Detecting mentions of characters in German novels with Apache UIMA^{TM} Ruta (in German)

30. September 2014

Inhaltsverzeichnis

1 Einleitung **1**

2 Installation **2**
2.1 Eclipse 2
2.2 UIMA Ruta 2
2.3 TreeTagger 3

3 Erste Schritte **3**
3.1 Die UIMA Ruta Workbench 3
3.2 Das erste Projekt 3

4 Die Sprache UIMA Ruta **8**

5 Personenfindung in „Der Idiot“ **12**
5.1 Einbindung von TreeTagger 13
5.2 Weiterentwicklung der Regeln 18

6 Der Annotation Test **23**
6.1 Das Referenzdokument 24
6.2 Fehleranalyse und Verbesserung 28

1 Einleitung

In diesem Tutorial wird die regelbasierte Skriptsprache UIMA Ruta anhand eines Beispiels vorgestellt. Das Ziel ist es, jedes Vorkommen einer Person (Vorname, Nachname etc.) in einem literarischen Text zu annotieren. Zunächst werden die Programme Eclipse, UIMA Ruta und TreeTagger installiert und konfiguriert. Zur Einführung in die Entwicklung werden grundlegende Elemente der Sprache

Ruta erläutert und die Ordnerstruktur innerhalb eines Projekts beschrieben. Anschließend wird das erste Kapitel eines Romans auf eindeutige Muster untersucht, um wichtige Merkmale zum Auffinden von Personen zu erhalten. Die Erläuterung verschiedener Regeln soll dabei helfen, die Syntax zu verstehen und Grundtechniken kennenzulernen. Nachdem TreeTagger vollständig in ein Projekt integriert wurde, können mehr Typen zur Regelentwicklung genutzt werden. Zum Schluss wird der *Annotation Test* in die Entwicklung miteinbezogen, der das Testen der Regeln automatisiert. Speziell hierfür wird ein vorher markiertes Referenzdokument benötigt.

2 Installation

2.1 Eclipse

Wir rufen zunächst die Internetpräsenz `www.eclipse.org` auf. Dort klicken wir auf den Reiter *Downloads* oder die Schaltfläche *Download Eclipse*. Auf der nachfolgenden Seite wird das Paket *Eclipse Standard* heruntergeladen. Zuvor müssen wir noch das entsprechende Betriebssystem und den Systemtyp auswählen. In Windows finden wir die Information unter **Windows** → **Computer** → **Rechtsklick** → **Eigenschaften** im Abschnitt *System*. Wenn wir das Paket heruntergeladen haben, muss es, z.B. mit dem freien Packprogramm 7-Zip[1], entpackt werden. In dem entpackten Ordner ist die Datei *eclipse.exe*, mit der die Anwendung gestartet wird. Beim Start von Eclipse muss ein Ordner angegeben werden, in den alle Projekte gespeichert werden. Mit Klicken auf *Use this as the default and do not ask again* werden wir nicht bei jedem Start aufgefordert einen Arbeitsplatz anzugeben.

2.2 UIMA Ruta

Wir befinden uns nun auf der Arbeitsoberfläche von Eclipse und wollen UIMA Ruta installieren. Dazu navigieren wir zu **Help** → **Install New Software...**. Anschließend erstellen wir unter *Add...* ein neues Repository mit dem Link `http://www.apache.org/dist/uima/eclipse-update-site/`. Jetzt wird nach den Paketen von UIMA Ruta gesucht und vor dem Eintrag *Apache UIMA Ruta* ein Haken gesetzt. Wir bestätigen zweimal und akzeptieren die Lizenzbedingungen. Daraufhin wird das Programm mit allen benötigten Paketen installiert. Nachdem Eclipse neu gestartet wurde, kann die UIMA Ruta Workbench benutzt werden. Im Fenster rechts oben klicken wir noch auf das Fenstersymbol mit dem Plus und bestätigen den Eintrag UIMA Ruta, um die Perspektive zu wechseln.

[1] `http://www.7-zip.org/`

2.3 TreeTagger

TreeTagger weist jedem Wort einer Wortart zu und bestimmt dessen Grundform. Die Wortarten sind dann in UIMA Ruta als Typen verfügbar und können in den Regeln benutzt werden. Wir laden uns das Programm auf der Internetseite `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/` unter der Überschrift *Windows version* herunter. Außerdem werden für die Entwicklung deutscher Texte die deutschen Parameterdateien benötigt. Nachdem wir alles entpackt haben, kopieren wir die Parameterdateien in den Unterordner *lib*. Jetzt erstellen wir eine Kopie von der größeren *german-utf8.par* Datei und benennen sie in *tagger-de-litte-endian.par* um. Danach wird ein neues Textdokument mit dem Inhalt „encoding=utf8" erstellt. Diese erhält den selben Namen, hat jedoch die Dateiendung „.properties". Dabei ist darauf zu achten, dass der Dateityp in den Eigenschaften anschließend *PROPERTIES-Datei* und nicht *Textdokument* ist. Damit UIMA Ruta den Part-of-speech Tagger auf dem Datenträger findet, muss ein neuer Eintrag in den Umgebungsvariablen von Windows erstellt werden. Wir klicken auf **Windows** → **Computer** → **Rechtsklick** → **Eigenschaften** und links im Reiter auf **Erweiterte Systemeinstellungen**. Dort kann unter *Umgebungsvariablen...* eine neue Systemvariable erstellt werden. Diese muss den Namen „TREETAGGER_HOME" haben und als Wert den Speicherort von TreeTagger, also beispielsweise „C:\Programme\TreeTagger".

3 Erste Schritte

3.1 Die UIMA Ruta Workbench

Nach der Installation von Eclipse und UIMA Ruta befinden wir uns auf der Arbeitsoberfläche von Eclipse (Abb. 1). Die Arbeitsoberfläche ist zunächst aufgeteilt in vier Bereiche. Auf der linken Seite ist der *Script Explorer*, in dem sämtliche Projekte verwaltet und in einer hierarchischen Struktur angezeigt werden. Unten und auf der rechten Seite gibt es verschiedene Teilbereiche, die beispielsweise Fehler in der Programmsyntax anzeigen. Die wichtigsten Teilbereiche für unser Beispiel sind der *Selection View*, der *Annotation Test* und der *Annotation Browser View*. Der obere Teil der Arbeitsoberfläche ist der eigentliche Arbeitsbereich. Hier werden die Regeln für das NER-System entwickelt, aber auch die markierten Dokumente angezeigt oder bearbeitet.

