den Texten entfernt wurden und die Ergebnisse mit geringen Schwankungen denen des Likelihood-Maßes mit einer Mindestsignifikanz von 0 oder 5 entsprechen. Dies kann anhand der Exportdateien der Datenbanken im Anhang nachgeprüft werden.

Um die Auswertung nicht unnötig aufzublähen, wird die Methode für diese Arbeit auf Likelihood beschränkt. Die Ergebnisse des Häufigkeitsmaßes befinden sich trotzdem auf der beiliegenden CD.

Abbildung 19 zeigt die Verteilung der insgesamt vorgeschlagenen Relationen zwischen Datenkategorie und Kandidat und die Verteilung der Anzahl der Datenkategorien, die nie als Kandidaten vorgeschlagen wurden.

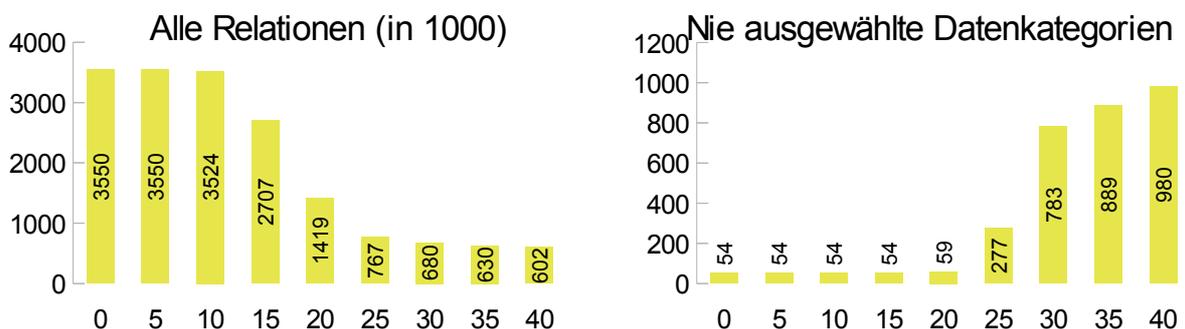


Abbildung 19: Gefundene Relationen

Auch hier trifft die Untermengeneigenschaft aus den gleichen Gründen wie oben zu.

Abbildung 20 zeigt das Verhältnis zwischen korrekt vorhergesagten Relationen und verpassten Relationen. Korrekt vorhergesagt wurde eine Relation, wenn das Subjekt der Vorlage und das Objekt einem Kandidaten entspricht. Nicht vorhergesagt wurde eine Relation, wenn diese in RELcat steht, aber nicht von der Kandidatensuche gefunden wurde.

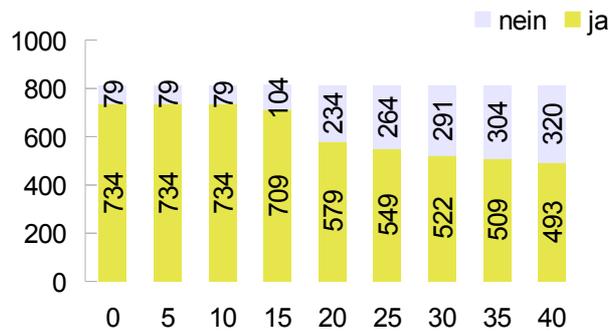


Abbildung 20: Vorhergesagte Relationen

Die Diagramme in Abbildung 17 und 18 zeigen, wie die Menge der Kandidaten, analog zur Menge der Begriffe, bei Mindestsignifikanz 25 einen Punkt erreicht, an dem nach einer starken Steigerung der Genauigkeit in Form einer Reduktion der Kandidatenmenge keine weiteren auffälligen Veränderungen in den Werten mehr auftreten. Die Werte in Abbildung 19 zeigen ebenfalls eine wachsende Präzisierung<sup>68</sup> bis zu Mindestsignifikanz 25, wobei ein starker Anstieg der Anzahl der nie als Kandidaten in Frage kommenden Datenkategorien bis auf Mindestsignifikanz 30 anhält.

Analog dazu verschlechtert sich die Menge der korrekt vorhergesagten Relationen.

Da die durchschnittliche Anzahl der Kandidaten bei Mindestsignifikanz 30 noch bei dem verhältnismäßig hohen Wert 201 liegt und es zu erwarten ist, dass die Volltextbeschreibungen der Datenkategorien in ISOcat später ausführlicher werden, werde ich für diese Arbeit die Mindestsignifikanz auf 30 festlegen. Die Verbesserung der Genauigkeit ist bis zu diesem Punkt deutlich spürbar und die Menge an gefundenen Kandidaten ist noch mehr als ausreichend, vor allem, da es noch 3 andere Suchmethoden gibt.

#### **d BaseTePorter-Test**

Dieser Test wertet, wie der Terminologie-Extraktions-Test, verschiedene Konfigurationen der Datenaufbereitung aus.

Statt aber die Mindestsignifikanz und das Signifikanzmaß zu testen, werden hier die einzelnen Methoden selbst getestet :

<sup>68</sup> Hier ist nicht das Maß Precision gemeint

base → Text nach Leerzeichen tokenisiert bildet Begriffe.

te → Ergebnisse der Terminologie-Extraktion bilden Begriffe.

po → Auf Porter-Stamm reduzierte Ergebnisse der Terminologie-Extraktion bilden Begriffe.

black → Ignorieren der Einträge der Blacklist.

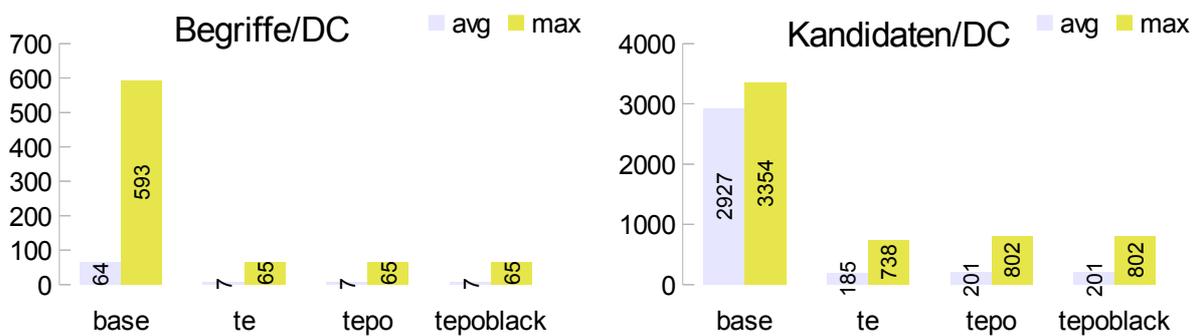


Abbildung 21: Anzahl der Begriffe pro Datenkategorie

Als Signifikanzmaß wurde der im Terminologie-Extraktions-Test ermittelte Wert 30 festgelegt. Die vollständigen Ergebnisse liegen zusammen mit den Ergebnissen für Mindestsignifikanz 25 auf der beiliegenden CD<sup>69</sup>.

Sucht man die Kandidaten anhand der unbearbeiteten Volltextbeschreibung, zeigt sich, dass eine große Menge von Kandidaten gefunden wird. Grob lässt sich sagen, dass fast jede Datenkategorie ein Kandidat für eine Relation mit fast jeder anderen Datenkategorie wäre, was insbesondere von Überschneidungen bei Stopwörtern gefördert wird. Mit der Extraktion relevanter Begriffe lässt sich diese Ergebnismenge auf ein überschaubares Maß reduzieren. Werden die Begriffe auf ihren Porter-Stamm reduziert, zeigt sich eine geringe Veränderung der Anzahl der Begriffe, die in diesen Diagrammen keinen Einfluss haben. Die Erhöhung der gefundenen Kandidaten zeigt aber, dass sich die Ergebnismenge dahingehen verbessert hat, dass einzelne Datenkategorien mit in die Kandidatenmenge aufgenommen wurden, die andernfalls ignoriert worden wären, da die extrahierten Begriffe auf Wortformebene nicht übereinstimmten. Die Auswirkungen der Blacklist sind vernachlässigbar, aber in den entsprechenden Ergebnisdateien dokumentiert.

<sup>69</sup> results/tests/BaseTePorterTest

Abbildung 22 zeigt, wie sich das Verhältnis von vorhergesagten zu nicht vorhergesagten Relationen verhält.

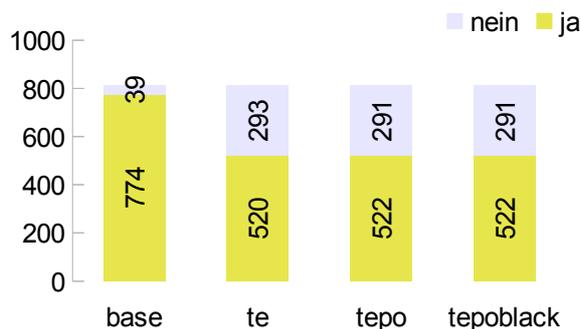


Abbildung 22: Vorhergesagte Relationen

Das gute Abschneiden von 'base' in diesem Test ist irreführend, da bei diesem Durchlauf fast alle Datenkategorien miteinander in Relation standen, also natürlich auch nahezu alle Richtigen.

Es zeigt sich, dass, obwohl die Mindestsignifikanz mit 30 auf einem vergleichsweise hohen Wert liegt, bereits knapp 2/3 der Relationen in RELcat vorhergesagt werden konnten. Die Ergebnisse für Mindestsignifikanz 25 zeigen ein Verhältnis von 547/813 korrekt vorhergesagten Relationen.

Die Reduzierung auf den Porter-Stamm schafft 2 zusätzliche korrekt vorhergesagte Relationen, was ein vergleichsweise geringer Anteil ist. Die Vergrößerung der Kandidatenmenge zeigt aber, dass eine potentielle Fehlerquelle – Wortformen – eliminiert wurde.

Die Blacklist, die zum Zeitpunkt der Arbeit nur aus dem Wort 'contain' besteht, bringt keine in den Diagrammen sichtbaren Änderungen aber es ist immer sinnvoll einen Mechanismus zu haben, über den sich Wörter aus der Bearbeitung ausschließen lassen. Der Eintrag 'contain' hat dabei die Gesamtmenge an Begriffen von 36,767 auf 36,762 reduziert.

## 6.2 Evaluation des Imports

Eine Evaluation des Importes erscheint nicht sinnvoll, da die Daten hierfür nicht effektiv bewertbar sind. Es gibt keine Möglichkeit, das Mapping der Tags der

unterschiedlichen Welten zu evaluieren, da bei der Erstellung der Daten ein großer Interpretationsspielraum gegeben ist. Als Beispiel hierfür sei das in der dieser Arbeit beiliegenden mappings.properties festgeschriebene Mapping

`www.w3.org/2000/01/RDF-schema#comment = explanation`

genannt. Mir persönlich erscheint es sinnvoll, einen Kommentar auf das dcif-Tag `explanation` zu mappen. Zum Zeitpunkt dieser Arbeit wurde allerdings in der gold2010-Ontologie der Text, der in ISOcat unter der Eigenschaft `dcif:definition` steht, unter

`www.w3.org/2000/01/RDF-schema#comment`

beschrieben. Dieser wurde dann auf `<dcif:explanation>` gemappt. Die Richtigkeit des Mapping ist stark von der jeweiligen Situation abhängig und nicht frei von Einflüssen subjektiver Sichtweisen. Dies auf Basis eines nicht automatisiert erstellten Mappings objektiv zu bewerten, wäre eine Sisyphusarbeit. Die automatische Erstellung eines solchen Mappings wiederum ist ein Thema, welches genügend Stoff für eine weitere Arbeit liefert.

Auch wären Eigenschaften wie der Abdeckungsgrad von Tripel und dcif-Eigenschaften nichtssagend oder sogar irreführend, da Tripelanzahlen in Ontologien in der offenen Welt des Semantic Web schon alleine wegen ihrer Offenheit keine Aussagekraft haben – ähnlich dem Maß Lines of Code, welches als Qualitätskriterium für Quellcode mehr Probleme mit sich bringt als löst<sup>70</sup>. Eine Ontologie, die hauptsächlich aus Strukturinformationen besteht, würde, selbst wenn alle wichtigen Informationen extrahiert wurden, einen geringen Abdeckungsgrad von Tripel auf dcif-Tags enthalten, während ein hoher Abdeckungsgrad keinerlei Aussagekraft über die Richtigkeit der Zuordnung enthalten würde.

Die Datei `gold2010.owl.dcif`, die in dem Beispieldurchlauf in Kapitel 3.4 erzeugt wurde, wurde gegen das dcif-Schema<sup>71</sup> mit `jing`<sup>72</sup> getestet und durch den Systemadministrator validiert<sup>73</sup>. Andere Dateien werden je nach Mapping genauso erzeugt und sollten damit zumindest syntaktisch valide sein. Die semantische

---

70 Nutzung von Bibliotheken, ineffizientes Programmieren, absichtliches „Aufbauschen“

71 <https://catalog.clarin.eu/isocat/12620/schemas/DCIF.html>

72 <http://www.thaiopensource.com/relaxng/jing.html>

73 Die von ISOcat heruntergeladene, und damit vorher importierte, dcif-Datei `results/importfiles/ISOcat/gold-2010.owl.dcif` befindet sich auf der CD. Öffentlich zugänglich ist dieses Datenkategorieset nicht, da es nur im privaten Bereich importiert wurde.

