Universität Bielefeld

Bachelorarbeit

# Entity Linking through Indexing : Implementierung und Evaluierung

*Author:*
Tim Pontzen

*Supervisor:*
Prof. Dr. Philipp Cimiano

*A thesis submitted in fulfilment of the requirements for the degree of Bachelor of Science*

*in the*

Semantic Computing Group
Universität Bielefeld

20. Mai 2014

# Declaration of Authorship

I, Tim Pontzen, declare that this thesis titled, 'Entity Linking through Indexing : Implementierung und Evaluierung' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

# Abkürzungen

| | |
|---|---|
| **BI** | **B**lock**I**ndex |
| **ERU** | **E**ntity-**R**ecognition-**U**nit |
| **GUI** | **G**enerell **U**ser **I**nterface |
| **IBEL** | **I**ndex **B**ased **E**ntity **L**inker |
| **IBELU** | **I**ndex **B**ased **E**ntity **L**inker **U**tility |
| **idf** | **i**nverse **d**ocument **f**requency |
| **JAR** | **J**ava **AR**chieve |
| **NER** | **N**amed **E**ntity **R**ecognizer |
| **tf** | **t**erm **f**requency |
| **URI** | **U**niform **R**esource **I**dentifier |

# Abbildungsverzeichnis

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

# Kapitel 2

# Systemübersicht

In diesem Kapitel wird die Systemarchitektur und dessen Informationsfluss behandelt. Insgesamt besteht das System aus 4 Teilen:

- Graphical User Interface (GUI)
- Entity-Reconition-Unit (ERU)
- Linking Unit
- Index

Das nachfolgende Bild bietet einen Überblick über die Architektur und das Zusammenspiel der einzelnen Komponenten. Deren Funktion wird im Anschluss näher erläutert.

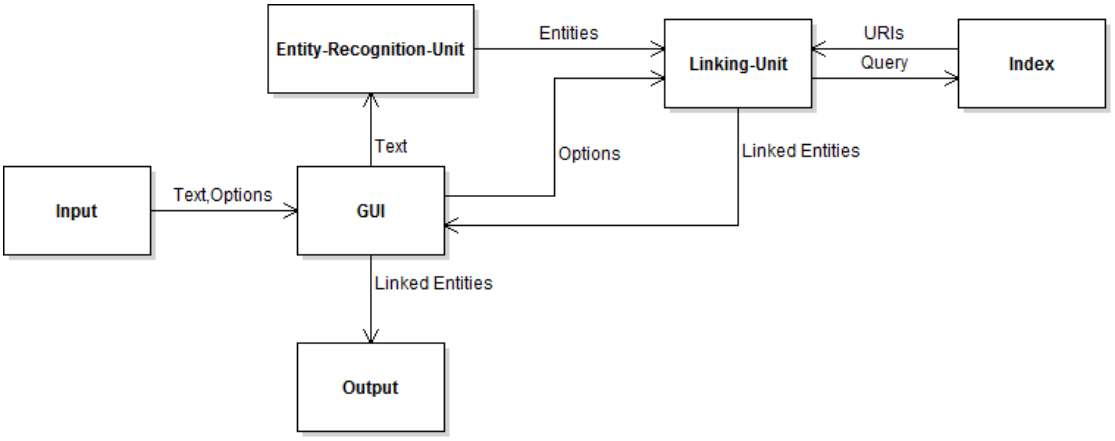


Abbildung 2.1: Systemübersicht

## 2.1 Graphical User Interface

Die GUI stellt die Schnittstelle zwischen Benutzer und Programm dar. Diese erlaubt ihm seinen zu anotierenden Text zu übergeben und zwischen verschiedenen Optionen zu wählen. Der Input wird entgegenommen und dessen Text an die Entity-Recognition-Unit weitergeleitet. Am Ende bekommt es die verlinkten Entities von der Linking-Unit zurürck und gibt diese an den Benutzer weiter.

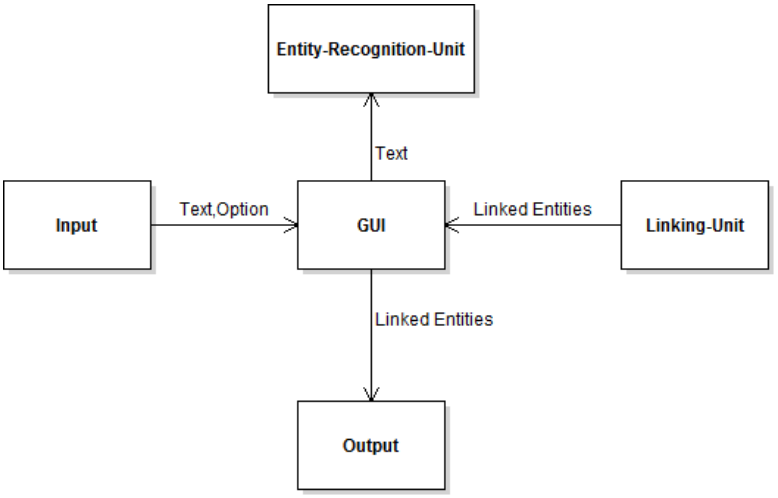


Abbildung 2.2: GUI-Verbindungen

## 2.2 Enitity-Recogniction-Unit

Diese Einheit ist für die Erkennung von Named Entities verantwortlich. Die bedeutet, dass sie den ihr übergebenen Text analysiert und alle sich darin befindlichen Entitäten extrahiert (z.B. Personen oder Orte). Das Ergebnis dieses Prozesses wird dann an die Linking-Unit weitergeleitet.