Bevor wir loslegen, sollte noch die Textkodierung in den Einstellungen von Eclipse geändert werden. Dazu rufen wir die Einstellungen mit **Window** → **Preferences** auf (Abb. 2). Anschließend navigieren wir zu **General** → **Workspace** und ändern bei *Text file encoding* den Eintrag in *UTF-8*.

3.2 Das erste Projekt

UIMA Ruta organisiert die Projekte im *Script Explorer* auf der linken Seite der Arbeitsoberfläche. Um ein neues Projekt zu erzeugen klickt man in einen freien Bereich des *Script Explorers* mit **Rechtsklick** → **New** → **UIMA Ruta**

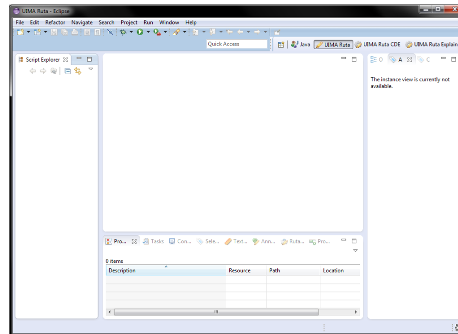

Abbildung 1: Die UIMA Ruta Workbench.

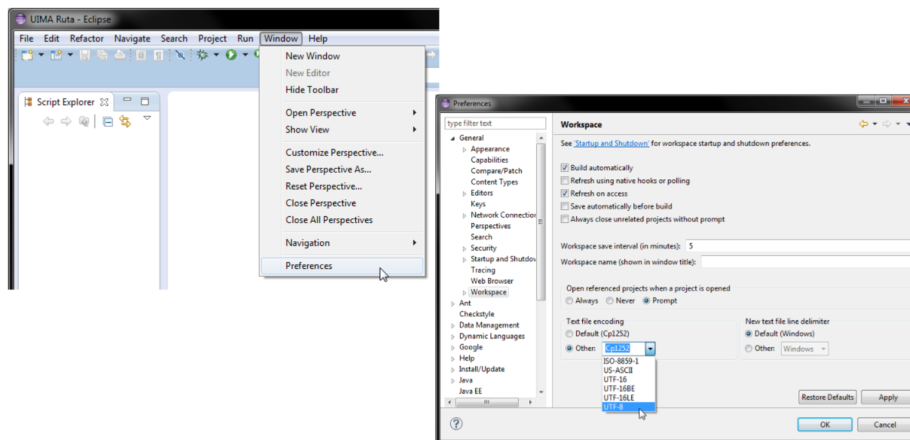

Abbildung 2: Die Einstellungen von Eclipse aufrufen und die Textkodierung ändern.

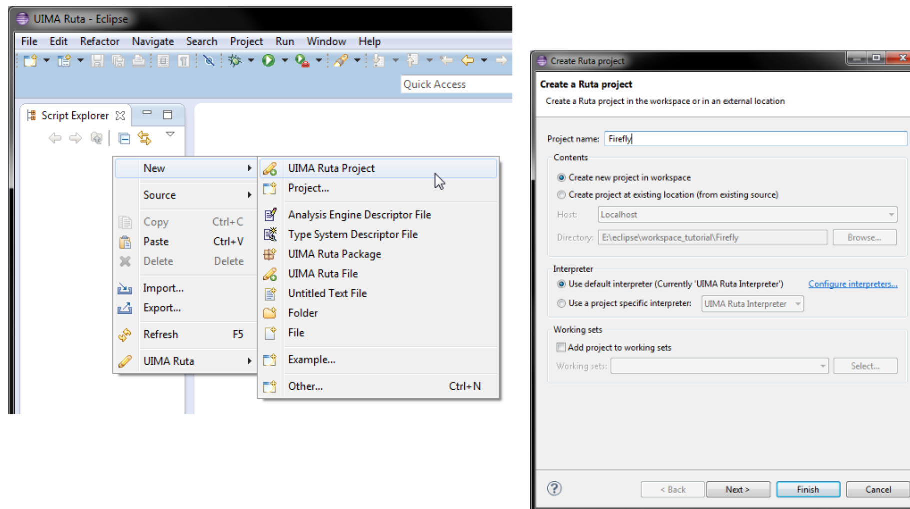

Abbildung 3: Ein neues Projekt erstellen und einen Namen angeben.

Project (Abb. 3). Im folgenden Fenster muss jetzt nur noch ein passender Name für das Projekt eingegeben und mit *Finish* bestätigt werden. UIMA Ruta erstellt daraufhin das Projekt mit allen nötigen Ordnern. Das ausgeklappte Projekt mit der vordefinierten Ordnerstruktur wird in Abb. 4 gezeigt. Das Projekt untergliedert sich in die Ordner *script*, *descriptor*, *input*, *output*, *resources* und *test*.

Im *script* Ordner werden alle UIMA Ruta Skripte und Pakete gespeichert, die im Laufe der Programmentwicklung entstehen. Das Analysewerkzeug und die verschiedenen Typesystems von UIMA Ruta befinden sich in *descriptor*. Hier können auch eigene Typesystems eingefügt und benutzt werden. Als Anfänger sollte allerdings nichts verändert werden, da sonst unerwartete Fehler auftreten könnten. In *input* speichert man die Texte, die von den Skripten bearbeitet werden sollen. Dies können Text-, HTML- oder XMI-Dateien sein. XMI-Dateien sind von UIMA Ruta erstellte Textdokumente, die bereits mit Annotationen versehen sind. Die vom Skript annotierten Dateien werden dann im *output* Ordner als XMI-Dateien gespeichert. Auf jede Datei im *input* Ordner folgt eine Datei in *output*. UIMA Ruta bietet die Möglichkeit Wortlisten, Wörterbücher oder Tabellen zur Regelentwicklung einzubeziehen. Diese Zusatzdateien werden im Ordner *resources* gespeichert, damit UIMA Ruta darauf zugreifen kann. Zum Evaluieren der Regeln gibt es den Ordner *test*. Hier werden markierte XMI-Dokumente in den jeweiligen Ordnern als Referenz gespeichert. Der *Annotation Test* prüft dann, wie gut die aktuellen Regeln auf die Texte zugeschnitten sind.

Ist ein größeres Projekt geplant, kann der Ordner *script* noch weiter in sog. Pakete aufgeteilt werden. Um ein neues Paket zu erstellen, klickt man einfach auf *script* mit **Rechtsklick** → **New** → **UIMA Ruta Package** (Abb. 5). Der Name wird dann im nächsten Fenster mit *Finish* bestätigt.