Qualität hängt ausschließlich von dem vom Nutzer erstellten Mapping ab.

### 6.3 Evaluation der Suchmethoden

Hierfür wurde der RELcatCandidateFinderTest implementiert, der die Suchmethoden einzeln und kombiniert auswertet, wobei die mit Mindestsignifikanz 30 erstellte Datenbank genutzt wurde. Die vollständigen Ergebnisse befinden sich im Ordner results/tests/candfindtest.

Abbildung 23 zeigt die Verteilung der vorhergesagten Relationen zu den nicht vorhergesagten und die Anzahlen der Kandidaten pro Datenkategorie für die einzelnen Suchmethoden.

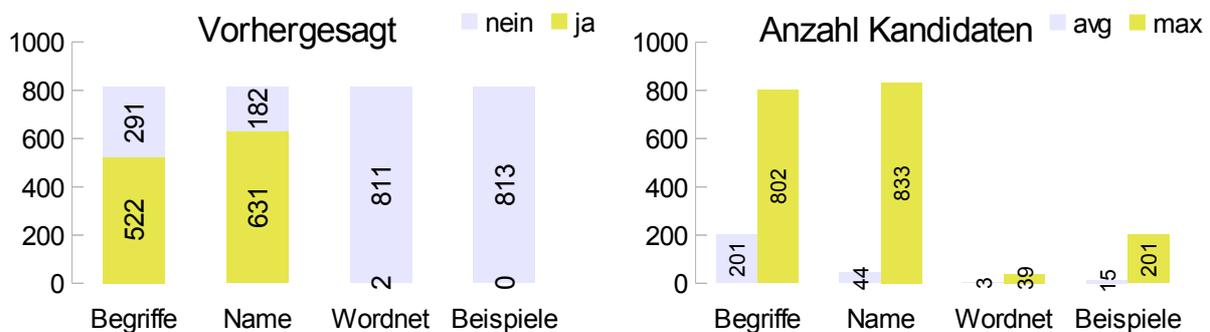


Abbildung 23: 1 Methoden einzeln

Auffällig ist die geringe Menge von Kandidaten, die bei der WordNet®-Suche und der Beispiel-Suche zurückgegeben werden. Dies liegt an der geringen Anzahl der Datenkategorien, die die Voraussetzungen erfüllen, um für diese Methoden als Kandidaten in Frage zu kommen : Für die WordNet®-Suche muss ein Konzept x in WordNet® mit einem anderen Konzept y verbunden sein, wobei x und y in ISOcat den gleichen Namen oder gemeinsame Teilnamen haben. Dies trifft zum Zeitpunkt der Arbeit auf 222 Datenkategorien zu.<sup>74</sup> Die Beispiel-Suche erfordert, dass für die Datenkategorien, die als Vorlage oder als Kandidat dienen sollen, in ISOcat mindestens ein Beispiel angegeben wurde. Dies trifft auf 873 Datenkategorien zu<sup>75</sup>,

<sup>74</sup> Siehe Ergebnisdatei CandidateFinderTestWordNet.txt

<sup>75</sup> Siehe Datenbanktabelle pid\_example\_stats. (pid\_example kann Mehrfacheinträge für eine PID enthalten)

von denen 8 in RELcat erwähnt sind. Für die beiden Suchmethoden kommt also generell nur ein Bruchteil der 3368 Datenkategorien in Frage. Bei der WordNet®-Suche handelt es sich zusätzlich um eine sehr genaue Suchmethode, bei der man eigentlich davon ausgehen kann, dass das Ergebnis korrekt ist. Diese Genauigkeit schränkt die Ergebnismenge natürlich weiter stark ein.

Kombiniert man die Ergebnisse der Suchmethoden auf eine Weise, bei der sie sich gegenseitig ergänzen, also eine gemeinsame Kandidatenmenge füllen, ändern sich die Ergebnismengen entsprechend. Abbildung 24 bis Abbildung 26 zeigen die Ergebnisse für jede 2er Kombination der Methoden.<sup>76</sup>

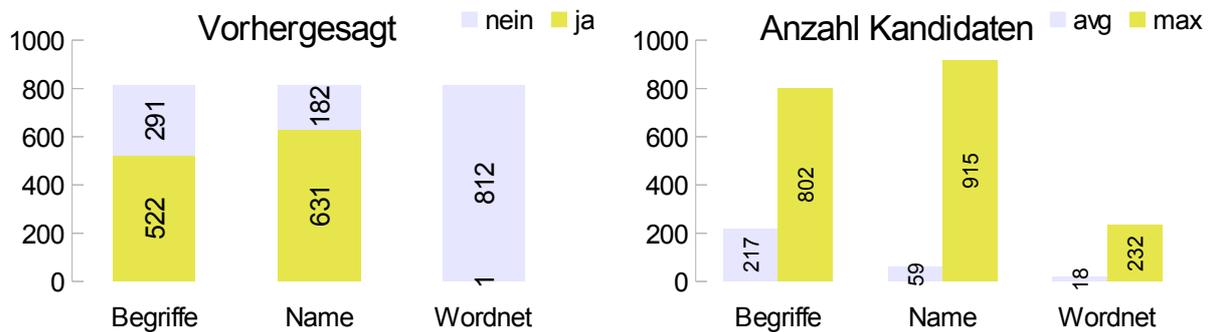


Abbildung 24: Beispiele und ...

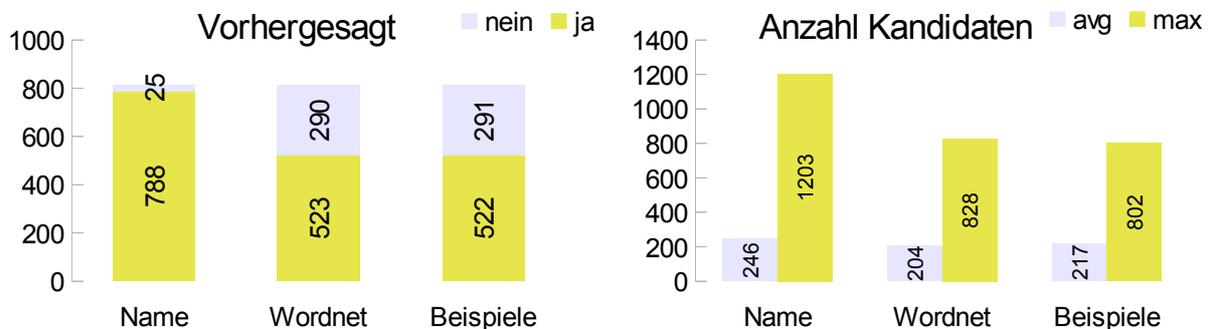


Abbildung 25: Begriffe und ...

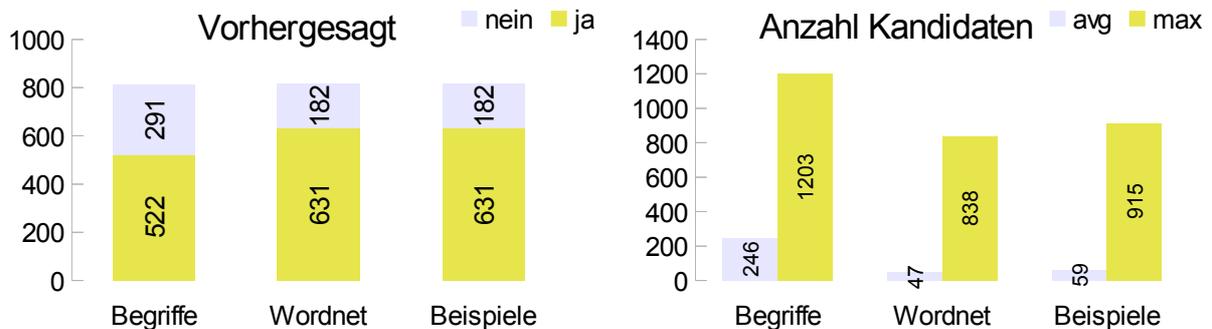


Abbildung 26: Name und ...

Die Diagramme in Abbildung 24 bis Abbildung 26 zeigen bereits, dass sich die Anzahl der Kandidaten verschiedener Suchmethoden nicht einfach aufaddiert. Das liegt daran, dass es sich dabei nicht um disjunkte Mengen handelt – Kandidaten können von mehreren Methoden in der Ergebnismenge zurückgegeben werden. Diese Eigenschaft kann einen weiteren Ansatzpunkt für weiterführende Arbeiten aufzeigen, nämlich die Gewichtung der Kandidaten anhand der Suchmethode oder des Signifikanzwertes beziehungsweise anhand der Anzahl der diesen Kandidaten vorschlagenden Methoden.

Abbildung 27 bis Abbildung 29 zeigen die Ergebnisse für die Kombinationen von 3 Methoden. Die Namen sind abgekürzt.

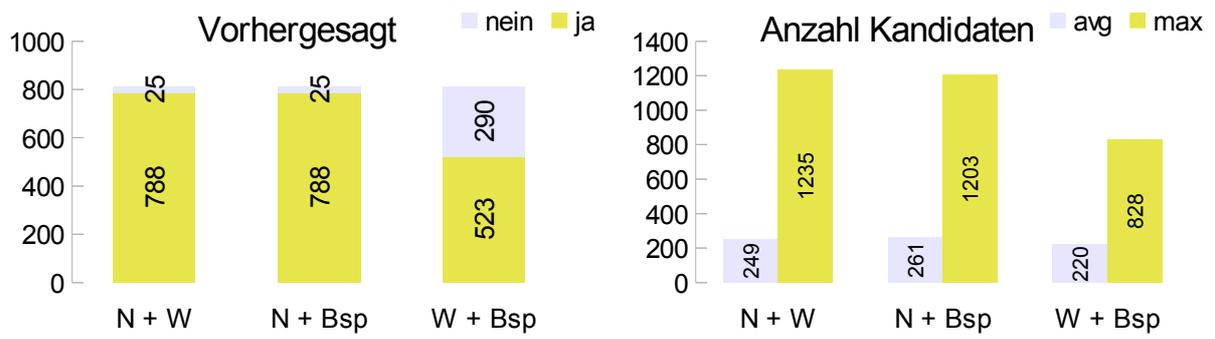


Abbildung 27: Begriffe und ... und ...

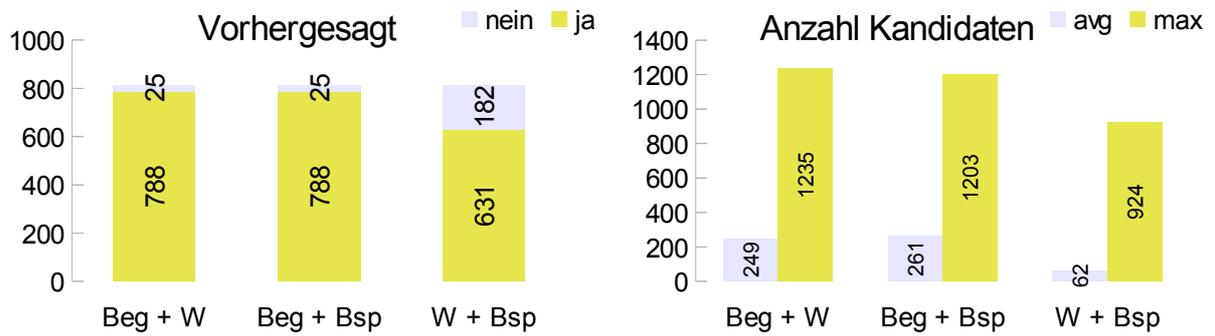


Abbildung 28: Name und ... und ...

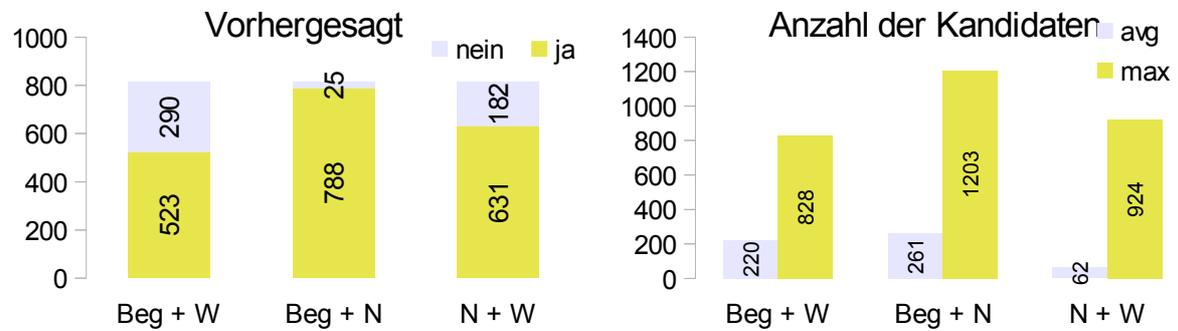


Abbildung 29: Beispiel und ... und ...