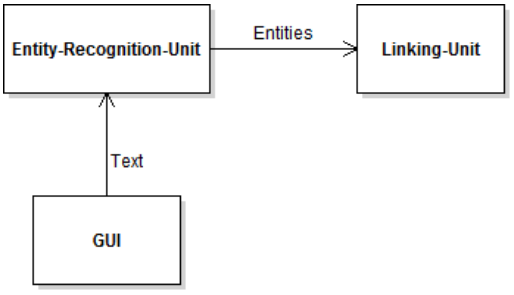


Abbildung 2.3: Enitity-Recognition-Unit

## 2.3 Linking-Unit und Index

Die Linking-Unit versucht mit Hilfe des Indexes für jede an sie weitergeleitete Entität eine passende URI zu finden. Dazu erstellt sie, unter Berücksichtigung der vom Benutzer eingestellten Optionen, eine Query und lässt den Index diese ausführen. Dessen Rückgabewert wird dann mit den Named Entities verknüpft und an die GUI weitergereicht.

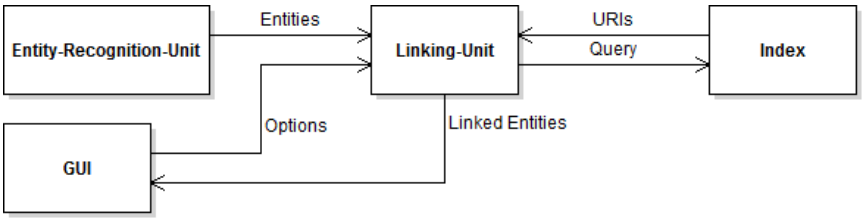


Abbildung 2.4: Linking-Unit

## 2.4 Informationsfluss

Um die Informationsverarbeitung noch einmal in einer Gesammtübersicht zu betrachten, sind alle Abläufe in dem folgenden Sequenzdiagramm verdeutlicht. Folgende Szenarien sind abgedeckt :

- Texteingabe
- Optionsauswahl
- Annotierung

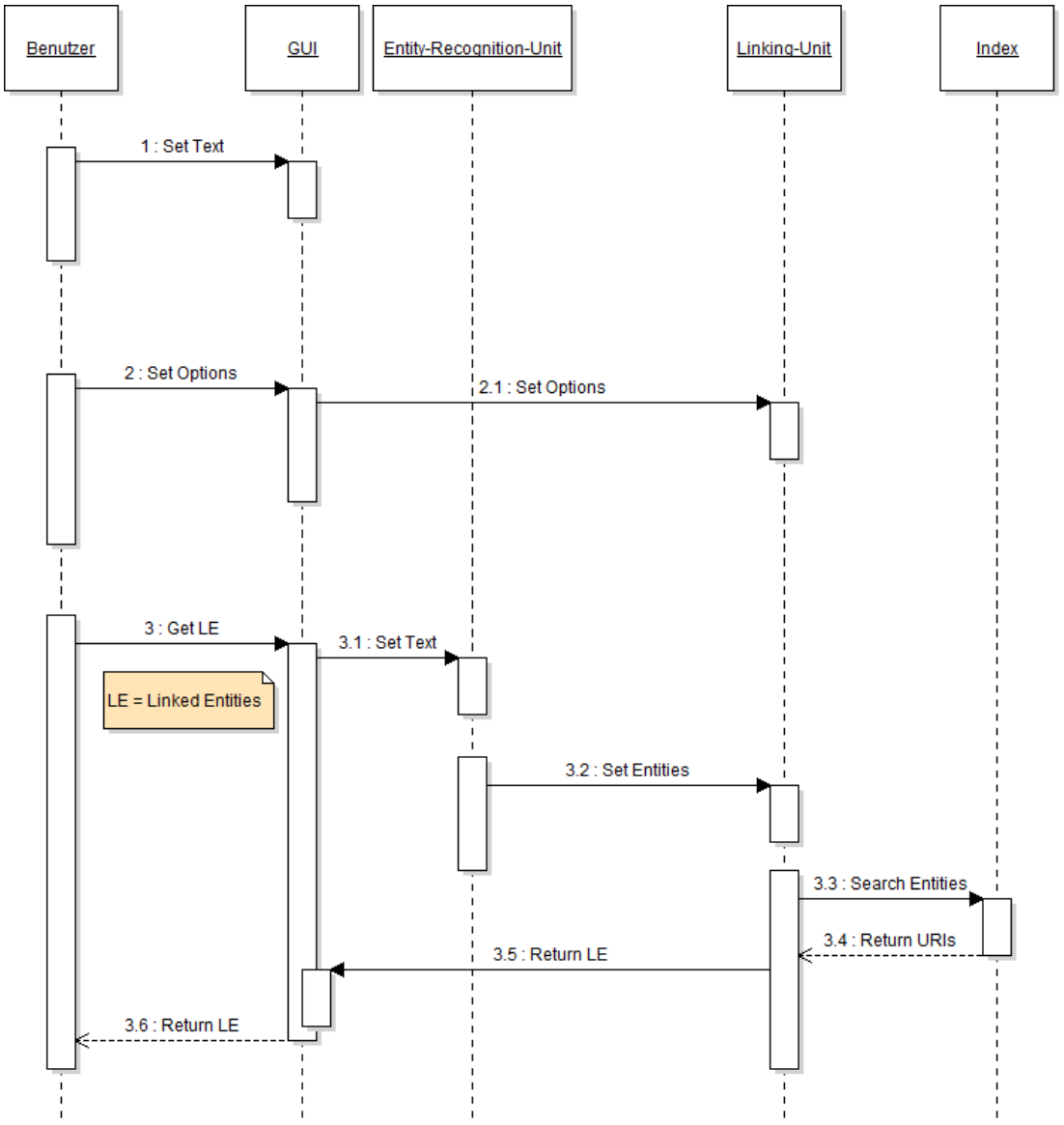


Abbildung 2.5: Programmablauf

# Kapitel 3

# Implementierung

Der Source-Code für den Index Based Entity Linker(IBEL) wird als ein Maven[1]-Projekt für Java verwaltet. Dies ermöglicht ein komfortables Warten und Weiterentwickeln des Codes, da alle benötigten Bibliotheken automatisch über das zentrale Maven-Repository bezogen werden. Zusätzlich ist sichergestellt, dass diese die richtige Version besitzen um Kompatibilitätsprobleme zu vermeiden.
Als Entwicklungsumgebung wurde Netbeans 8.0 verwendet und als Java-Version Java 7.
Im Folgenden wird nun die Implementierung der einzelnen Funktions-Einheiten, welche in Kapitel 2 vorgestellt wurden, behandelt.