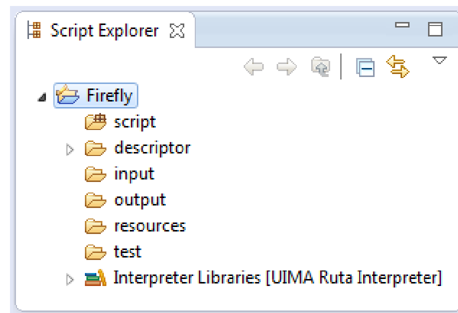

Abbildung 4: Das neue Projekt mit vordefinierter Ordnerstruktur.

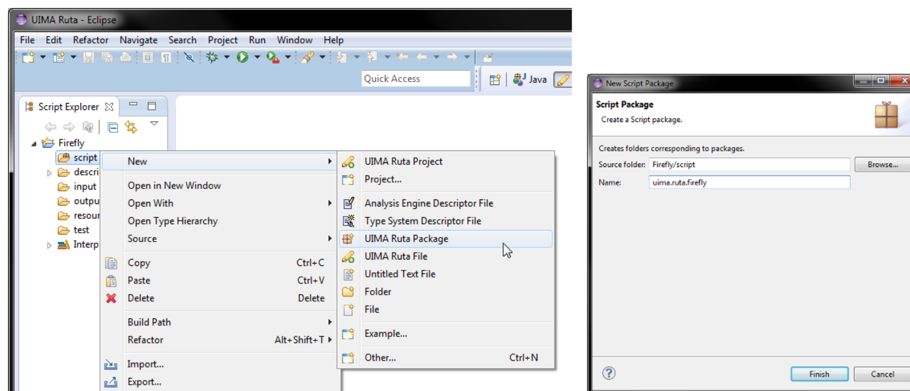

Abbildung 5: Ein neues Paket erstellen und einen Namen angeben.

Die Ansicht der Paketstruktur kann störend sein, wenn bei jedem Start von Eclipse durch die Pakete geklickt werden muss. Deshalb sollten wir noch die Einstellungen des *Script Explorer* aufsuchen und die hierarchische Struktur mit Klicken auf **Dreieck** → **Script Folder Presentation** → **Flat** (Abb. 6) ändern. Ein neues Skript wird durch Klicken auf das jeweilige Paket mit **Rechtsklick** → **New** → **UIMA Ruta File** (Abb. 7) erstellt. Der Name wird dann wieder mit *Finish* bestätigt.

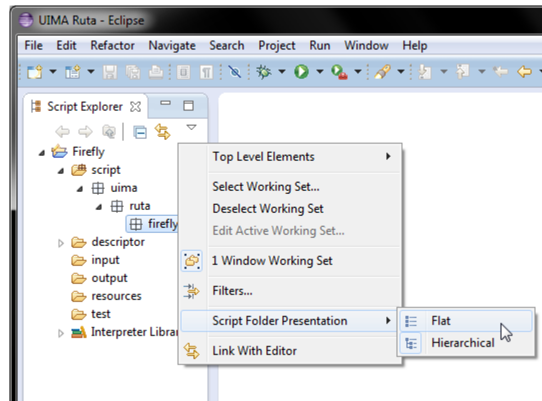

Abbildung 6: Hierarchische in flache Struktur ändern.

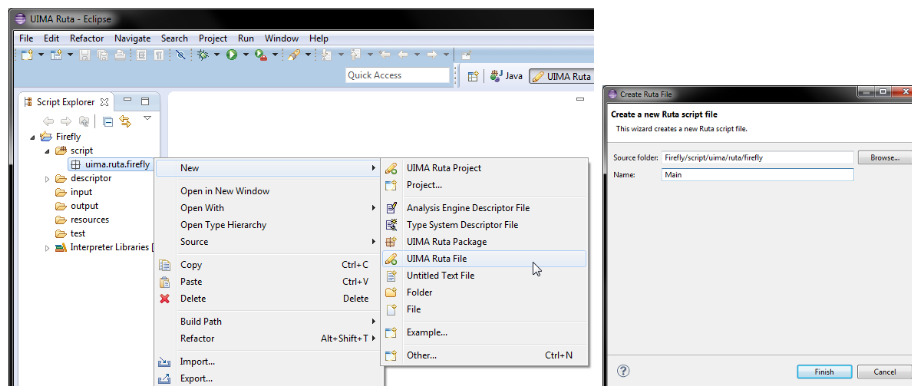

Abbildung 7: Ein neues Skript erstellen und einen Namen angeben.

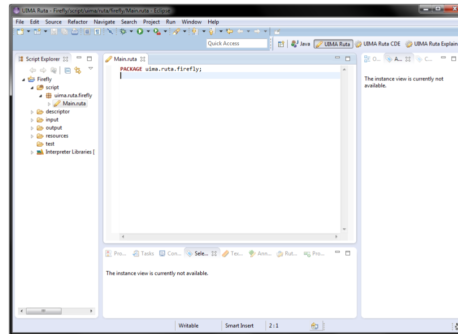

Abbildung 8: Die Arbeitsoberfläche nach erfolgreichem Anlegen eines neuen Projekts.

Das waren die ersten Schritte in UIMA Ruta. In Abb. 8 ist die Arbeitsoberfläche nach erfolgreichem Anlegen aller Dateien. Nun kann mit dem Schreiben der Regeln begonnen werden.

4 Die Sprache UIMA Ruta

Ein kleines Einstiegsbeispiel soll dabei helfen, die Sprache zu verstehen. Zunächst brauchen wir einen Text, auf dem wir die Regeln entwickeln. Dazu klicken wir auf *input* mit **Rechtsklick** → **New** → **File** (Abb. 9) und wählen im folgenden Fenster einen Namen mit der Endung „.txt“ für ein Textdokument.

Danach füllen wir das neu erzeugte Dokument mit einem Text, den wir anschließend mit UIMA Ruta bearbeiten wollen. In Abb. 10 steht ein Text mit mehreren Personen. Wir wollen versuchen diese Personen zu annotieren.

Der erste Schritt der Regelentwicklung ist die Analyse des gegebenen Textes. Ein zentraler Punkt ist, dass in der deutschen Sprache jeder Name großgeschrieben wird. Wir können uns also auf diese Teilmenge der Wörter im Text beschränken. Außerdem fällt im Beispiel auf, dass immer nach „Herr“ bzw. „Frau“ ein Nachname steht. Nach dieser Art von Mustern müssen wir also Ausschau halten, wenn wir automatisiert nach Personen suchen wollen. Die Vornamen im Text sind etwas schwerer ausfindig zu machen. Eine weitere Möglichkeit in UIMA Ruta ist daher die Verwendung von Wortlisten. „Klaus“, „Lisa“ und „Stefan“ sind gängige Vornamen. Warum sollte man diese also nicht zur Verwendung in eine Wortliste schreiben?