Abbildung 30 zeigt schließlich die Ergebnisse, die man erhält, wenn alle Suchmethoden zusammenarbeiten.



Abbildung 30: Alle Methoden kombiniert

25 Datenkategorien, die in RELcat referenziert werden, werden nicht mit den zugehörigen Kandidaten in Verbindung gebracht. Diese nicht vorhergesagten Relationen sind in der folgenden Tabelle aufgelistet. - steht dabei für 'keine Informationen', + steht dafür, dass jede der Datenkategorien Informationen liefert.

Vorlage	Kandidat	Beispiele vorhanden	Volltext vorhanden
DC-1846 comment	DC-3698 etymology gloss (english)	-	+
DC-1846 comment	DC-3700 encyclopedic infomation (regional)	-	+
DC-1846 comment	DC-3705 encyclopedic information (vernacular)	-	+
DC-1975 external reference	DC-3701 etymology source	-	+
DC-3051 infinitive auxiliary	DC-3059 absolute case	Vorlage	+
DC-3687 Bibliography	DC-1975 external reference	-	+
DC-3688 borrowed word(english)	DC-2494 source language	-	+
DC-3697 encyclopedic infomation(english)	DC-1846 comment	-	+
DC-3700 encyclopedic infomation (regional)	DC-1846 comment	-	+

<b>Vorlage</b>	<b>Kandidat</b>	<b>Beispiele vorhanden</b>	<b>Volltext vorhanden</b>
DC-3701 etymology source	DC-1975 external reference	-	+
DC-3702 national language	DC-3692 cross-reference gloss (regional)	-	+
DC-3702 national language	DC-3698 etymology gloss ( english)	-	+
DC-3702 national language	DC-3720 lexical function gloss (regional)	Kandidat	+
DC-3705 encyclopedic information (vernacular)	DC-1846 comment	-	+
DC-3719 lexical function gloss (national)	DC-2649 translating and interpreting	Kandidat	+
DC-3702 national language	DC-481 literal	-	+
DC-3723 lexeme (vernacular)	DC-3516 syntactic word	-	+
DC-3725 notes (anthropology)	DC-1846 comment	-	+
DC-3726 notes (discourse)	DC-1846 comment	-	+
DC-3727 notes (grammar)	DC-1846 comment	Vorlage	+
DC-3728 notes (phonology)	DC-1846 comment	-	+
DC-3729 notes (questions)	DC-1846 comment	-	+
DC-3730 notes (sociolinguistics)	DC-1846 comment	-	+
DC-3731 notes (general)	DC-1846 comment	-	+
DC-3738 morphology (vernacular)	DC-2320 morpho-syntactic	-	+

Dass diese Relationen anhand der Namen nicht vorhersagbar waren, ist offensichtlich, da die Namen kaum Ähnlichkeiten aufweisen. Die Suche anhand der Beispielinformationen konnte ebenfalls nicht fruchten, da nie bei beiden Datenkategorien Beispiele zum Vergleichen angegeben wurden. Um die fehlenden Ergebnisse anhand der Volltexte zu finden, müsste die Mindestsignifikanz der Terminologie-Extraktion verringert werden. Dies würde, wie im zugehörigen Test in

Kapitel 6.1.1c gezeigt, zu einer Erhöhung der Ungenauigkeit führen, die bei dieser Suchmethode – neben der Dauer der Terminologie-Extraktion – auch so schon ein großer Schwachpunkt ist.

## 6.4 Recall, Precision und F-measure

Für Suchalgorithmen gibt es 4 Möglichkeiten, wie die Antwort zu bewerten ist:

	<b>Richtig</b>	<b>Falsch</b>
<b>Gefunden</b>	richtig + gefunden (a)	falsch + gefunden (c)
<b>Nicht gefunden</b>	richtig + nicht gefunden (b)	falsch + nicht gefunden (d)

Precision bezeichnet den Anteil der Richtigen an allen Gefundenen. Recall bezeichnet die Menge der Gefundenen unter allen Richtigen.

$$Precision = \frac{a}{a+c} \quad (4)$$

$$Recall = \frac{a}{a+b} \quad (5)$$

$$F\text{-measure} = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (6)$$

Ein Kandidat gilt hier als richtig, wenn er in RELcat als Objekt in einer Relation zu der Datenkategorie, die als Vorlage gilt, steht. Existiert keine Relation zwischen Vorlage und Kandidat in RELcat, gilt diese Verbindung als falsch.

Um den Einfluss des zu den Suchmethoden passenden Evaluationsmaßes auf Recall und Precision zu testen, wurden diese in 10er-Schritten abgestuft. Die zugehörigen Dateien und ein Abbild der Relationen zum Zeitpunkt des Testdurchlaufs befinden sich im Ordner results/tests/PrecRec der beiliegenden CD.

Aufgrund des frühen Stadiums von RELcat ist die Menge der als korrekt bestätigten Relationen noch nicht repräsentativ, was erwarten lässt, dass die Precision schlecht sein wird.

Folgende Diagramme zeigen, wie die Änderung der geforderten Termüberdeckung sich auf Recall und Precision auswirken. Dabei wurde nur die Volltextsuchmethode verwendet, da wenn diese fehlschlägt, die Termüberdeckung 0 ist. Erwartungsgemäß

liegt die Menge der gefundenen richtigen Relationen weit unter der Menge der gefundenen. Diese werden zur besseren Lesbarkeit nicht mit angezeigt, stehen aber in den Outputdateien. Aufgrund der geringen Precision und des daraus resultierend geringen F-measure wurde dieser in den Diagrammen mit 100 multipliziert, liegt also nicht zwischen 0 und 1, sondern zwischen 0 und 100.

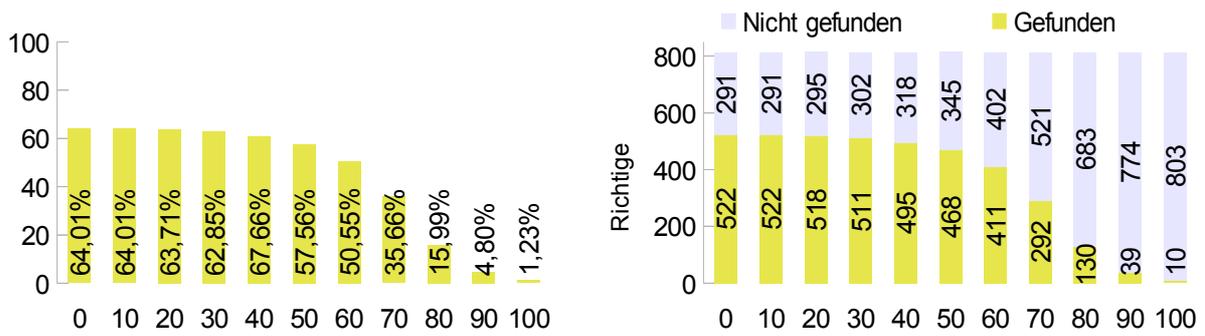


Abbildung 31: Termüberdeckung : Recall

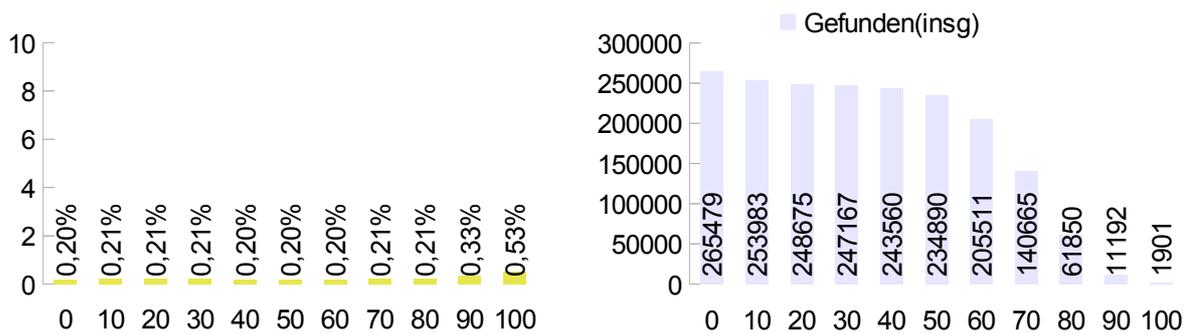


Abbildung 32: Termüberdeckung : Precision

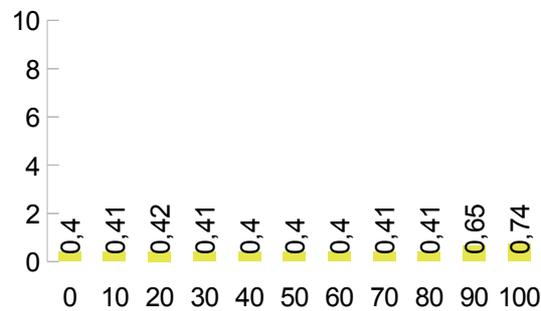


Abbildung 33: Termüberdeckung : F-measure \* 100

Folgende Diagramme zeigen Recall und Precision für die Namensähnlichkeit. Dabei wurden die Ergebnismengen aller Suchmethoden verwendet, da die Namensähnlichkeit unabhängig von der Suchmethode das Ergebnis bewerten kann.

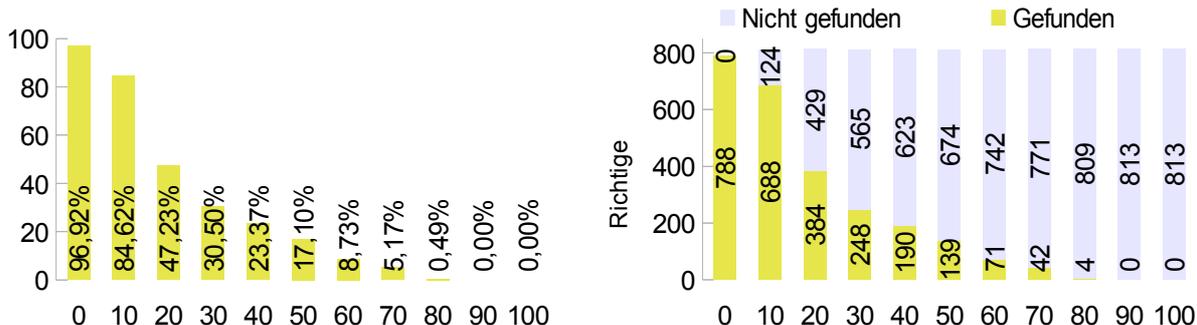


Abbildung 34: Namensähnlichkeit : Recall

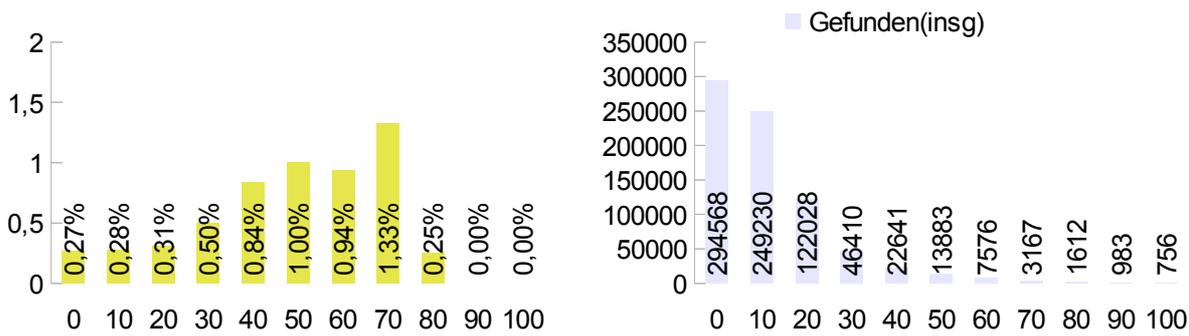


Abbildung 35: Namensähnlichkeit : Precision

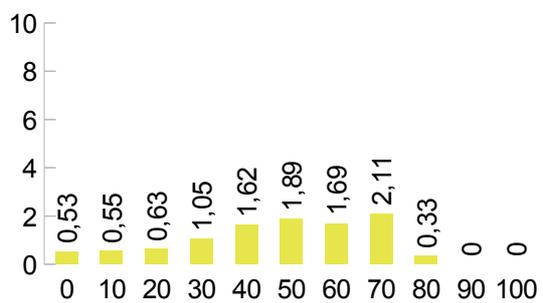


Abbildung 36: Namensähnlichkeit : F-measure \* 100

Die Diagramme auf den folgenden Seiten zeigen die Ergebnisse aufgeschlüsselt nach den einzelnen Suchmethoden. Die Ergebnisse der Suche per Beispiel können dabei ignoriert werden, da keines der Ergebnisse eine in RELcat beschriebene Relation ist (Siehe Abbildung 23) und somit Precision und Recall immer 0 beträgt.