## 3.1 GUI und Controller

Um die einzelnen Bestandteile des IBEL zu kapseln und somit leichter zu Warten, wurde eine Controller implementiert, dessen alleinige Aufgabe es ist, die Kommunikation zwischen den Klassen zu handhaben.
Die Gui nimmt weiterhin die Eingaben des Benutzers entgegen, jedoch wird die Auswertung der Interaktion nun von dem Controller übernommen. Hierfür implementiert dieser das ActionListener-Interface des java.awt.event-Packages.

[1] http://maven.apache.org/

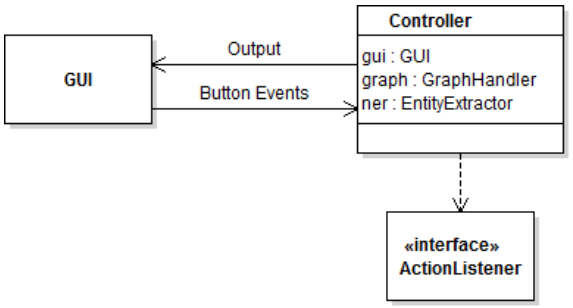


Abbildung 3.1: GUI and Controller

## 3.2 Enitity-Recognition-Unit

Um den vom Benutzer eingegeben Text untersuchen zu können, benutzt IBEL den Stanford NER, einen von der Universität Stanford entwickelten Named Entity Recognizer, der Wortsequenzen in einem Text kennzeichnet, bei denen es sich um Namen für Dinge handelt, wie zum Beispiel Personen oder Firmennamen [2].

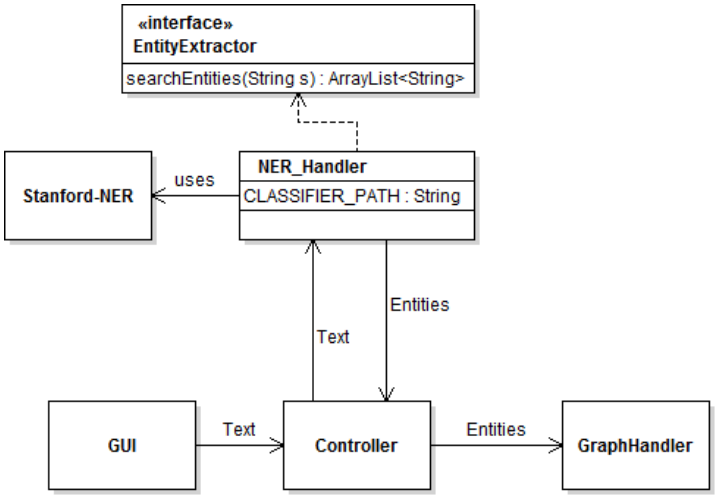


Abbildung 3.2: Entity-Recognition-Unit Implementation

Die Schnittstelle zwischen dem Controller und der Enitity-Recognition-Unit(ERU) bildet das EntityExtractor-Interface. Jede Klasse die dieses Implementiert, kann vom Controller genutzt werden um Entitäten in Texten erkennen zu lassen. Der NER_Handler, welcher dieses Interface implementiert, nutzt die Classifier des Stanford-NER um eine Liste mit Entitäten in dem ihm übergebenen Text zu extrahieren.

[2] http://nlp.stanford.edu/software/CRF-NER.shtml

## 3.3 Linking-Unit

Nachdem die Entity-Recognition-Unit alle gefundenen Entitäten an den Controller zurückgeben hat, werden diese an den GraphHandler zum Entity Linking weitergeleitet. Dieser erstellt Querys ür Lucene, einer performanten Open Source Java-Bibliothek für Textsuche[3], und lässt diese dann ausühren. Die ermittelten URIs werden mit den Entitäten verknüpft und an den Controller zurückgegeben.

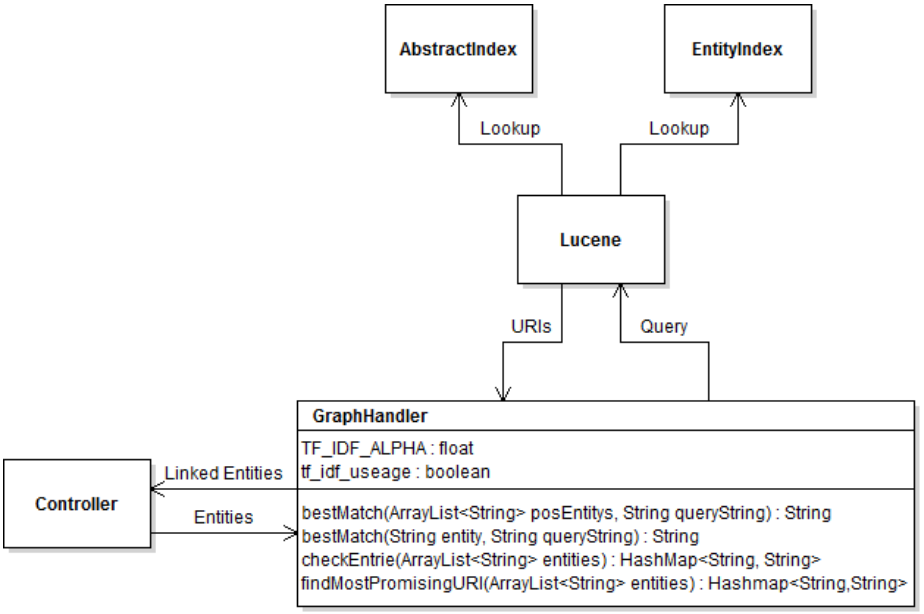


Abbildung 3.3: Entity Linking Unit

### Ablauf des Linkings

Der Entity-Linking-Algorithmus erhält eine Liste von Entitäten und durchläuft dann 3 Phasen:

1. Titelvergleich : Es wird geschaut ob die Entität ein Titel für eine DBpedia Seite ist. Zum Beispiel wäre die Entität „United States“ der Titel der DBpedia Seite mit der URI „http://dbpedia.org/resource/United_States“ .