Die aufgeführten Punkte sind das Gerüst für unsere Regeln. Als erstes wird der Text analysiert und die Ergebnisse notiert. Erst danach überlegt man sich, wie die Informationen umgesetzt werden sollen. In Abb. 11 sieht man das Ergebnis unserer Regelumsetzung. Im *Script Explorer* ist jetzt zusätzlich die vom Skript erstellte Ausgabedatei im *output* Ordner. Außerdem wurde *resources* eine Wortliste mit Vornamen hinzugefügt. Auf der rechten Seite ist das Skript.
Die Wortlisten sind immer so aufgebaut, dass in jeder Zeile ein Ausdruck ste-

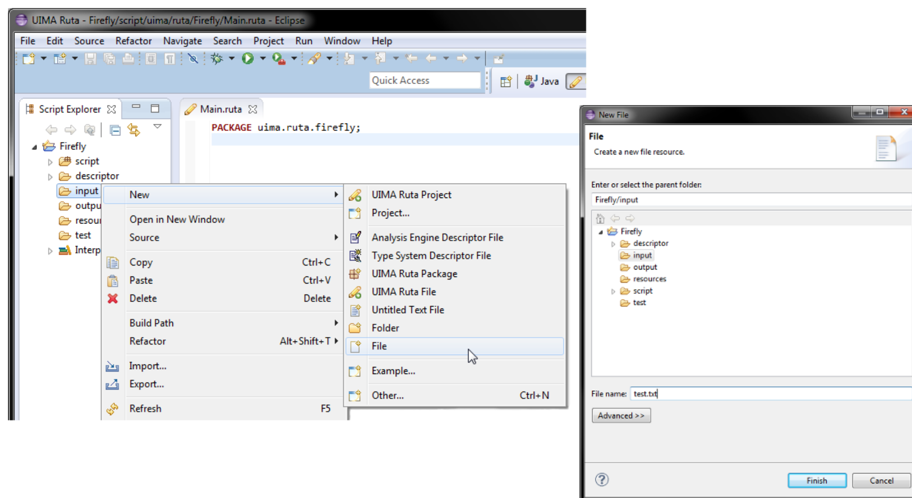

Abbildung 9: Ein neues Textdokument erstellen und einen Namen angeben.

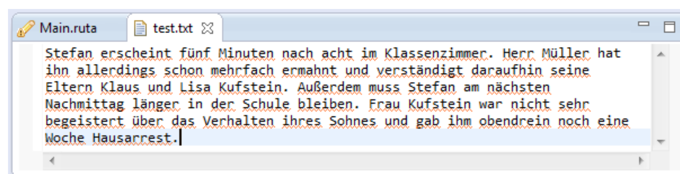

Abbildung 10: Beispieltext zum Entwickeln von Regeln.

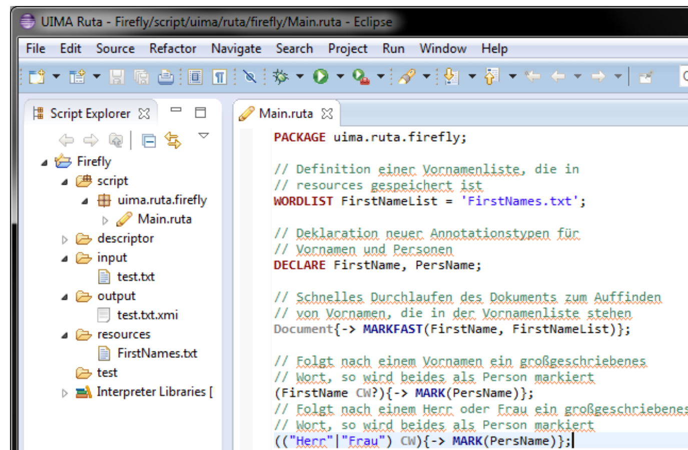

Abbildung 11: Auf der linken Seite ist die vordefinierte Ordnerstruktur mit Inhalt. Auf der rechten Seite ist das Skript mit den Regeln zur Personenfindung im Beispieltext.

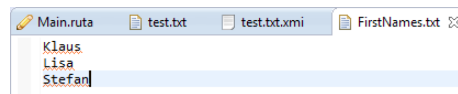

Abbildung 12: Zeilenweiser Aufbau der Wortliste mit Vornamen.

hen muss. Ein Ausdruck ist in unserem Fall ein Vorname. Dies bedeutet aber nicht, dass nur ein Wort pro Zeile stehen darf. Suchen wir beispielsweise den Ort „Frankfurt am Main“, können wir das auch genau so in eine Zeile schreiben. Abb. 12 zeigt die Wortliste *FirstNames.txt* aus unserem Beispiel. Das Erstellen einer Wortliste ist analog zur Erstellung einer Eingabedatei.

Wenden wir uns zu den Regeln in Abb. 11 und gehen das Skript Zeile für Zeile durch. `PACKAGE` gibt das Paket des Skriptes an, sodass UIMA Ruta den Speicherort findet. Danach folgen zwei Kommentarzeilen, die mit einem doppelten Slash (`//`) eingeleitet werden. Ein Kommentar dient der Beschreibung eines Abschnittes in einem Programm bzw. Skript. So kann sich eine zweite Person schnell zurechtfinden, ohne den Code zeilenweise durchgehen zu müssen. Die Wortliste wird mit `WORDLIST` eingebunden. Vor dem Gleichheitszeichen steht der Variablenname, mit dem wir die Wortliste innerhalb der Regeln ansprechen können. Danach kommt der Dateiname in einfachen Anführungszeichen. Jede Definition, Deklaration oder Regel wird mit einem Strichpunkt (`;`) beendet. Mit `DECLARE` definieren wir neue Annotationstypen *FirstName* und *PersName*. Alle vordefinierten Annotationstypen sind in Abb. 13 abgebildet und werden von UIMA Ruta selbst erzeugt.