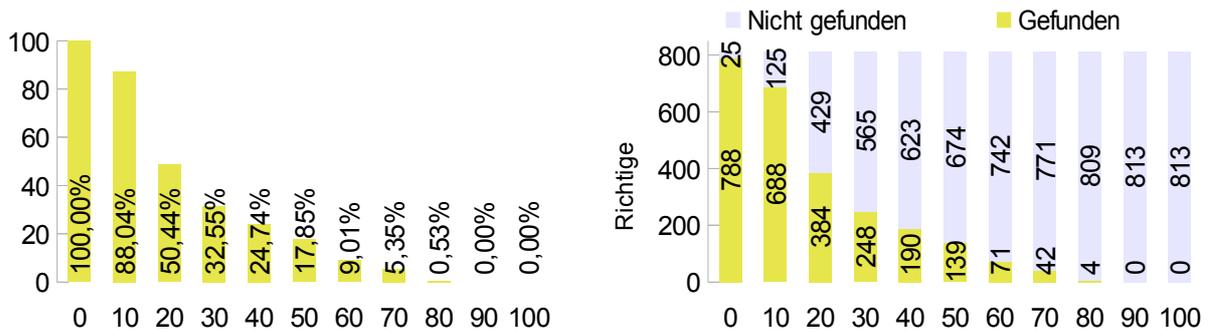


Abbildung 37: Namensähnlichkeit : Recall nur Namenssuche

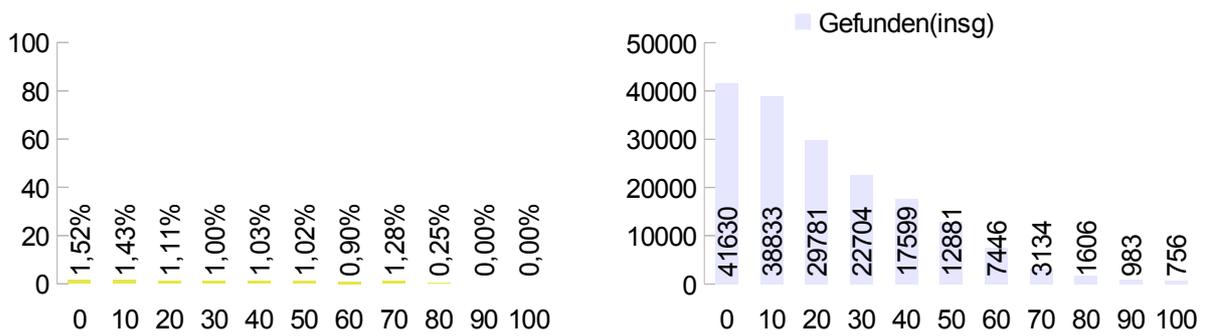


Abbildung 38: Namensähnlichkeit : Precision nur Namenssuche

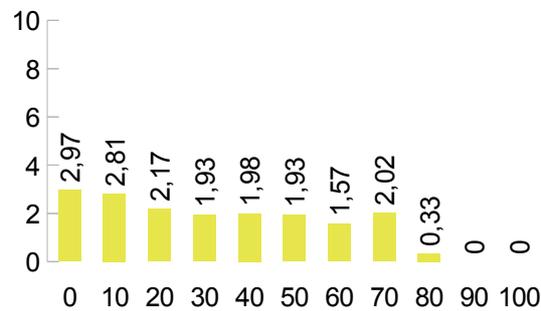


Abbildung 39: Namensähnlichkeit : F-measure \* 100 nur Namenssuche

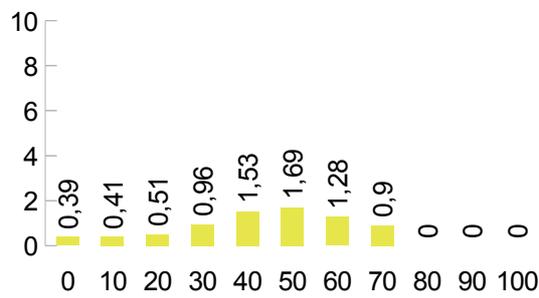
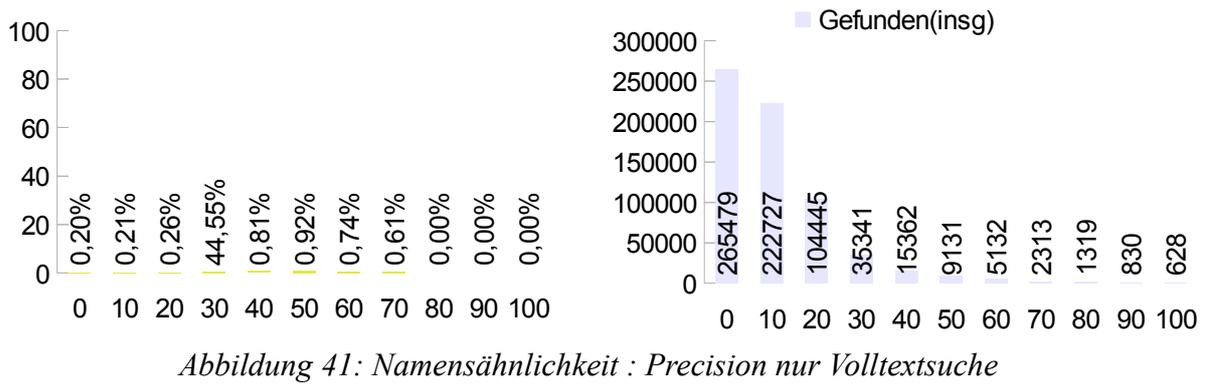
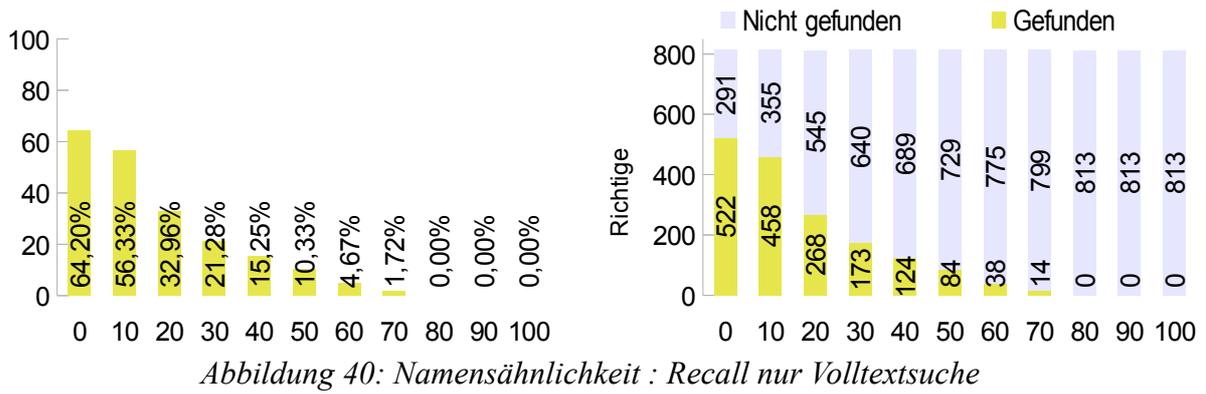


Abbildung 42: Namensähnlichkeit : F-measure \* 100 nur Volltextsuche

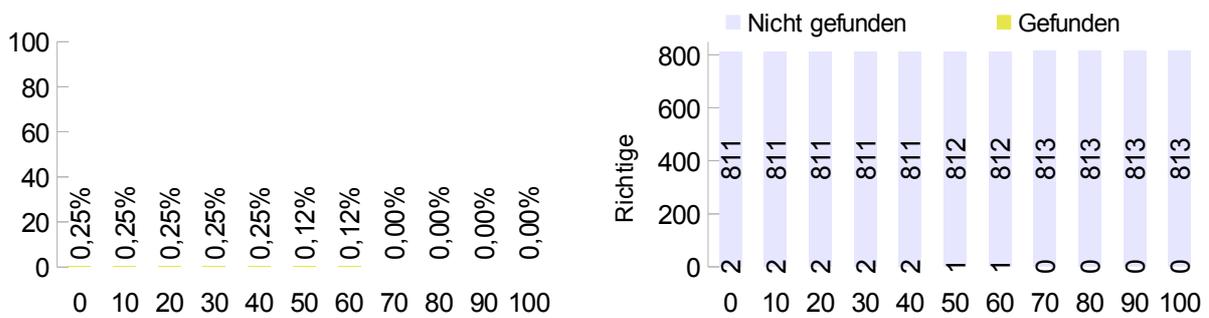


Abbildung 43: Namensähnlichkeit : Recall nur WordNet®-Suche

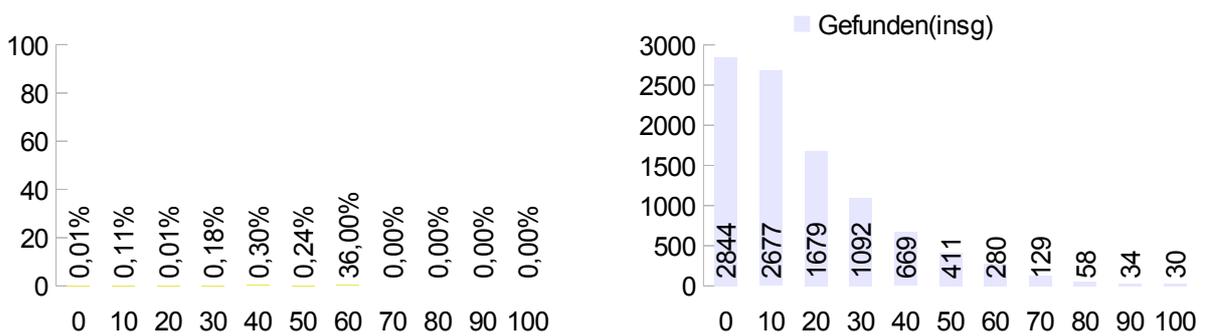


Abbildung 44: Namensähnlichkeit : Precision nur Namenssuche

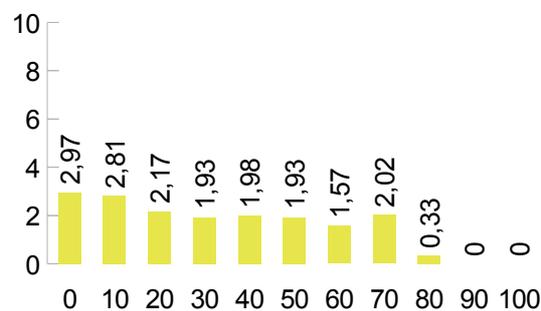


Abbildung 45: Namensähnlichkeit : F-messure \* 100 nur WordNet®-Suche

Die Ergebnisse sind sehr schlecht. Vor allem die Precision, die es kaum über 1% schafft, legt die Vermutung nahe, dass keine der Methoden überhaupt funktioniert.

Precision ist in diesem Fall das Verhältnis der als richtig bestätigten gefundenen Relationen zu den insgesamt gefundenen Relationen. Als richtig bestätigt können nur Relationen in RELcat gelten, also insgesamt 813. Wenn für jede Datenkategorie nur 1 Kandidat vorgeschlagen würde, könnten maximal 813 von 3,369 Datenkategorien bestätigt werden, was einer maximal möglichen Precision von 24,14% entspricht.

Diese 813 Relationen wurden dabei mithilfe von Fachwissen manuell – also nicht algorithmisch nachvollziehbar – erstellt und beziehen sich tendenziell auf Relationen, die Datenkategorien innerhalb eines Datenkategoriesets oder zwischen einzelnen wenigen Datenkategoriesets verbinden. Allein in dem zur Gold2010-Ontologie gehörenden Set<sup>77</sup> finden sich 493 Relationen, die Unterklassenbeziehungen zwischen Datenkategorien innerhalb des Datenkategoriesets Gold2010 beschreiben. Dies bedeutet, dass sich mehr als die Hälfte der in RELcat existierenden Relationen zum derzeitigen Zeitpunkt nur auf das Datenkategorieset Gold2010 beziehen.

Die Datengrundlage ist zum derzeitigen Zeitpunkt für die Berechnung der Precision von Verfahren, die unabhängig von der Zugehörigkeit von Datenkategorien zu ihren Sets arbeiten und ohne Fachwissen auskommen müssen, ungeeignet.

Ein Teil der Motivation dieser Arbeit besteht darin, genau diesen Umstand – der geringe Umfang von RELcat – zu beheben, weshalb es sinnvoll ist, zu untersuchen, wie sich mit einer sich ändernden Datengrundlage die Ergebnisse verändern würden, wenn die Verfahren angewendet werden.

Um dies zu testen, wurde die Datenmenge, die sonst ein Abbild der Datenmenge in RELcat ist, um die Kandidaten erweitert, die per Namenssuche mit einer Namensähnlichkeit >70% oder über die WordNet®-Suche als Kandidaten gefunden wurden. Diese Erweiterung der Datenbasis ist allerdings nur eine Notlösung – die Ergebnisse verbessern sich natürlich, wenn ein Teil der Ergebnisse in die Grundmenge mit aufgenommen wird – aber stellt im Moment die einzige Möglichkeit dar, den Einfluss des geringen Umfangs von RELcat zu testen.