2. Anchorsuche : Falls die Titelsuche kein Ergebnis liefert, so wird überprüft ob die Entität ein verzeichneter Anchor einer URI ist. In dem United States Beispiel

[3] http://lucene.apache.org/core/

wäre „USA“ einer dieser Anchors. Da ein Anchor disambig sein kann, wird versucht über den semantischen Kontext eine Entscheidung zu treffen. Dazu werden die Nachbaranchor der möglichen Kandidaten mit allen anderen gefunden Entitäten verglichen. Der Kandidat mit den meisten Übereinstimmungen hat die größte Wahrscheinlichkeit die gesuchte URI zu sein und wird deshalb mit der Entität verknüpft.

3. tf-idf-Suche : Wurde in den ersten beiden Schritten keine passende URI gefunden und der Benutzer hat die entsprechende Option angeben, so wird versucht über eine tf-idf-Suche eine URI zu ermitteln. Dazu wird in den Abstract-Texten aller DBpedia Seiten nach dem Vorkommen der Entität gesucht. Der “term frequencey“-Wert (tf) gibt dabei an, wie häufig die Entität gefunden wurde und die “inverse document frequencey“ (idf) wertet die Bedeutung im Kontext mit allen Dokumenten (in diesem Falle DBpedia Seiten). Die 20 erfolgreichsten Suchtreffer werden nun behandelt als wären sie Kandidaten für einen disambiguen Anchor. Mit ihnen wird wie in der vorherigen Phase verfahren.

Pseudocode :

```
function findURIs(List entities)
        map<k,v> result;
        URI tmp;
        forall entity in entities
                tmp=searchTitle(entity)                          \\titelsuche
                if(tmp!=null)
                        result.put(entity,tmp);
                else
                        tmp=anchorSearch(entity)                 \\Anchorsuche
                        if(tmp!=null)
                                result.put(entity,tmp);
                        else if(tf_idf_Search==true)
                                list posURIs=tf_idf_Lookup(entity)\\Abstractsuche
                                tmp=anchorSearch(posURIs)\\Anchorsuche mit
                                                        \\Abstractsuchergebnissen
                                result.put(entity,tmp)
                        endif
                endif
        endfor
        return result;
```

In dem Folgenden Aktivitätsdiagramm wird der noch einmal Linking-Algorithmus verdeutlicht.

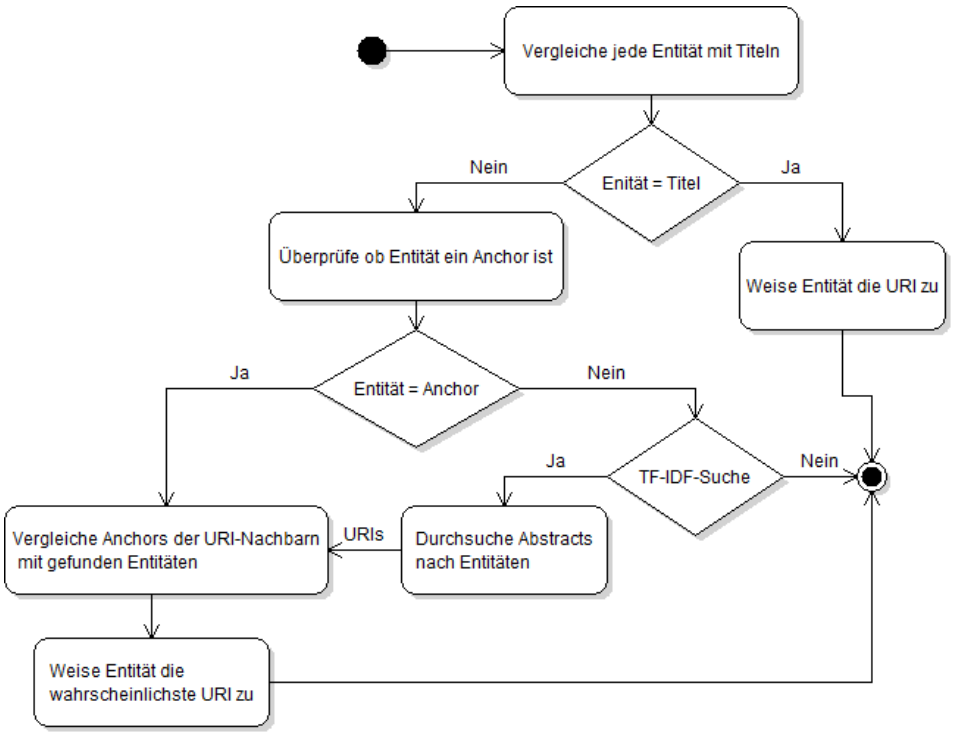


Abbildung 3.4: Entity Linking

## 3.4 Index

Ein Lucene-Index ist eine indexierte Ansammlung von Dokumenten. Ein Dokument besitzt eine beliebige Anzahl von Feldern in denen Informationen gespeichert werden können. Der Index aus der Systemübersicht wurde in der Implementierung in 2 Indexe unterteilt. Dies liegt hauptsächlich daran, dass sie unterschiedliche Formate für besitzen, um die aufgabenspezifischen Querys effizienter zu machen.

### 3.4.1 EntityIndex

Der EntityIndex wurde als BlockIndex angelegt, d.h. er besitzt 2 unterschiedliche Dokumenttypen:

1. Kinddokument : Ein Kinddokument ist ein gewöhnliches Dokument.

2. Elterndokument : Elterndokumente unterscheiden sich von Kinddokumenten durch ein Indikatorfeld das für alle Elterndokumente den gleichen Wert hat.