Unsere Deklarationen werden erst im Skript erzeugt, indem wir einen gefundenen Abschnitt markieren. Eine Option wäre `MARKFAST`. Dazu muss zunächst

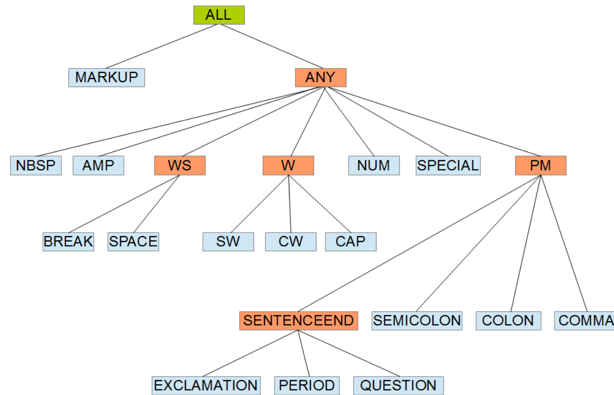

Abbildung 13: Sämtliche Klassen von Tokens von UIMA Ruta.

mit `Document` das aktuelle Dokument angesprochen werden. Jetzt folgt die Aktion in geschweiften Klammern ({}) und einem Pfeil nach rechts (`->`). Die Aktion `MARKFAST` nimmt dann zwei Parameter entgegen. Der Erste ist der Annotationstyp, den wir annotieren wollen. Der zweite Parameter ist die Wortliste, mit der wir das Dokument abgleichen. Anschließend wird das gesamte Dokument auf die Zeilen in der Wortliste geprüft. Jeder Treffer wird als *FirstName* markiert und ist als solcher in den Regeln verfügbar. Eine andere Option ist `MARK`. Die Aktion `MARK` hat meistens einen Parameter. Dieser ist der Annotationstyp mit dem wir einen bestimmten Abschnitt annotieren wollen, hier als *PersName*. Der Ausdruck vor der geschweiften Klammer steht in runden Klammern (`()`). Somit wird jeder Abschnitt, der den Regeln innerhalb der Klammern entspricht, als *PersName* markiert. Die Regel `FirstName CW?` sucht nach einem *FirstName* und prüft dann, ob ein großgeschriebenes Wort (CW = capitalized word) folgt. Ist dies der Fall, wird beides als *PersName* markiert. Ein Treffer in unserem Text ist also „Lisa Kufstein", da „Lisa" zuvor als *FirstName* markiert wurde und „Kufstein" ein großgeschriebenes Wort ist. Das Fragezeichen (`?`) ist ein Quantifizierer, der die Optionalität von `CW` ausdrückt. Deswegen wird auch „Stefan" und „Klaus" als *PersName* markiert. Die letzte Regel `("Herr"|"Frau") CW` sucht nach dem Ausdruck „Herr" oder „Frau". Folgt darauf ein großgeschriebenes Wort, wird es als *PersName* markiert. Diese Regel findet „Herr Müller" und „Frau Kufstein" in unserem Beispiel. Wir starten das Skript, indem wir den grünen Kreis mit dem Pfeil unterhalb der Menüleiste drücken. Mit einem Doppelklick auf die Datei im *output* Ordner lässt sie sich einsehen. Sollte UIMA Ruta nach einem geeigneten Typesystem fragen (Abb. 14), sucht ihr den Namen eures Skripts mit angehängtem „TypeSystem.xml".

Das Ergebnis unserer Entwicklung ist in Abb. 15 zu sehen. Auf der linken Seite ist der markierte Text, während auf der rechten Seite im *Annotation Browser*

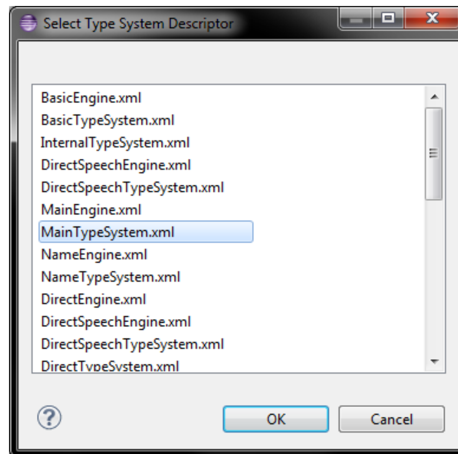

Abbildung 14: Auswahlfenster für ein Typesystem.

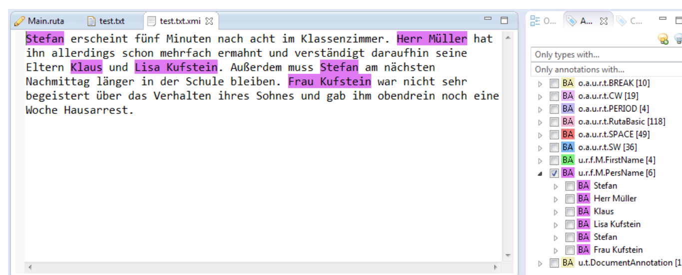

Abbildung 15: Auf der linken Seite ist der markierte Text. Rechts im *Annotation Browser View* können Annoationstypen ausgewählt werden, die im Text markiert werden sollen

View jede Annotation des angezeigten Textes ausgewählt werden kann.

Jeder Annotationstyp wird in einer eindeutigen Farbe hervorgehoben. Mit dem angefertigten Skript haben wir jede Person in dem Beispieltext gefunden. Im nächsten Kapitel wenden wir uns dem literarischen Text zu. Eine vollständige Auflistung und Erklärung aller Spracheigenschaften von UIMA Ruta ist im *Apache UIMA^{TM} Ruta Guide and Reference*[2].

5 Personenfindung in „Der Idiot“

Das Wissen aus Kapitel 4 wollen wir nun auf das erste Kapitel von Dostojewskis „Der Idiot“anwenden. Es werden fortgeschrittene Regeln verwendet, die u.a. nicht auf die Technik des *Part-of-speech Tagging* verzichten können. Dabei wird

[2] `https://uima.apache.org/d/ruta-current/tools.ruta.book.html`

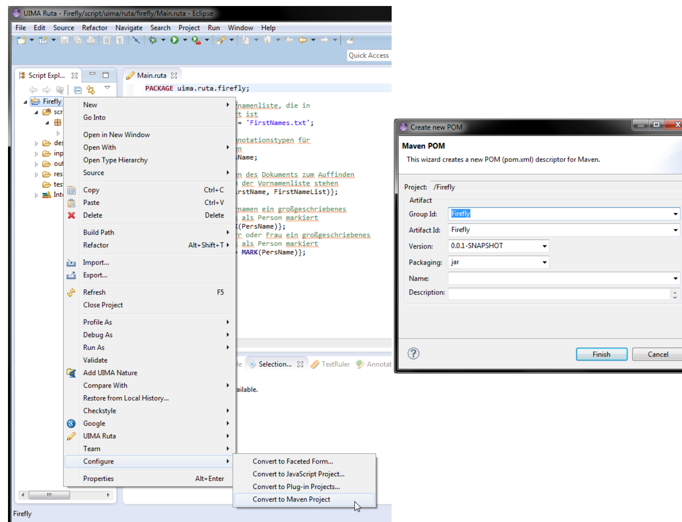

Abbildung 16: Das UIMA Ruta Projekt wird in ein Maven Projekt konvertiert.

jedes Wort und Satzzeichen einer Wortart (part of speech) zugeordnet. Es werden sowohl Wortdefinitionen, als auch der Kontext zur Analyse herangezogen. Der in Kapitel 2.3 vorgestellte TreeTagger ist ein solches Analysewerkzeug, den wir für unser Beispiel in UIMA Ruta einbinden werden.