70% wurde als Grenzwert gewählt, da bei diesem Wert ein (sehr kleines) lokales Maximum der Precision sichtbar ist. Die Kandidaten wurden nicht manuell gefiltert. Auf die Kandidaten der Beispiel-Suche wurde verzichtet, da diese überhaupt noch nicht evaluiert werden konnte. Auf die Ergebnisse der Volltextsuche wurde verzichtet, um diese Methode als Kontrollmethode einsetzen zu können und damit zu verhindern, dass bessere Ergebnisse ausschließlich daraus entstehen, dass einfach nur ein Teil der Ergebnisse in die Grundmenge mit aufgenommen wird. Sofern die einzelnen Methoden funktionieren wie sie sollen, müssten sich die Ergebnisse der Beispiel- und der Volltextsuche analog zu den Ergebnissen der Namens- und

---

<sup>77</sup> <http://lux13.mpi.nl/relcat/set/gold> oder [data/RELcat/gold.xml](http://lux13.mpi.nl/relcat/gold.xml)

WordNet®-Suche verbessern.

Die Grundmenge wurde dabei von 813 auf 6602 Relationen erweitert. In der Datei `results/tests/PrecRec/RELcatDataCategories_additionaldata.txt` befindet sich ein Abbild der so simulierten umfangreicheren RELcat.

Diese Auswertung wurde analog zu den Auswertungen ohne Erweiterung einmal für alle Methoden und anschließend nach Methoden aufgeschlüsselt durchgeführt. Die Skalen der Diagramme wurden auf 100 normiert.

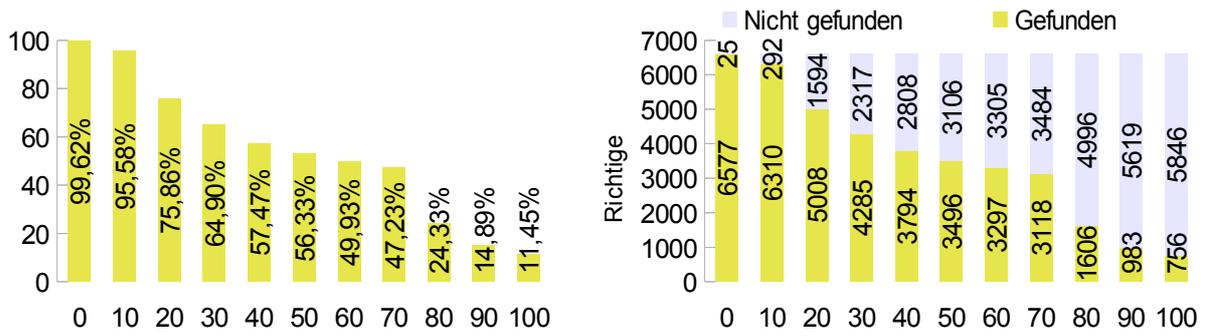


Abbildung 46: Namensähnlichkeit : Recall, Daten erweitert

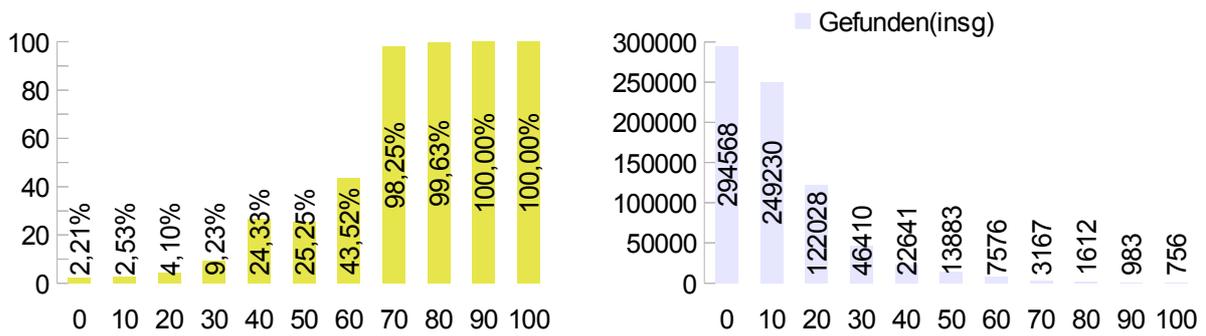


Abbildung 47: Namensähnlichkeit : Precision, Daten erweitert

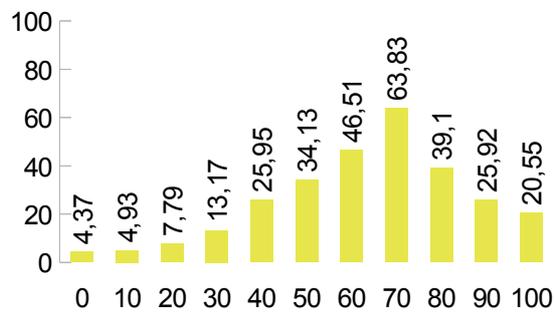


Abbildung 48: Namensähnlichkeit : F-measure \* 100, Daten erweitert

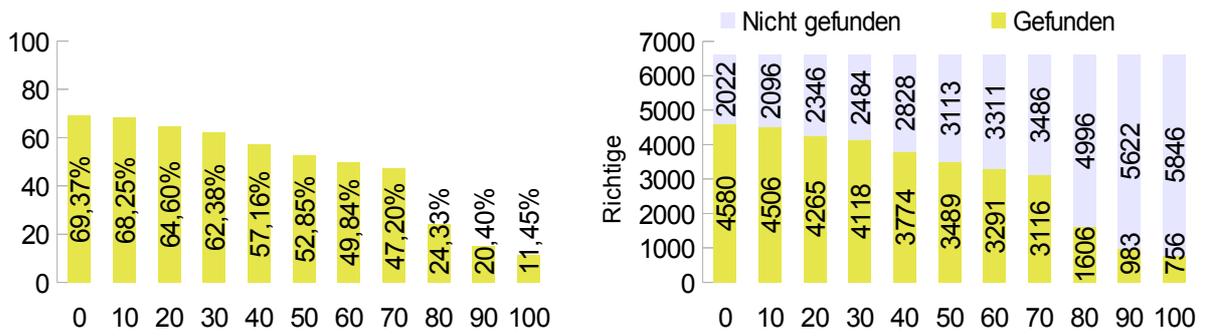


Abbildung 49: Namensähnlichkeit : Recall, Daten erweitert, nur Namenssuche

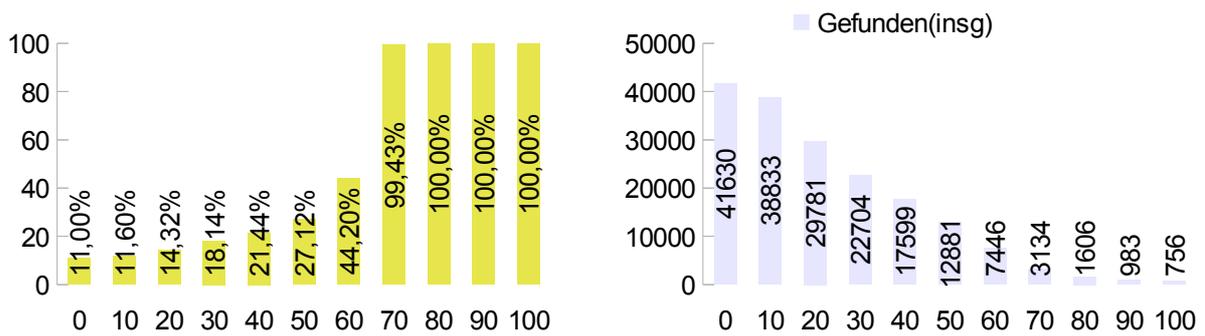


Abbildung 50: Namensähnlichkeit : Precision, Daten erweitert, nur Namenssuche

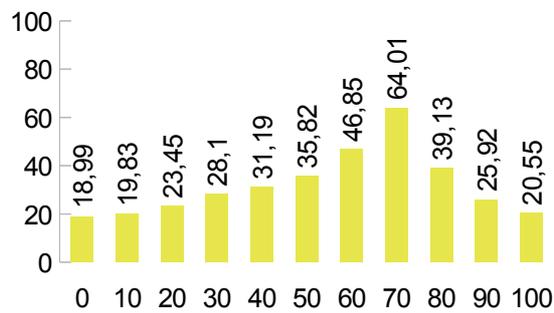


Abbildung 51: Namensähnlichkeit : F-measure \* 100, Daten erweitert, nur Namenssuche

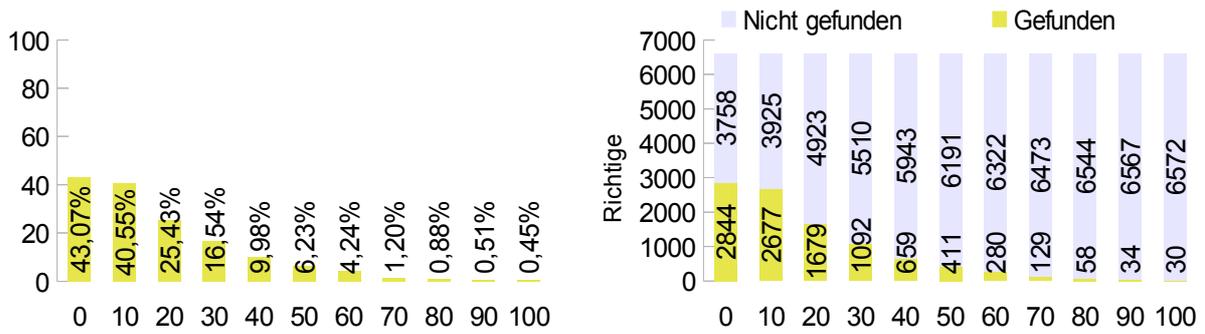


Abbildung 52: Namensähnlichkeit : Recall, Daten erweitert, nur WordNet®-Suche

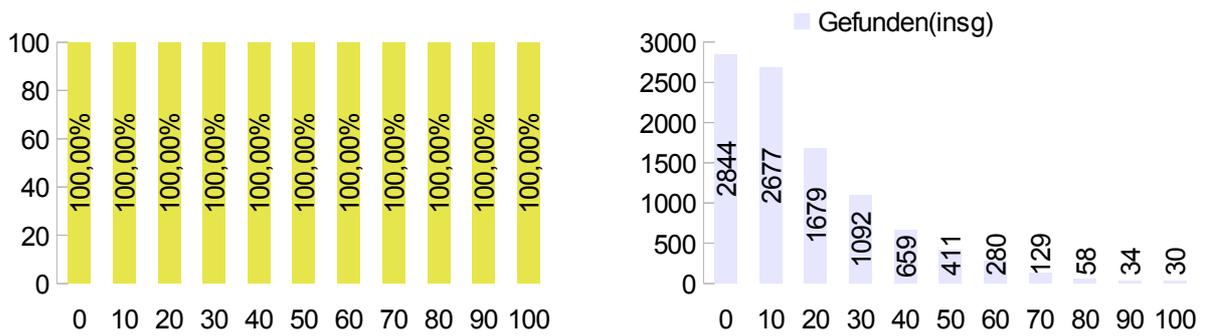


Abbildung 53: Namensähnlichkeit : Precision, Daten erweitert, nur WordNet®-Suche

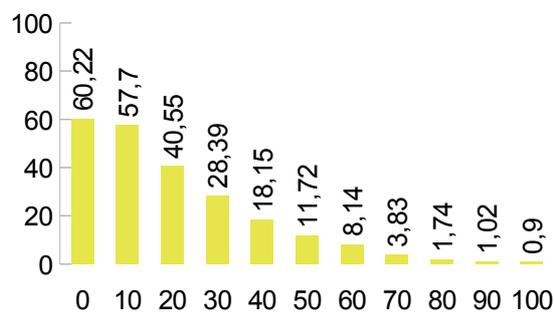
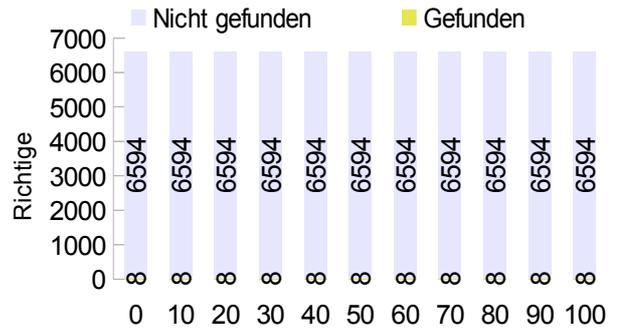
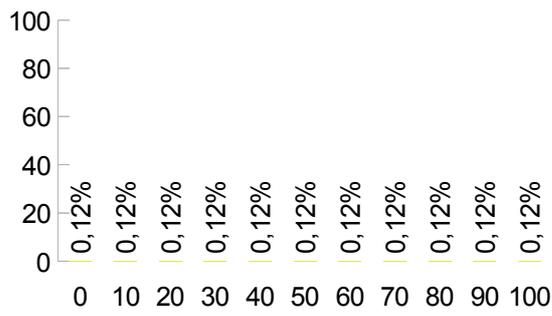


Abbildung 54: Namensähnlichkeit : F-measure \* 100, Daten erweitert, nur WordNet®-Suche