Um Eltern und deren Kinder bei der Queryauswertung richtig zuordnen zu können, muss bei der Indexerstellung eine bestimmte Reihenfolge eingehalten werden : Erst werden alle Kinddoumente eines gleichen Elter hinzugefügt, gefolgt von dem Elterndokument.
Dieses besondere Format ermöglicht `ToParentBlockJoinQuerys`. Diese Querys selektieren erst Elterndokumente und führen dann eine weitere Query auf den Kindern dieser Eltern aus. Dadurch kann sehr gezielt und somit effizient gesucht werden.
In der Implementation repräsentieren Elterndokumente die einzelnen Seiten aus DBpedia. Sie speichern den Titel, die URI und alle Anchors einer Seite. Die Kinddokumente besitzen nur ein Feld mit einem Anchor eines Nachbarn der URI aus dem Elterndokument.
Zur Veranschaulichung wird hier noch einmal das Format des EntityIndexes gezeigt:

```
<anchorN_{1_1}> (Kind 1)
...
<anchorN_{n_1}> (Kind n)
<URI_1,anchor_{1_1} ... anchor_{i_1},title_1,type> (Elter der Kinder 1 - n)
```

### 3.4.2 AbstractIndex

Der AbstractIndex ist ein Standart-Index, dessen Dokumente 2 Felder enthalten:

1. URI
2. Abstract der URI

Sein Format sieht wie Folgt aus:

```
<URI_1,abstract_1>
...
<URI_n,abstract_n>
```

Da Lucene für sein Query-Scoring tf-idf-Werte verwendet und auch genau diese bei der Abstract Suche erwünscht sind, liefert eine normale Query bereits das gewünschte Ergebnis.

# Kapitel 4

# Evaluierung

Dieses Kapitel beschäftigt sich intensiv mit der Analyse des Linking-Algorithmus. Insbesondere wird auf die Aspekte Geschwindigkeit und Genauigkeit eingegangen. Als Basis für die Auswertung wurden Texte aus Wikipedia[1] genommen und für die Semantik-Analyse Datensätze des Max Plank Institutes.
Die Tabellen für die Graphen finden sich im Anhang.

## 4.1 Geschwindigkeit

Der maßgeblicheste Faktor der Geschwindigkeit ist die Benutzungsdauer des Programms. Man spricht hierbei von einem "warmed up Index", was bedeutet, dass aufgrund von Cashe-Einträgen Daten schneller ermittelt werden können.

### 4.1.1 Unwarmed Mode

Da der "warmed up"-Modus sehr inkonsistente Ergebnisse für wiederholte Versuche mit gleichen Texten liefert, wird der "unwarmed mode" betrachtet, in dem das Programm noch über keinerlei Cashe-Einträge für Querys verfügt. Dies ist auch gleichzeitig eines der Worst-Case-Szenarien des Linking-Algorithmus.

[1] www.wikipedia.org

In einem durchschnittlichen Text lassen sich fast 80% aller , durch den Stanford-NER , gefunden Entitäten entweder mit einem Titel oder einem Anchor verbinden. Der Rest muss mit Hilfe von tf_idf-Werten bestimmt werden.

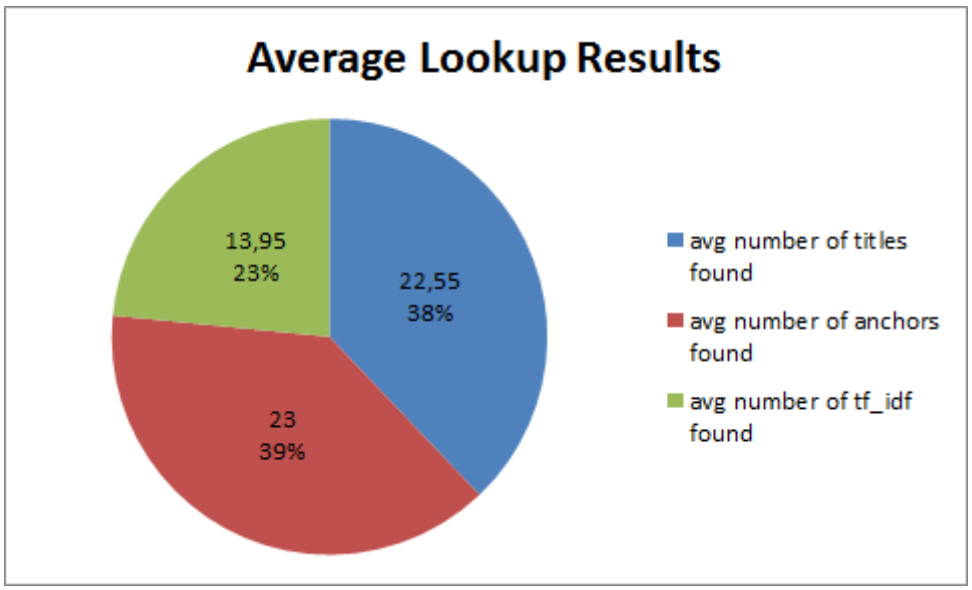


Abbildung 4.1: Average Lookup Results

Um die einzelnen Operationen in einem zeitlichen Zusammenhang betrachten zu önnen wird als nächstes die durchschnittlich benötigte Zeit für jede dieser Operationen betrachtet.