5.1 Einbindung von TreeTagger

Als erstes muss das Projekt *Firefly* konvertiert werden. Dazu klicken wir auf das Projekt mit **Rechtsklick** → **Configure** → **Convert to Maven Project**, wie in Abb. 16 dargestellt. Im darauffolgenden Fenster wird mit *Finish* ein neues *pom.xml* erstellt. Ist der Eintrag *Convert to Maven Project* nicht verfügbar, fehlt das entsprechende Paket. Zur Installation des Paketes gehen wir zu **Help** → **Install New Software...** und wählen unter *Work with* alle eingetragenen Seiten aus (Abb. 17). Anschließend suchen wir das Paket *m2e - Maven Integration for Eclipse* unter dem Punkt *Collaboration*. Die Installation startet, nachdem wir zweimal auf *Next* drücken und den Lizenzbedingungen zustimmen. Falls während der Installation ein Fenster erscheint, das uns vor unsigniertem Inhalt warnt, wird es einfach mit *OK* bestätigt.

Ist die Datei *pom.xml* erstellt, öffnet sie sich sofort in der Übersicht (Abb. 18). Wir navigieren unten von *Overview* nach *pom.xml*, um den Inhalt der Datei zu bearbeiten (Abb. 19).

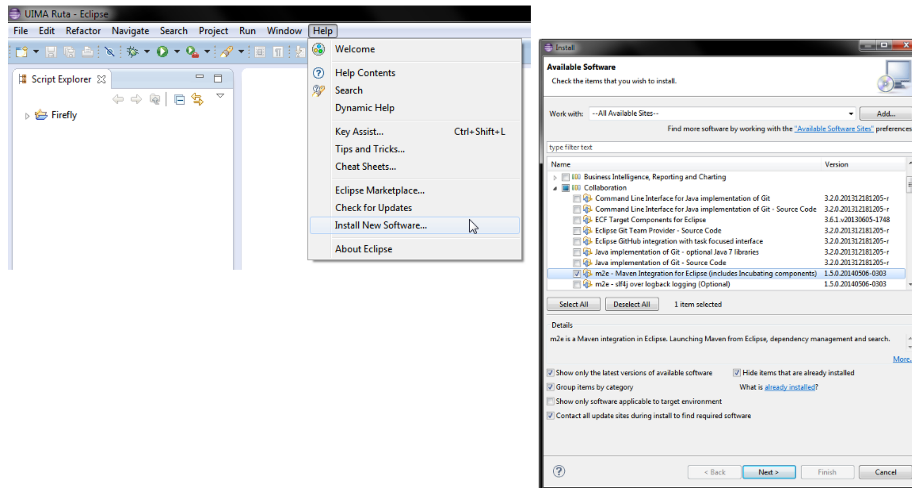

Abbildung 17: Im Menü *Help* wird die Softwareinstallation ausgewählt. Anschließend müssen die entsprechenden Pakete zur Installation ausgewählt werden.

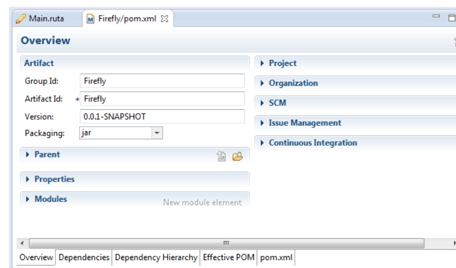

Abbildung 18: Übersicht der POM-Datei.

Abbildung 19: Der Inhalt von *pom.xml* nach der Konvertierung.

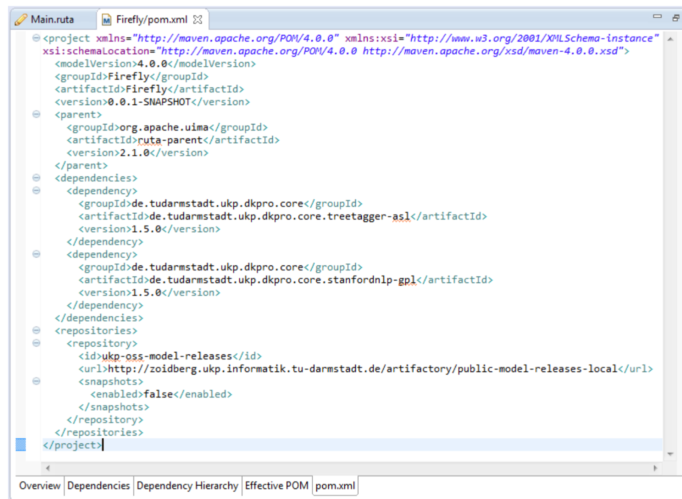

Abbildung 20: Der Inhalt des *pom.xml* nach Einfügen der Abhängigkeiten.