55: Namensähnlichkeit : Recall, Daten erweitert, nur Beispiel-Suche

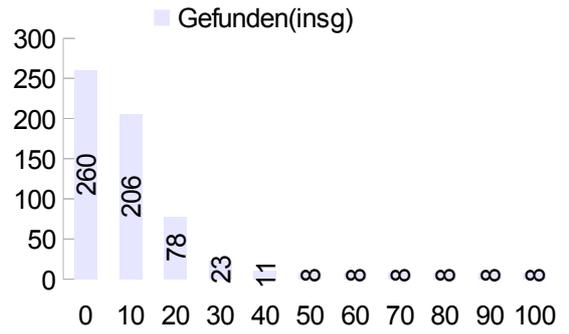
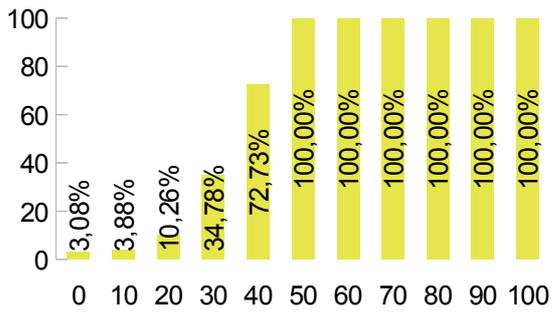


Abbildung 56: Namensähnlichkeit : Precision, Daten erweitert, nur Beispiel-Suche

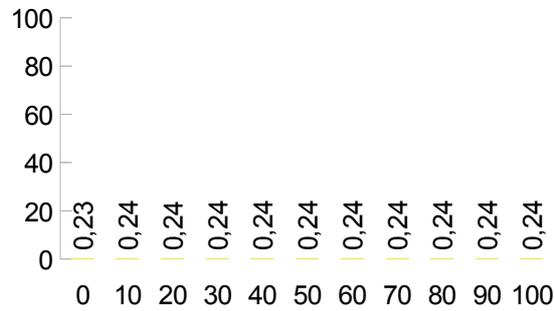


Abbildung 57: Namensähnlichkeit : F-measure \* 100, Daten erweitert, nur Beispiel-Suche

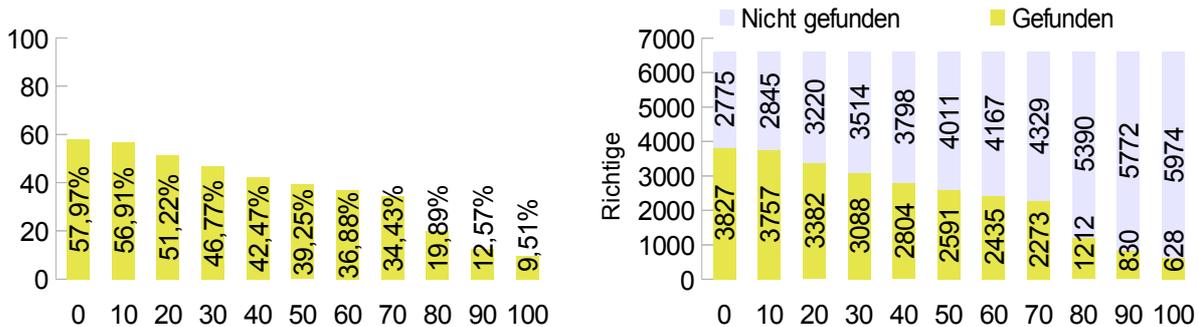


Abbildung 58: Namensähnlichkeit : Recall, Daten erweitert, nur Volltextsuche

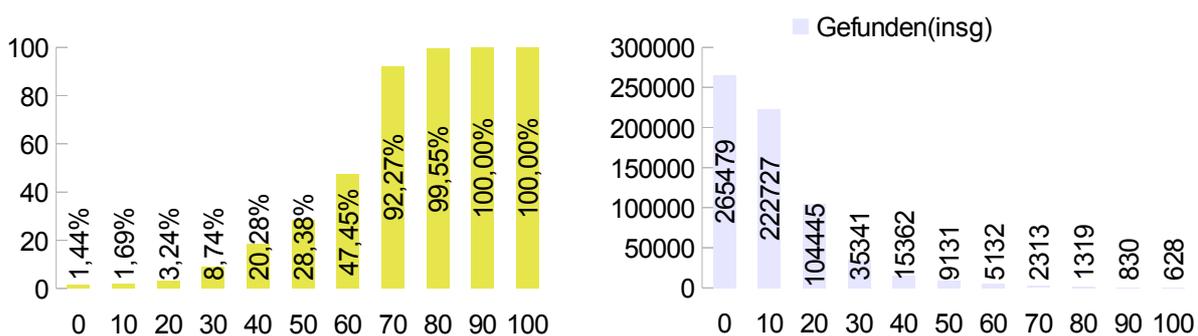


Abbildung 59: Namensähnlichkeit : Precision, Daten erweitert, nur Volltextsuche

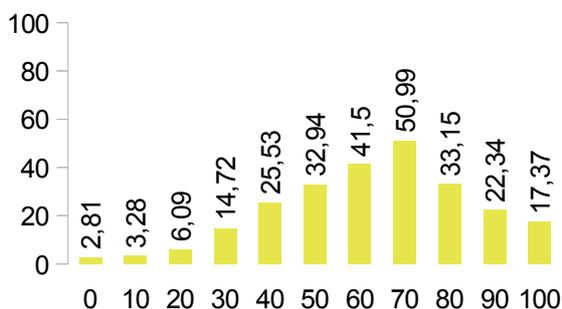


Abbildung 60: Namensähnlichkeit : F-measure \* 100, Daten erweitert, nur Volltextsuche

Wie sich herausstellt, haben sich Precision und Recall für die nicht zur Datenanreicherung verwendeten Verfahren ebenfalls verbessert. Der geringe Recall bei der Beispiel-Suche kann ebenfalls wieder mit der geringen Menge von als richtig bestätigten Relationen – in dem Fall 8 – begründet werden. Ein weiterer Testlauf mit einer weiter vergrößerten Datenbasis beziehungsweise mehr Datenkategorien, die für die Beispiel-Suche in Frage kommen, würde voraussichtlich auch hier wieder bessere Ergebnisse bringen. Die Verbesserung bei der Volltextsuche zeigt aber, dass die schlechte Precision durch die wenig umfangreiche RELcat verursacht wird –

während die Precision stark angestiegen ist, hat sich die Menge der gefundenen Kandidaten (verständlicherweise<sup>78</sup>) nicht geändert.

Bei der Erweiterung der Daten ist keine Filterung vorgenommen worden, die Kandidaten wurden also so mit aufgenommen, wie sie von den Suchmethoden zurückgeliefert wurden. Die Namens- und die WordNet®-Suche sind dabei recht verlässliche Suchmethoden, weshalb die so simulierte umfangreichere RELcat, wenigstens als Notlösung für die Testläufe verwendet werden kann. Da es sich aber um einen Test mit künstlich – und vor allem mit teils eigentlich zu testenden Methoden – aufgeblähten Daten handelt, ist es empfehlenswert, zu einem späteren Zeitpunkt eine erneute Evaluierung durchzuführen. Auch wenn es intuitiv klar ist, dass jede der Suchmethoden in der Lage ist, den Suchraum wenigstens stark einzuschränken, kann nicht zweifelsfrei nachgewiesen werden, dass eine rein zufällige Auswahl von Kandidaten schlechtere Ergebnisse liefern würde. Dafür bräuchte man eine umfangreichere sicher richtige Datenbasis.

---

78 Es wurde nichts an der Suche geändert, also wieder die gleichen Datenkategorien gefunden.

## 7 Vorschläge für weiterführende Arbeiten

Die graphische Benutzeroberfläche, die die Webservices nutzt, wurde nur rudimentär entwickelt, um die Funktionalitäten beziehungsweise die zugehörigen Aufrufe zu veranschaulichen. Diese Oberfläche ist noch sehr ausbaufähig, was aber nicht zu einer Masterarbeit passen würde, sondern sich eher als eine Aufgabe für ein Praktikum anbieten würde.

Aufgrund des Alpha-Status der RELcat ist nicht auszuschließen, dass die Funktionalität von RELcatComparator zukünftig überarbeitet werden muss. Durch die Abkoppelung in eine andere Klasse und der klaren Aufgabenteilung der Methoden dieser Klasse sollten diese Anpassungen bei Bedarf aber unkompliziert umsetzbar sein.

Die Informationen der RELcat werden für die Kandidatensuche nicht mit einbezogen. Denkbar wäre zum Beispiel, wie in Kapitel 5.2.3 erwähnt, transitive Kandidatenrelationen aus der RELcat mit in die Ergebnismenge aufzunehmen. Wenn also für DC-1 ein Kandidat DC-2 gefunden wird und in RELcat DC-2 rel:sameAs DC-55 steht, dann ist DC-55 auch ein möglicher Kandidat für DC-1.

Thematisch ähnlich, da es auch um eine Weiterverarbeitung der Kandidatenmenge ginge, wäre es, zu untersuchen, inwiefern sich aus den Namen von Kandidat und Vorlage weitere Schlüsse ziehen lassen. In Kapitel 5.2 sind hierfür schon eine Idee dargestellt, die in der Umsetzung der Weboberfläche eingeflossen sind, nämlich rel:sameAs bei Namensähnlichkeit 1.

Auch die Nutzung anderer manuell erstellter Wissensdatenbanken wie beispielsweise Thesauri wäre sinnvoll.

Weiter differenzierte Suchmethoden könnten Datenkategorie-spezifische Einträge finden – beispielsweise kann Named Entity Recognition, also Namenserkennung, an die Datenkategorie DC-4115 oder Datumsangaben an DC-2510 gekoppelt werden.

In Kapitel 6 wird eine mögliche Gewichtung der Kandidaten anhand der zugrundeliegenden Suchmethode angesprochen. Hierfür müssten die Suchmethoden genauer untersucht werden, um damit einen zu der Methode gehörigen Gewichtungsfaktor zu bestimmen.

## 8 Zusammenfassung

Insgesamt wurde in dieser Arbeit ein funktionierendes System aufgebaut, welches die Arbeit mit ISOcat und RELcat stark vereinfachen kann. Die Weboberfläche, welche eigentlich nur zu Testzwecken erstellt wurde, ist bereits produktiv einsetzbar, um Daten für RELcat zu generieren. Der praktische Nutzen dieser Weboberfläche ist dabei nicht zu unterschätzen, insbesondere, da es vorher keine Ansätze gab, die diese Funktionalitäten umsetzen, die Kandidatenmenge also nicht eingeschränkt werden konnte.

Weiter sind die Suchmethoden gut dafür geeignet, um innerhalb der RELcat nach Doppelgängern zu suchen. Die Methoden sind in sich abgeschlossen, wodurch sie problemlos in das bereits für ISOcat erstellte Webinterface eingebunden werden könnten.

Eine Evaluation der Suchmethoden war mit den gegebenen Daten nicht aussagekräftig durchführbar und sollte später wiederholt werden. Durch die Nutzung einer simulierten umfangreicheren RELcat konnte aber schon ein Ausblick gegeben werden, wie die Ergebnisse sich verändern können, wenn RELcat weiter ausgebaut wird. Dieser Ausblick wurde allerdings mit stark geschönten Daten durchgeführt und sollte daher skeptisch betrachtet werden.

Der Massenimport von Daten kann – sofern er auf dieser Ebene überhaupt gewünscht ist – ebenfalls die Funktionalität des Webinterface mit einfachen Mitteln stark erweitern.

Die Menge der Kandidaten, die für eine Relation für eine Datenkategorie in Frage kommen, wurde von anfangs 3368 auf durchschnittlich 264, beziehungsweise 65 ohne Volltextsuche reduziert. Die in RELcat bereits gefundenen Relationen werden dabei mit einer hohen Verlässlichkeit (788/813) abgedeckt. Allerdings muss man sich hier auch vor Augen führen, dass bei der Volltextsuche Datenkategorien von Gold2010 sich gegenseitig favorisieren, da sie eine gemeinsame Quelle als signifikanten Term haben. Diese machen wiederum knapp 50% von RELcat aus.

Aber selbst wenn man die Volltextbeschreibungen ignoriert, werden noch 631/813 – also mehr als  $\frac{3}{4}$  aller – Relationen erfolgreich vorhergesagt. Es ist absehbar, dass die Ergebnisse sich mit zunehmender Qualität der Beschreibungen der

Datenkategorien verbessern werden.

Die Suche per WordNet® liefert generell die besten Ergebnisse, da diese auf vorher manuell angefertigtem Wissen basieren, was sogar ein Raten der zugehörigen Relation sinnvoll werden lässt.

Die Namenssuche liefert ebenfalls sehr verlässliche Ergebnisse, ein Raten der Relation wurde in dieser Arbeit, abgesehen von der in Kapitel 5.2 beschriebenen sameAs-Regel, aber noch nicht umgesetzt.