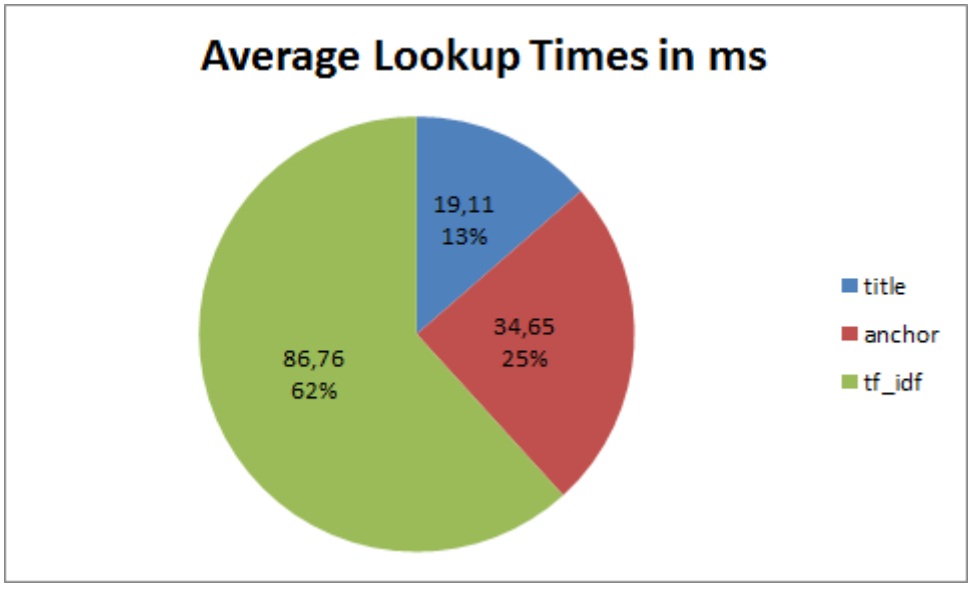


Abbildung 4.2: Average Lookup Times

Wie man in der Abbildung gut sehen kann, sind Titel- und Anchorsuche günstige Operationen im Vergleich zu der tf-idf-Suche, wobei die Titelsuche fast doppelt so schnell veräuft wie die Anchorsuche. Im Idealfall sollten die zu linkenden Entitäten entweder ein Titel oder ein Anchor sein.

**Anmerkung** : Bei der Abbildung handelt es sich nur um die benötigte Zeit der verschiedenen Operationen. Da diese aber sequentiell Ablaufen, bedeutet dies, dass die Linking-Zeit der Summer der benutzten Operationen entspricht (z.B. beträgt die Zeit für Anchor-Linking Titel-Lookup + Anchor-Lookup).

### 4.1.2 Warmed Up

Zum Vergleich mit einem “warmed up Index “ wurde die eine Hälfte der Testtexte benutzt um Cashe-Einträge anzulegen und die andere andere Hälfte wurde dann normal annotiert. Das Ergebnis ist ein deutlicher Anstieg in Geschwindigkeit.

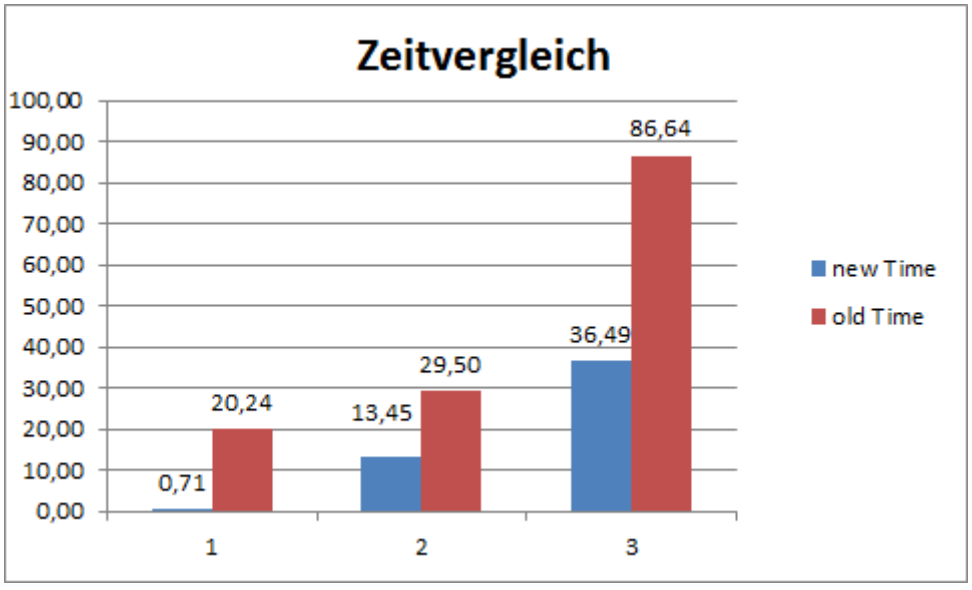


Abbildung 4.3: zeitvergleich

## 4.2 Genauigkeit

Nach der Betrachtung der Geschwindigkeit ist nun die Ergebnis-Qualität der einzelnen Operationen interessant.

### 4.2.1 Titelsuche

Da die Titelsuche nur für exakte Übereinstimmungen ein Ergebnis liefert, werden die Entitäten nahezu immer richtig verlinkt. Eine Fehlverlinkung tritt nur auf, wenn eine Entität mit einem Titel übereinstimmt, aber dieser im semantischen Kontext eine andere Bedeutung hat.
Beispiel :
Tiger was lost in the woods when he got divorced from Elin.[2]
Der Stanford-NER liefert für diesen Satz die 2 Entitäten Tiger und Elin zurück. Die Titelsuche findet dann für Tiger die URI `http://dbpedia.org/resource/Tiger`. Richtig wäre in diesem Falle aber `http://dbpedia.org/resource/Tiger_Woods`.

### 4.2.2 Anchorsuche

Die Anchorsuche besitzt die selben Eigenschaften wie die Titelsuche. Ihre Ergebnisse sind sehr Präziese und nur falsch, wenn durch den semantische Kontext eine andere Bedeutung entsteht.