Vor dem Eintrag `</project>` muss zur Einbdinung von TreeTagger folgender Text eingefügt werden:

```
<parent>
  <groupId>org.apache.uima</groupId>
  <artifactId>ruta-parent</artifactId>
  <version>2.1.0</version>
</parent>
<dependencies>
  <dependency>
    <groupId>de.tudarmstadt.ukp.dkpro.core</groupId>
    <artifactId>de.tudarmstadt.ukp.dkpro.core.treetagger-asl</artifactId>
    <version>1.5.0</version>
  </dependency>
  <dependency>
    <groupId>de.tudarmstadt.ukp.dkpro.core</groupId>
    <artifactId>de.tudarmstadt.ukp.dkpro.core.stanfordnlp-gpl</artifactId>
    <version>1.5.0</version>
  </dependency>
</dependencies>
<repositories>
  <repository>
    <id>ukp-oss-model-releases</id>
    <url>
      http://zoidberg.ukp.informatik.tu-darmstadt.de/
      artifactory/public-model-releases-local
    </url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

Das Ergebnis ist in Abb. 20 zu sehen.

Nachdem das *pom.xml* bearbeitet wurde, müssen wir das Projekt updaten. Dazu klicken wir auf das Projekt und drücken **Alt** + **F5**. Darauf erscheint ein

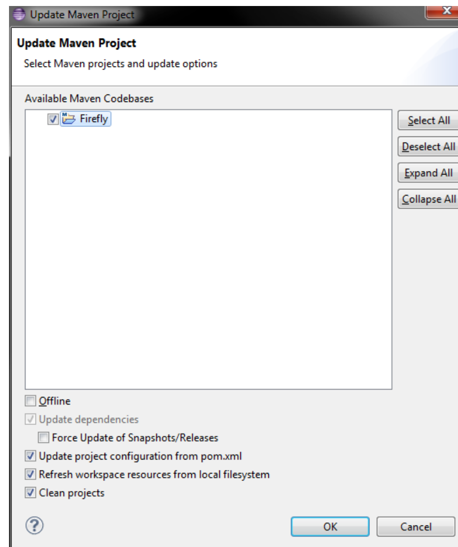

Abbildung 21: Fenster zum Aktualisieren eines Maven Projektes.

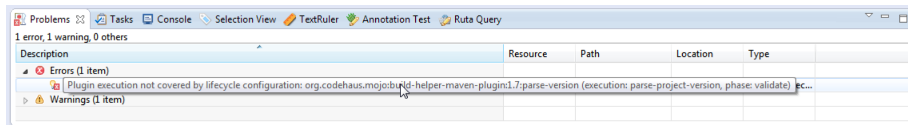

Abbildung 22: Möglicher Fehler nach der Projektkonvertierung.

Fenster zum Aktualisieren des Projektes (Abb. 21), das wir einfach mit *OK* bestätigen.

Sollte nun unter *Problems* der Fehler aus Abb. 22 vorhanden sein, beheben wir ihn mit der Schnellhilfe. Dazu klicken wir auf den Eintrag mit **Rechtsklick** → **Quick Fix** und bestätigen das nächste Fenster mit *Finish*.

Anschließend öffnet sich der *m2e Marketplace* (Abb. 23), den wir ebenfalls mit einem *Finish* bestätigen. Wir folgen den Schritten zur Installation des *build-helper*, nach denen der Fehler behoben sein sollte.

Der nächste Schritt fügt das Typesystem von TreeTagger hinzu. Dazu kopieren wir die Datei *GeneratedDKProCoreTypes.xml* aus dem Anhang in den Ordner *descriptor*. Anschließend sollte der *Script Explorer* wie in Abb. 24 aussehen.

Nun kann TreeTagger in UIMA Ruta verwendet werden. Als letzten Schritt erzeugen wir, wie in Kapitel 3.2 beschrieben, ein neues Paket und Skript. Dieses Skript ruft den Part-of-speech Tagger auf und versieht das Eingabedokument mit neuen Annotationstypen. Jetzt kann es in *Main.ruta* aufgerufen werden, damit wir die Annotationstypen in unseren Regeln nutzen können. Der Inhalt des neuen Skripts ist in Abb. 25 zu sehen.

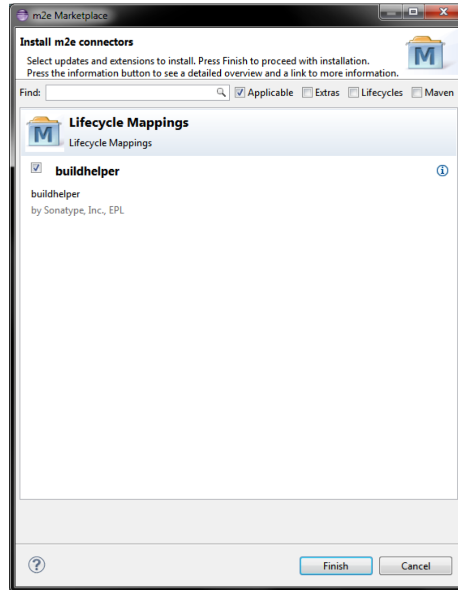

Abbildung 23: Der *m2e Marketplace*.

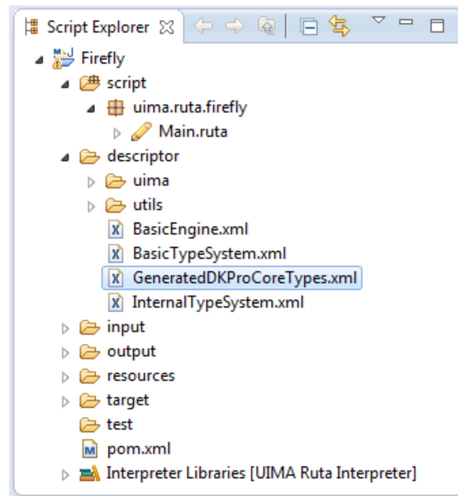

Abbildung 24: Das Typesystem DKPro in *descriptor* kopieren.

```
PACKAGE uima.ruta.postag;

IMPORT PACKAGE de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos FROM GeneratedDKProCoreTypes AS pos;
IMPORT de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.Lemma FROM GeneratedDKProCoreTypes;
UIMAFIT de.tudarmstadt.ukp.dkpro.core.stanfordnlp.StanfordSegmenter;
UIMAFIT de.tudarmstadt.ukp.dkpro.core.treetagger.TreeTaggerPosLemmaTT4J;

Document{-> SETFEATURE("language", "de")};