Die Suche anhand der Beschreibungstexte liefert Kandidaten, bei denen es überhaupt nicht mehr möglich ist eine Relation zu raten. Selbst die Frage, ob es sich um einen nützlichen Kandidaten handelt, lässt sich nicht mit Gewissheit beantworten, da eine Begriffsüberschneidung auch auftreten kann, wenn in mehreren Datenkategorien eine wissenschaftliche Quelle als signifikant auftretender Term vorkommt. Dies führt zu einem starken Noise in der Ergebnismenge, der durch eine hohe Mindestsignifikanz in der Terminologie-Extraktion bereits reduziert wurde. Diese Methode sollte also eher dazu eingesetzt werden, Füllkandidaten zu suchen, die, wenn keine der anderen Methoden Ergebnisse liefert, „wenigstens irgendetwas mit der Vorlage zu tun haben“.

Eine Sonderstellung nimmt die Suche anhand der Beispiele ein. Diese Methode, obwohl sie auf einer sehr abstrakten Datenaufbereitung basiert, liefert erstaunlich gute Ergebnisse und das Toleranzintervall der Beispiele spiegelte in praktischen Versuchen die Erweiterung der Toleranz bei den gefundenen Kandidaten gut wieder. Als Beispiel verweise ich auf die Beispielrechnung in Kapitel 4.2.3. Hierbei handelt es sich allerdings um eine rein subjektive Beobachtung, bei der eine objektive Auswertung noch aussteht. Diese ist zum Zeitpunkt dieser Arbeit noch nicht sinnvoll, da die Datenmenge im Moment nicht ausreicht und sollte zu einem späteren Zeitpunkt durchgeführt werden.

## 9 Literaturverzeichnis

- 1: <http://de.clarin.eu> (Stand 04.2013).
- 2: <http://www.isocat.org/index.html> (Stand 04.2013).
- 3: [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=37243](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37243) (Stand 04.2013).
- 4: [https://catalog.clarin.eu/isocat/12620/model/DCR\\_data\\_model.pdf](https://catalog.clarin.eu/isocat/12620/model/DCR_data_model.pdf) (Stand 04.2013).
- 5: <http://lux13.mpi.nl/relcat/site/index.html> (Stand 04.2013).
- 6: M.A. Windhouwer. RELcat: a Relation Registry for ISOcat data categories. In European Language Resources Association (ELRA) (ed), Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC 2012), Istanbul, Turkey, May 23-25, 2012.
- 7: <http://www.isocat.org/forum/viewtopic.php?f=3&t=14&p=42> (Stand 04.2013).
- 8: <http://wortschatz.uni-leipzig.de/~cbiemann/software/toolbox/TE.html> (Stand 04.2013).
- 9: <http://tartarus.org/martin/PorterStemmer/index.html> (Stand 04.2013).
- 10: M.F. Porter: An algorithm for suffix stripping. In: Program, 14(3), S. 130-137, Juli 1980.
- 11: <http://de.wikipedia.org/wiki/Levenshtein-Distanz> (Stand 04.2013).
- 12: <http://www.w3.org/TR/2004/REC-owl-features-20040210/> (Stand 04.2013).
- 13: David Pansch. Verfahren zur Datenintegration strukturell heterogener Quellen im Kontext der eHumanities, Leipzig, Germany, September 2010.
- 14: <http://wordnet.princeton.edu/wordnet/> (Stand 04.2013).
- 15: <http://www.isocat.org/12620/examples/dcif-example.dcif> (Stand 04.2013).
- 16: Menzo Windhouwer, Sue Ellen Wright. Linking to linguistic data categories in ISOcat. In C. Chiarcos, S. Nordhoff and S. Hellmann (eds), *Linked Data in Linguistics - Representing and Connecting Language Data and Language Metadata* (LDL 2012), © Springer-Verlag, pp 99-107, Frankfurt/Main, Germany, March 7-9, 2012.

## 10 Anhang

### A Installation

#### RDF2Dcif

JAVA 1.6 wird vorausgesetzt.

RDF2Dcif benötigt eine Datei mappings.properties im Arbeitsverzeichnis, die zum Konfigurieren des Mappings genutzt werden sollte.

Das Programm kann über Kommandozeile oder über den Codebeispiele in Kapitel 3.3 in einem eigenen Programm gestartet werden.

Beim Aufruf über die Kommandozeile wird der Ort der Datei, die als Vorlage dienen soll in Form einer Url, also inklusive Protokoll, angegeben.

Beispielaufrufe:

```
java -jar RDF2Dcif http://www.lingvoj.org/lingvoj.rdf
java -jar RDF2Dcif file:/// [Pfad zu Daten]/lingvoj.rdf
```

#### RELcatCandidateFinder

JAVA 1.6 wird vorausgesetzt.

Auf der CD befinden sich ausführbare jar-Dateien für die Updatemethoden, jeden Test, jede Suchmethode und weitere ausgewählte Methoden. Diese funktionieren ebenfalls als JAVA-Bibliotheken.

Für die Aufbereitung der Daten ist eine MySQL-Datenbank mit Lese- und Schreibrechten und ein Internetzugang nötig.

#### Datenbank

Die Daten für die Verbindung können in der Datei conf.properties angepasst werden und es muss entweder eine Datenbank mit dem in dieser Datei angegebenen Namen

für den angegebenen Nutzer angelegt werden oder der angegebene Nutzer über root-Rechte verfügen, sodass die Datenbanken angelegt werden können.

### **Erzeugte Tabellen**

Die Methode UpdateHandler → updateISOcatMirror() bildet die in ISOcat enthaltenen Informationen auf folgende Tabellen ab:

Name	Zweck	Primärschlüssel
pid_descr	unbearbeitete Volltextbeschreibung für prakt. Verwendung	pid
pid_example	Auflistung aller <dcif:example> für jede Datenkategorie	hash(pid+example)
pid_example_stats	Durchschnittlicher relativer Anteil von Zeichenart an allen Beispielen	pid
dcs_latestchange	Letztes Änderungsdatum des Datenkategoriesets	Name des Datenkategoriesets
pid_name_dcs	Name und Datenkategorieset zu jeder pid	pid
pid_source	(Otionale) Quellen für die Datenkategorien.	Auto-inc. PK

In der Datenbank werden noch eine Vielzahl anderer Tabellen angelegt. Diese sind nach den darin abgebildeten Datenkategoriesets benannt und enthalten die den jeweiligen Datenkategorie-Pids entsprechenden aus den Beschreibungstexten extrahierten und auf ihren Porter-Stamm reduzierten Begriffe.

### **Vorbereiten der Datenbank**

Ist die Datenbankverbindung korrekt konfiguriert, genügt es, die Methoden UpdateHandler → updateISOcatMirror() und anschließend UpdateHandler → updateExampleStats() aufzurufen, die anschließend die Datenbank aufbauen. Solange diese Methoden laufen, wird eine Datei 'Running' im Arbeitsverzeichnis

abgelegt, die verhindert, dass diese Methode erneut aufgerufen wird. Sofern das Update planmäßig beendet wurde, wird diese Datei gelöscht, andernfalls muss sie manuell gelöscht werden.

Die Daten werden aus dem öffentlichen Arbeitsplatz von ISOcat extrahiert. Dabei werden immer die kompletten Datenkategoriesets nachgebaut und anschließend, vorausgesetzt es existieren darin vorher unbekannte Datenkategorien, eine Datenbank angelegt und entsprechend gefüllt.

Dabei ist es mehrmals vorgekommen, dass in der Datenbank unbekannte Datenkategorien in RELcat referenziert wurden. Weitere Recherchen ergaben, dass einige Datenkategorien nur in den Dateien der profile-Urls des öffentlichen Arbeitsplatzes enthalten sind. Deshalb werden diese, nachdem alle dcs-Urls abgearbeitet wurden, analog nachgebildet<sup>79</sup>. Die Profile enthalten 1288 Datenkategorien, die in keinem anderen Datenkategorieset aufgelistet sind. Beispiele hierfür sind DC-100, DC-124 und DC-4363. Eine vollständige Liste befindet sich in Form eine csv-Datei auf der CD<sup>80</sup>.

Die Dauer des Updates der Datenbank wird dadurch verdoppelt.<sup>81</sup>

Psceudocode:

```
hole alle Datenkategoriesets als dcs aus guest-workplace
für jede dcs
{
    baue dcs nach
    reduziere dcs
    leere tabelle mit namen der dcs.
    fülle tabelle mit namen dcs mit dcs
}

hole alle Profile als profiles aus guest-workplace
für jedes profile
{
    baue dcs nach
    reduziere dcs
    leere tabelle mit namen des Profils
    fülle tabelle mit namen dcs mit dcs
}
```

---

79 Durch diese Reihenfolge wird sichergestellt, dass Datenkategorien bevorzugt in den Tabellen der Datenkategoriesets abgelegt werden.

80 data/misc/dc\_in\_profile.csv oder SELECT pid, name, dcs FROM `pid\_name\_dcs` WHERE dcs LIKE "profile%"

81 Siehe CD results/tests/logs/Log.txt

Das Update dauert mehr als 5 Stunden auf einem Laptop HP-625 mit 4GB Arbeitsspeicher, AMD V160 Prozessor (2,40GHz) unter Windows 7 64Bit.

Während des Updates wird eine Log-Datei Log.txt im Arbeitsverzeichnis erstellt.

## **Onlinedienste**

Die Webservices liegen dieser Arbeit sowohl in Form einer war-Datei als auch in Form von Quellcode bei.

Um diese Services nutzen zu können wird ein Server vorausgesetzt, der war-Dateien deployen kann. Entwickelt wurden die Webservices für Apache Tomcat<sup>82</sup> 6.

Die graphische Nutzeroberfläche wurde so ausgelegt, dass sie hauptsächlich mithilfe der Webservices arbeitet. Deshalb müssen diese nach dem Deployen unter der in der Konfigurationsdatei 'conf.properties' angegebenen URL erreichbar sein.

---

<sup>82</sup> <http://tomcat.apache.org/>

## B Paketstruktur

Der Programmcode ist auf 6 Pakete unterteilt. Im folgenden werde ich auf die wichtigsten Aspekte der einzelnen Klassen eingehen, wobei ich mich aber auf Wesentliches reduzieren werde. Genauere Informationen finden sich in dieser Arbeit beiliegenden JavaDoc beziehungsweise im Quellcode selbst.

In der Implementierung der Terminologie-Extraktion der ASV-Toolbox fanden sich zwei Probleme, die eine Änderung des Quellcodes erforderten, weshalb die Pakete `de.uni-leipzig.asv.toolbox.te`, `de.uni-leipzig.asv.toolbox.viterbitagger`, `de.uni-leipzig.asv.toolbox.util` und `de.uni-leipzig.asv.toolbox.utils` in den Quellcode des Programms mit aufgenommen wurden.

- Es wurden sämtliche Thread-Mechanismen entfernt, da diese dazu führten, dass es schwierig wurde diesen Code ohne Thread-Handling auszuführen. Die Dokumentation zur Toolbox sagt dazu, dass diese Threads implementiert wurden, um die Terminologie-Extraktion bei Bedarf beenden zu können – „It is used to stop the process of extraction if it is needed[8].
- Es fanden sich 2 ungeschlossene RandomAccess-Files im Paket `de.uni-leipzig.asv.toolbox.viterbitagger`, die von mir geschlossen wurden. Diese Änderungen betreffen die Klassen `Lexicon_extern` (Zeile 130) sowie `Transitions_extern` (Zeile 81). Dieses Problem erwies sich als schwerwiegend und sorgte bei der Nutzung als Webservice über einen Server nach einer Weile zum Absturz desselben aufgrund von zu vielen geöffneten Dateien.

```
try {
    this. dataFile = new RandomAccessFile(filename
        + Transitions_extern. binStr, "r");
    this. SMOOTHVAL = this. dataFile. readDouble();
    this. dataOffset = this. dataFile. getFilePointer();
    this. dataFile. close();//<<-----***Hier wurde geändert**
} catch (Exception e) {
    System. err. println("File IO problem with1 "
        + (filename + Transitions_extern. binStr) +
        " : " + e. GetMessage());}
```

## jt.msc.access

Die Klassen dieses Paketes stellen die Schnittpunkte zur MySQL-Datenbank sowie zu den Daten von WordNet® bereit. Die Methoden von *DBAccess* sollten dabei selbsterklärend sein und dienen im wesentlichen dazu, den Quellcode übersichtlich zu halten. Die Informationen zum Datenbankzugang werden aus *conf.properties* entnommen.

*WordNetAccess* dient als Schnittstelle zu der WordNet®-Datenbank, wobei hier bereits von RELcat-Relationen auf WordNet®-Relationen gemappt wird.

Dabei gilt immer

Subjekt – Prädikat - Objekt

Vorlage - Relation - Kandidat

RELcat	WordNet®
rel:sameAs	Synonymy
rel:superClass	Instance-Hypernymy Hypernymy
rel:subClass	Instance-Hyponymy Hyponymy
rel:hasPart	Member - Holonymy
rel:partOf	Member - Meronymy
rel:hasDirectPart	Substance-Holonymy Part - Holonymy
rel:directPartOf	Substance-Meronymy Part - Meronymy

## jt.msc.datamodel

Die Klassen im *datamodel* bilden das Datenmodell von ISOcat, also *DataCategory* und *DataCategorySet*, auf JAVA-Objekte ab. *Candidate* erweitert *DataCategory* um eine vorgeschlagene Relation, die als „rel:related“ initialisiert wird, da diese Relation grundsätzlich für gefundene Kandidaten angenommen werden kann.