### 4.2.3 Tf-Idf-Suche

Die tf-idf-Suche ist im Vergleich zu den ersten beiden Suchenmethoden sehr unzuverlässig. Sie ermittelt zwar eine URI für eine Seite, die zu dem semantischen Kontext und der Entität passt, ist aber meißtens eine falsche Verlinkung. Folgende Ursachen können dafür verantwortlich sein:

- Die Entität besitzt keine passende DBpedia-Seite.
- Der indexierte Abstract-Text der richtigen DBpedia-Seite ist zu kurz um eine Disambiguirung zu ermöglichen.
- Eine thematisch nahe verwandte DBpedia-Seite besitzt einen größeren Abstract-Text mit mehr Referenzen auf die Entität.
- Der Stanford-NER hat ein falsches Ergebnis geliefert.

[2] Auszug des KORE50-Datensatzes des Max Planck Instituts

Der letzte Fall ist besonders von Interesse, da die tf-idf-Suche eine sehr teure optionale Operation ist. Bei der Verlinkung der Test-Texte wurde diese Suche für 220 Kandidaten verwendet. Circa 56% davon waren falsche Entitäten, wie zum Beispiel die Adjektive german, russian und jewish .

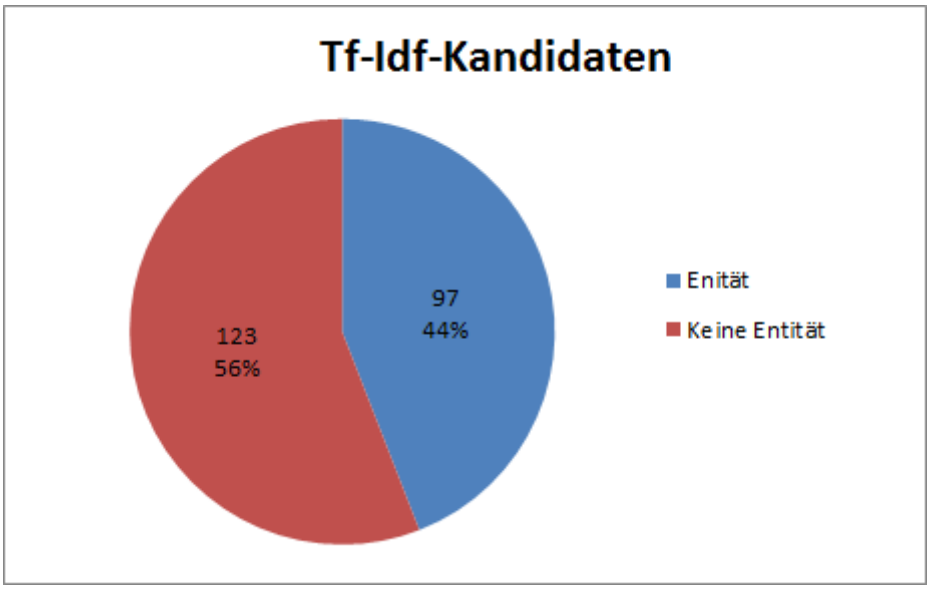


Abbildung 4.4: Tf-Idf-Kandidaten

## 4.3 Semantikanalyse mit KORE

Das Max Planck Institut für Informatik[3] bietet von Hand gefertigte Datensätze zum Testen von Disambiguirungsalgorithmen. Diese Keyphrase Overlap Relatedness for Entity Dismabiguation (KORE)[4] Testdaten sind so erstellt, dass sie besonders schwer zu Disambiguieren sind.

Da nahezu alle, vom Stanford-NER erkannten, Entitäten in diesen Texten, ihre Bedeutung nur über den semantischen Kontext erhalten und dies genau die schwäche der Titel- und Anchorsuche sind, schneidet der Index Based Entity Linker (IBEL) schlecht ab. Die Hauptproblematik ist, dass die Entitäten fast nie Anchor einer URI sind, noch der Titel zu einer indexierten Seite. Damit wird fast immer die tf-idf-Suche verwendet, welche den schwächsten Teil von IBEL darstellt.

Als Fazit für die Semantikanalyse ergibt sich, dass IBEL zwar Anchor gut über Kontextanalyse Disambiguiren kann, aber anchorlose Entitäten fast nie richtig verlinkt werden.

[3] http://www.mpi-inf.mpg.de/index_d.php
[4] http://www.mpi-inf.mpg.de/yago-naga/aida/download/KORE50.tar.gz

# Kapitel 5

# Deployment und Erweiterung auf andere Sprachen

## 5.1 Deployment

Um das Programm zu deployen muss lediglich die JAR mit compilierten Dependencies(falls das Programm aus dem Sourcecode compiliert wurde, ist dies die "IndexBasedEntityLinker-1.0-jar-with-dependencies.jar"), sowie ein passender Entity- und Abstract-Index in ein gleiches Verzeichnis kopiert werden.

### 5.1.1 Generierung der Indexe mit Utility-1.0

Sollten keine gültigen Indexe vorhanden sein, so können diese entweder manuell oder über die IndexBasedEntityLinker_Utility(IBELU) generiert werden.


Abbildung 5.1: Index Based Entity Linker Utility

Dafür werden folgende Dateien benötigt:

- Der DBpedia Anchor-Index der Computer Semantic Group der Universität Bielefeld
- Mapping-based Properties (en) von DBpedia[1]
- Extended Abstracts (en) von DBpedia[2]

Die Abfolge der auszführenden Operationen ist von links nach rechts sortiert und sollte auch in dieser Reihenfolge abgewickelt werden.
Zur Erstellung des Entity-Indexes werden der Anchor-Index und die Mapping-based Properties benötigt. Der Ablauf ist wie folgt:

1. Clean Properties: In dem zugehörigen Textfeld oberhalb des Buttons muss der Dateipfad (lokal oder absolut) zu der Mapping-Based Properties Datei angegeben werden. Diese wird dann von allen irrelevanten Daten gesäubert.
   Output: cleaned_properties.txt, cleaned_properties_neigborToEntity.txt und entities.txt.
2. Extract Anchors: Liest den Anchor-Index aus und schreibt alle <URI,anchor>-Paare in eine Datei.
   Output: anchors.txt
3. Pre BlockIndex: Erstellt für die Schnittmenge der URIs aus entities.txt und anchors.txt Dateien zur BlockIndex Generierung.
   Output:combined.txt, entity_anchors.txt, entity_neighbor_anchorsN.txt, neighbor_entity_anchorsE.txt
4. Create BlockIndex Erzeugt den BlockIndex(EntityIndex).
   Output : Blockindex

Zu Generierung des Abstract-Indexes werden die entities.txt aus dem ersten Teil und die Long-Abstracts benötigt:

[1] http://downloads.dbpedia.org/3.9/en/mappingbased_properties_en.nt.bz2
[2] http://downloads.dbpedia.org/3.9/en/long_abstracts_en.nt.bz2

1. Clean Abstracts: Nach Angabe des Dateipfades (lokal oder absolut) der long-abstracts-Datei werden für alle URIs die ein Abstract besitzen und in entities.txt vorkommen <URI, abstract>-Paare erstellt und gespeichert.
   Output: abstract_clean.txt

2. Create Abstract Index: Erstellt den Abstract-Index.
   Output:Abstract Index

Über Erfolg oder Misserfolg (Fehlermeldungen) der ausgeführten Operationen wird der Benutzer der IBELU über ein Konsolen-Feld (siehe Abbildung 5.1) informiert.

**Anmerkung**: Diese Operationen sind teilweise sehr Speicherintensiv. Es sollten mindestens 12GB RAM zur Verfügung gestellt werden und sichergestellt sein, dass die JVM diesen auch nutzen darf (VM Parameter: -Xmx12g).

### 5.1.2 Eigenständige Generierung von Dateien

Die IBELU kann auch nur zur reinen Indexerstellung genutzt werden, während die dafür benötigten Dateien anderweitig erstellt werden. Im Nachfolgenden werden die Dateien und ihre Formate näher erläutert.

#### Dateien zur Erstellung des Entity-Indexes

Um den Index über die IBELU zu erstellen, sind 2 Dateien erforderlich:

- combined.txt : In dieser Datei werden alle URIs auf ihre Nachbarn und deren Anchors gemappt. Ein Nachbar einer URI A ist in diesem Fall eine URI B, deren Graphknoten eine Kante zu dem Knoten von A besitzt.
  Format:

$\texttt{URI}_1\texttt{|Neighbor}_1\texttt{|anchor}_{1,1}\texttt{;...;anchor}_{1,i}$
...
$\texttt{URI}_1\texttt{|Neighbor}_{m_1}\texttt{|anchor}_{m_1,1}\texttt{;...;anchor}_{m_1,j}$
...
...
$\texttt{URI}_n\texttt{|Neighbor}_{m_n}\texttt{|anchor}_{m_n,1}\texttt{;...;anchor}_{m_n,k}$

- entity_anchors.txt : Diese Datei mappt alle URIs auf ihre Anchors.

  Format:

  $\texttt{URI}_1\,|\,\texttt{anchor}_{1,1};\ldots;\texttt{anchor}_{1,i}$
  ...
  ...
  ...
  $\texttt{URI}_n\,|\,\texttt{anchor}_{n,1};\ldots;\texttt{anchor}_{n,j}$

Diese Dateien sollten lexikographisch sortiert sein, um den resultierenden Index performanter zu machen.

**Anmerkung**: Natürlich können auch die Indexe eigenständig erstellt werden. Aufbau und Funktionsweise werden in dem Kapitel „Implementierung “ausführlich behandelt.

## 5.2 Erweiterung um andere Sprachen

Um IBEL mit anderen Sprachen nutzen zu können, müssen neue Indexe erstellt und die existierende Entity-Regonition-Unit angepasst beziehungsweise ersetzt werden.

### 5.2.1 Indexe

Für jede zu unterstüzende Sprache muss ein neuer Entity-Index und ein neuer Abstract-Index erstellt werden. In Kapitel 5.1 wurde dies bereits ausführlich behandelt. Die dafür benötigten Mapping-Based Properties sowie die long-Abstracts werden von DBpedia in 119 verschieden Sprachen zur Verfügung gestellt.
Die entsprechenden Anchors für diese Sprache müssen allerdings entweder eigenständig generiert oder aus einer anderweitigen Quelle bezogen werden.

### 5.2.2 Entity Recognition

Für die Entity-Regocnition-Unit kann, durch einen Austausch der Klassifizierer, weiterhin der Stanford-NER verwendet werden. Auf der Downloadseite des Stanford-NER[3] finden sich Klassifizierer für die deutsche und chinesische Sprache. Für andere Sprachen

[3] http://nlp.stanford.edu/software/CRF-NER.shtml

müssen eigene Klassifzierer trainiert werden. Das nötige Vorgehen dafür ist auf der FAQ-Seite[4] beschrieben. Um den neuen Klassifizierer nutzen zu können, muss der Ladepfad in der CLASSIFIER_PATH-Variable in der Klasse NER-Handler entsprechend angepasst werden.

Es wird auch jede andere Implementation von Named-Entity-Regocnition unterstütz. Dafür muss lediglich ein Adapeter[5] geschrieben werden, welcher das EntityExtractor-Interface implementiert. Dieser muss dann in der Klasse App.java der GUI anstelle des NER_Handlers im Konsturktoraufruf übergeben werden.

**Anmerkung** : Da das Programm im Nachhinein in eine Serverapplikation für die Semantic Computer Group umgebaut wird, wird die GUI voraussichtlich nicht mehr verwendet werden. Daher muss der Adapter dann den NER_Hanlder an einer anderen Stelle ersetzen.

[4] http://nlp.stanford.edu/software/crf-faq.shtml#a
[5] http://de.wikipedia.org/wiki/Adapter_(Entwurfsmuster)

# Kapitel 6

# Zusammenfassung

# Anhang A

# Evaluierungstabellen