## jt.msc.options

Die hier enthaltene Klasse *Options* liefert die Parameter, die in der conf. properties festgelegt wurden und initialisiert die Variablen für die Aufbereitung der Begriffsdatenbank. Letztere sollten nur mit Vorsicht geändert werden, da sie ein vollständiges Datenbankupdate auf einer zurückgesetzten Datenbank erfordern um den Datenbestand und die für Anfragen genutzten Informationen zu synchronisieren. Options.init() sollte vor Verwendung der anderen Funktionen aufgerufen werden.

Parameter, die mit „te\_“ beginnen, betreffen dabei die Terminologie-Extraktion der ASV-Toolbox.

<b>Parameter = Default</b>	<b>Erklärung</b>
porter = true	Extrahierte Begriffe auf Porter-Stamm reduzieren ? [true, false]
te = true	Begriffe extrahieren oder Volltext ? [true, false]
blacklist = true	Ignorieren von Begriffe der Blacklist ? [true, false]
exampleStatsVarianz = 0. 01	Höchstvarianz für die Beispiel-Suche Siehe 4.2.3c 0 → keine Abweichung 1 → Jedes Beispiel gilt
te_lang = en	Sprache des Textes und Vergleichskorpus
te_minTextFreq = 1	Mindestfrequenz eines Begriffes im Text [1...]
te_minCorpFreq = 1	Mindestfrequenz eines Begriffes im Vergleichskorpus [1...]
te_minPhraseFreq = 100	Mindestfrequenz einer Phrase im Text, Phrasen sind hier unerwünscht, deshalb 100 [1...]
te_minSig = 30	Mindestsignifikanz, mit der der Begriff auftritt [0...100]
te_sigFormular = Likelihood	Verwendetes Signifikanzformular . [Parameters. HQ oder Parameters. LR}

## jt.msc.test

Dieses Paket enthält die Tests zur Evaluierung.

Die Tests sind dabei im Konstruktor abgeschlossen, und liefern die Ergebnisdateien im Arbeitsverzeichnis. Weitere Informationen finden sich in den jeweiligen Quellcodedateien.

## jt.msc.tools

Dieses Paket enthält eine Sammlung von Klassen, die verschiedene genutzte Werkzeuge darstellen. Es handelt es sich dabei um Helferklassen (*Webseite*, *Logger*) sowie um Programmcode, der aus Fremdquellen übernommen wurde (*Porter*<sup>83</sup>, *LevenshteinDistance*<sup>84</sup>).

## jt.msc.worker

In jt.msc.worker finden sich die Klassen, die die eigentliche Arbeit, in Bezug auf die Datenaufbereitung (worker.databaseupdater), das Vergleichen (worker.comparator) und sonstige Aufgaben (worker.misc), übernehmen.

## jt.msc.worker.misc

*TermExtractor* entspricht bis auf die in oben erwähnten Änderungen und der Initialisierung mit den Parametern der Terminologie-Extraktion der Beispielimplementierung für die Term-Extraktion der ASV-Toolbox.

Folgender Quellcode erzeugt einen java.util.StringTokenizer, der die extrahierten Begriffe aus dem String 'text' enthält. Die Initialisierung der Parameter für die Terminologie-Extraktion wird dabei über den Konstruktor durchgeführt, sodass bei gleichen Parametern der *Termextraktor* wiederverwendet werden kann.

---

83 Verwendete Implementierung <http://alvinalexander.com/java/jwarehouse/lucene-1.3-final/src/java/org/apache/lucene/analysis/PorterStemmer.java.shtml>

84 Verwendete Implementierung : <http://mrfoo.de/archiv/1176-Levenshtein-Distance-in-Java.html>

```

TermExtractor te = new TermExtractor();
te.extract(text);
java.util.StringTokenizer terms = te.terms;

```

*Norm* enthält diverse Methoden, die sowohl für die Datenaufbereitung als auch für die Aufbereitung der Datenkategorie, die als Vorlage dienen soll, nützlich sind. Die Methoden wurden ausgelagert, um sicherzustellen, dass in beiden Fällen auf die gleichen Arbeitsabläufe zurückgegriffen wird. Sollten Änderungen an der Datenaufbereitung nötig sein, ist es empfehlenswert, diese hier umzusetzen.

Weiter kann in dieser Klasse mithilfe der Methode `toTrig(String pid, HashMap<String, Candidate> candidates)` eine übergebene Kandidatenliste in ein trig-Dokument geschrieben werden.

Diese Klasse dient außerdem dazu, eine erzeugte *DataCategory* auf die für die Kandidatensuche nötige Form zu normieren.

```

DataCategory cat = new DataCategory();
cat.name = "Adjective";
cat.definition = "A word that describes a noun";
cat = Norm.reduceDC(cat);

```

cat kann jetzt für die Suche verwendet werden.

*DCSFactory* dient zur Konstruktion der *DataCategorySelections* und baut diese mittels des DOM-Dokumentes, welches aus der übergebenen Webseite aufgebaut wird, nach. Die Methoden sind hier den jeweiligen Section-Namen der dcif-Dokumente nachempfunden und arbeiten diese sequentiell ab.

Folgender Quellcode erzeugt ein zur weiteren Verarbeitung fertiges Datenkategorieset

```

DataCategorySelection dcs = DCSFactory.build(new Webseite(url));

```

wobei 'url' die Url zum dcif-Dokument<sup>85</sup> des Datenkategoriesets ist. Soll die Datenkategorie aus einer lokalen Datei aufgebaut werden, muss die Url entsprechend angegeben werden.<sup>86</sup>

*RELcatCollector* erzeugt eine Datei, in der der derzeitige Stand der RELcat abgebildet wird. Diese kann für Tests genutzt werden, die mit RELcatdaten arbeiten.

---

<sup>85</sup> Beispielsweise <https://catalog.clarin.eu/isocat/rest/dcs/143dcif?dcif-mode=all>

<sup>86</sup> Siehe [http://en.wikipedia.org/wiki/File\\_URI\\_scheme](http://en.wikipedia.org/wiki/File_URI_scheme)

## jt.msc.worker.comparator

Hier befinden sich die Klassen, die für das Suchen von Kandidaten verantwortlich sind. Das Grundprinzip für die Arbeit mit diesen Klassen liegt darin, für jede Vorlage einen neuen Comparator zu erstellen, der dann verschiedene Suchmethoden anbietet.

*RELcatComparator* sucht die zu einer Datenkategorie passenden RELcat-Einträge. Dies funktioniert nicht für Datenkategorien, für die in RELcat keine Relationen vorhanden sind, weshalb diese Klasse sich nicht für den Einsatz mit neuen, künstlich erzeugten Datenkategorien eignet.

Folgender Code sucht alle Einträge für DC-3066

```
DataCategory cat = new DataCategory();
cat.pid="DC-3066";
RELcatComparator co = new RELcatComparator(cat);

//Welche Relationen gibt es zu dieser DC?
Iterator<String> relation = co.findRels().keySet().iterator();

//Suche für jede Relation (bsp rel:sameAs) die Objekte
while(relation.hasNext())
{
    co. searchRelations(relation.next());
}
HashMap<String, ArrayList<String>> rel_dc = co. getRelMap();
```

'rel\_dc' enthält nun die für 'cat' als Objekte gefundenen Datenkategorien, geordnet nach der jeweiligen Relation und kann folgendermaßen durchiteriert werden

```
Iterator<String> rels = rel_dc.keySet().iterator();
while(rels.hasNext())
{
    String rel = rels.next();
    Iterator<String> cats = rel_dc.get(rel).keySet().iterator();
    while(cats.hasNext())
        System.out.println(cat.pid+" "+rel+" "+cats.next());
}
```

Ergebnis:

```
DC-3066 rel:subClassOf http://www.isocat.org/datcat/DC-3415
DC-3066 rel:sameAs http://purl.org/linguistics/gold/Adjectival
```

Die SPARQL-Abfragen werden über den Webservice der RELcat abgegeben und benötigen die Angabe der Sets, in denen gesucht werden soll. Um diese leicht

ändern zu können wurden sie in einen String ausgelagert<sup>87</sup>, der bei Änderungen oder Erweiterungen angepasst werden müsste.

Im Gegensatz zum *RELcatComparator* eignet sich der *DCCComparator* auch für den Einsatz mit künstlich erzeugten Datenkategorien. Das Grundprinzip ist auch hier, dass für jede Datenkategorie, die als Vorlage dient, ein Comparator angelegt wird, der dann verschiedene Suchmethoden zur Verfügung stellt. Die Kandidatenmenge wird dabei bei jeder Suche zurückgesetzt.

Die Suchmethoden versuchen nicht vorher festgelegte Informationen, beispielsweise eine fehlende Definition für die Termanalyse oder ein fehlender Name, anhand der PID der übergebenen Datenkategorie mit in der Datenbank enthaltenen Informationen zu füllen. Wird eine Datenkategorie mit einer in ISOcat vorhandenen, aber nicht zu der Datenkategorie passenden, PID und für die jeweilige Methode fehlenden Informationen übergeben, können deshalb falsche Ergebnisse folgen.

Folgender Code erzeugt eine Datenkategorie mit den für die jeweilige Methode nötigen Informationen, sucht nach Kandidaten und gibt die gefundenen Datenkategorien aus:

WordNet®-Suche

```
Options.init();
DataCategory cat = new DataCategory();
cat.name = "verb";
cat = new Norm().reduceDC(cat);
DCCComparator co = new DCCComparator(cat);
co.findWordnetCandidates();
Iterator<Candidate> candidates =
co.getCandidates().iterator();
while(candidates.hasNext())
{
    Candidate c = candidates.next();
    System.out.println(c.pid + "->" + c.name + "\t" + c.prefRel);
}

DC-1596->past participle adjective    rel:subClassOf
DC-2935->present participle          rel:subClassOf
DC-1597->present participle adjective rel:subClassOf
```

---

87 RELcatComparator → sets

## Namenssuche

```
Options.init();
DataCategory cat = new DataCategory();
cat.name = "Plural Noun";
cat = new Norm().reduceDC(cat);
DCComparator co = new DCComparator(cat);
co.findNameCandidates();
Iterator<Candidate> candidates =
co.getCandidates().iterator();
while(candidates.hasNext())
{
    Candidate c = candidates.next();
    System.out.println(c.pid + "->" + c.name + "\t" + c.prefRel);
}

DC-2704->noun          rel:related
DC-3348->noun classifier rel:related
DC-3349->noun phrase   rel:related
DC-2289->noun chunk    rel:related
DC-2253->voice noun    rel:related
DC-2218->broken plural rel:related
DC-1463->personal pronoun rel:related
...88
```

## Beispiel-Suche

```
Options.init();
DataCategory cat = new DataCategory();
cat.example.add("tree");
cat.example.add("house");
cat = new Norm().reduceDC(cat);
DCComparator co = new DCComparator(cat);
co.findExampleCandidates();
Iterator<Candidate> candidates =
co.getCandidates().iterator();
while(candidates.hasNext())
{
    Candidate c = candidates.next();
    c.name = DBAccess.getDcName(c.pid); //Name für Suche
                                        //irrelevant,
                                        //deswegen noch
                                        //unbekannt
    System.out.println(c.pid + "->" + c.name + "\t" + c.prefRel);
}

DC-1387->singular          rel:related
DC-2662->not a natural format rel:related
DC-253->plural            rel:related
DC-255->mass noun         rel:related
DC-2465->delivery format  rel:related
DC-2277->mass noun        rel:related
DC-1370->pronoun          rel:related
...89
```

---

88 Insgesamt wurden 255 / 3368 gefunden. Siehe CD results/candidates/name\_plural\_noun.txt

89 Insgesamt wurden 30 / 3368 gefunden. Siehe CD results/candidates/example\_tree\_house.txt

## Volltextsuche

```
Options.init();
DataCategory cat = new DataCategory();
cat.definition="A noun is a word, that refers to things. Adjectives
can describe nouns. It is the same as a substantive. In germany nouns
begin with a big letter.";
cat = new Norm().reduceDC(cat);
DCComparator co = new DCComparator(cat);
co.findTermCandidates();
Iterator<Candidate> candidates =
    co.getCandidates().iterator();
while(candidates.hasNext())
{
    Candidate c = candidates.next();
    c.name = DBAccess.getDcName(c.pid);    //Name für Suche
                                           //irrelevant,
                                           //deswegen noch
                                           //unbekannt
    System.out.println(c.pid + "->" + c.name + "\t" + c.prefRel);
}

DC-3459->roman numeral gender    rel:related
DC-3190->feature assignment system    rel:related
DC-3069->adposition rel:related
DC-3197->feminine gender rel:related
DC-3086->arabic numeral gender rel:related
DC-2704->noun rel:related
DC-1523->specific term rel:related
...90
```

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann

Ort

Datum

Unterschrift