Document{-> EXEC(StanfordSegmenter)};
Document{-> EXEC(TreeTaggerPosLemmaTT4J, {pos.POS, Lemma})};
```

Abbildung 25: Ein Skript zum Ausführen des TreeTaggers.

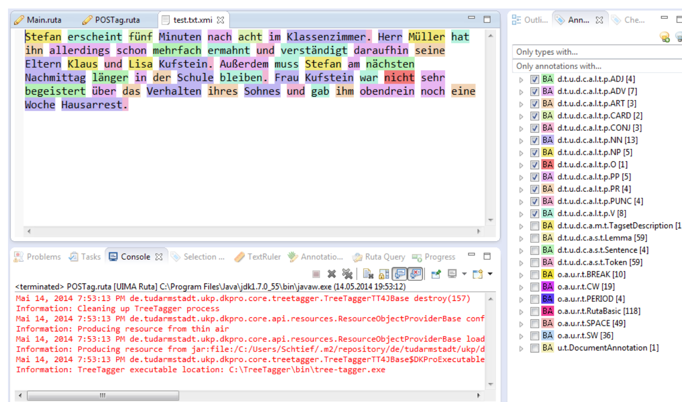

Abbildung 26: Im *Annotation Browser View* sind alle Annotationstypen von TreeTagger ausgewählt. Für jedes Wort gibt es links eine spezielle Zuordnung.

Führt man das Skript auf dem Eingabetext aus, können im *Annotation Browser View* weitere Annotationstypen ausgewählt werden (Abb. 26).

5.2 Weiterentwicklung der Regeln

Das Projekt wird immer größer, also ist es ratsam, möglichst jede Aufgabe aufzuteilen. Das bedeutet, wir haben ein Skript für die Annotationstypen von TreeTagger, ein Skript für das Finden von Namen im Text und ein anderes, das alle Skripte aufruft und somit das Hauptskript wird. Jetzt kann jeder Teilbereich getestet werden, ohne, dass wir jedes Skript ausführen müssen. Das Hauptskript wird *Main.ruta* sein (Abb. 27) und alle anderen relevanten Skripte aufrufen. Der Befehl `SCRIPT` importiert ein Skript, während es der Befehl `CALL` ausführt. Die zwei Zeilen vor dem Skriptimport müssen in jedes Skript eingefügt werden, das TreeTagger nutzt.

Die bisher entwickelten Regeln kopieren wir in das Skript *Name.ruta*, wie in Abb. 28 zu sehen ist. Dabei ist wieder zu beachten, dass die zwei Zeilen im Kopfteil eingefügt werden.

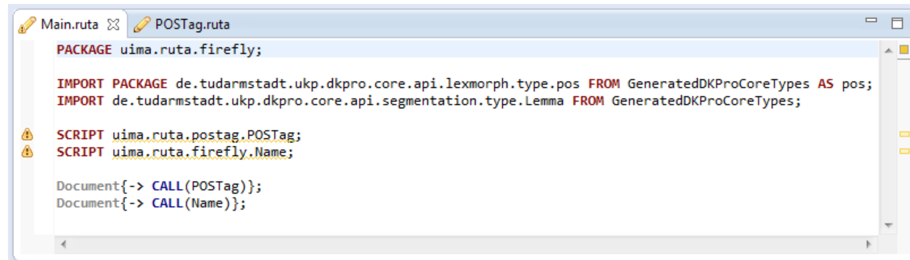

Abbildung 27: Das Hauptskript, indem alle Skripte aufgerufen und ausgeführt werden.

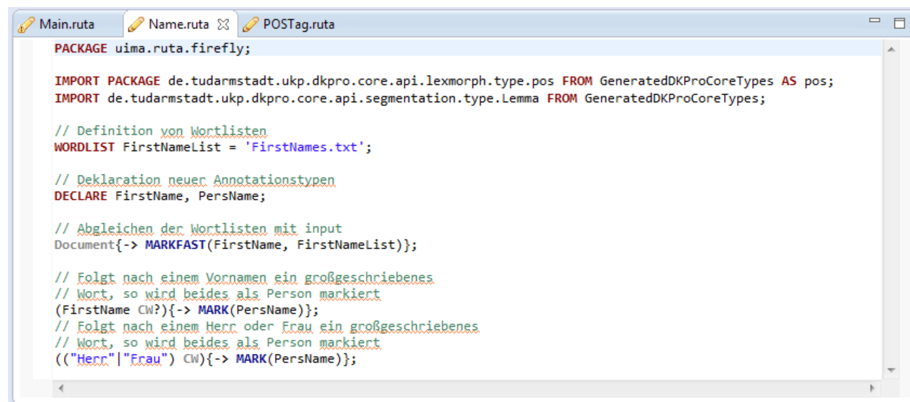

Abbildung 28: Der ursprüngliche Inhalt des Hauptskriptes wird in *Name.ruta* kopiert.

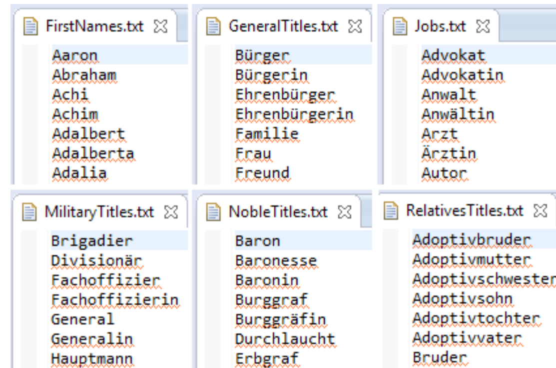

Abbildung 29: Ausschnitt der verschiedenen Wortlisten.

Der erste Schritt für unser neues Dokument ist wieder die Textanalyse. Zunächst kopieren wir aber das beigefügte Dokument *dostoevskij - idiot.txt* in den *input* Ordner und löschen die Datei *test.txt* aus *input* und *output*. Durch Doppelklicken öffnen wir das Textdokument und beginnen mit der Analyse. Währenddessen können die Vornamen, die noch nicht in unserer Liste stehen, hinzugefügt werden. Es fällt auf, dass häufig direkte Reden verwendet werden. Vor einer direkten Rede, bzw. danach, ist der Sprecher oft erwähnt. Weiter fällt auf, dass ein Titel, wie „Herr“, und ein folgendes großgeschriebenes Wort auf eine Person hindeutet. Ein Titel ist dabei ganz allgemein gehalten oder kann eine Berufsbezeichnung, Adelstitel oder Familienzugehörigkeit sein. Mit diesen Informationen können einfache Regeln realisiert werden. Wie für die Vornamen brauchen wir auch für alle Titel eine Wortliste. Dazu legen wir ein paar neue Wortlisten in *resources* an, wie es in Kapitel 3.2 beschrieben wurde. Alle Titel sollten alphabetisch geordnet sein, damit ein weiteres Hinzufügen von Titeln nicht zu Problemen führt. Zum Schluss haben wir jeweils eine Liste für Berufe, Adelstitel, Angehörige und Militärgrad. Die Titel bzw. Vornamen werden manuell eingefügt oder können aus einer Internetdatenbank heruntergeladen werden. Ein Ausschnitt der Wortlisten ist in Abb. 29 zu sehen.

Als nächstes müssen nur noch die Regeln angepasst und erweitert werden. Dazu zählen die neuen Wortlisten, neue Deklarationen für die Titel und das Durchlaufen des Dokuments mit `MARKFAST`. Die Änderungen und die neuen Regeln sind in Abb. 30 zu sehen. Zur besseren Verständlichkeit ist jeder Regelabschnitt mit einem großgeschriebenen Kommentar als Überschrift gekennzeichnet.

Die erste Regel im Vornamenabschnitt sucht nach Titeln, die zuvor als *RelativesTitle* markiert wurden. Folgt danach ein großgeschriebenes Wort, ist dies ein potenzieller Vorname und wird als *FirstName* markiert. Der `-PARTOF` Befehl verhindert, dass ein schon annotierter *FirstName* doppelt markiert wird. Die zweite Regel findet einen *FirstName* gefolgt von einem *FirstName*. In „Der Idiot“ hat fast jede Person einen zweiten Vornamen. Diese Vorkommen wollen wir

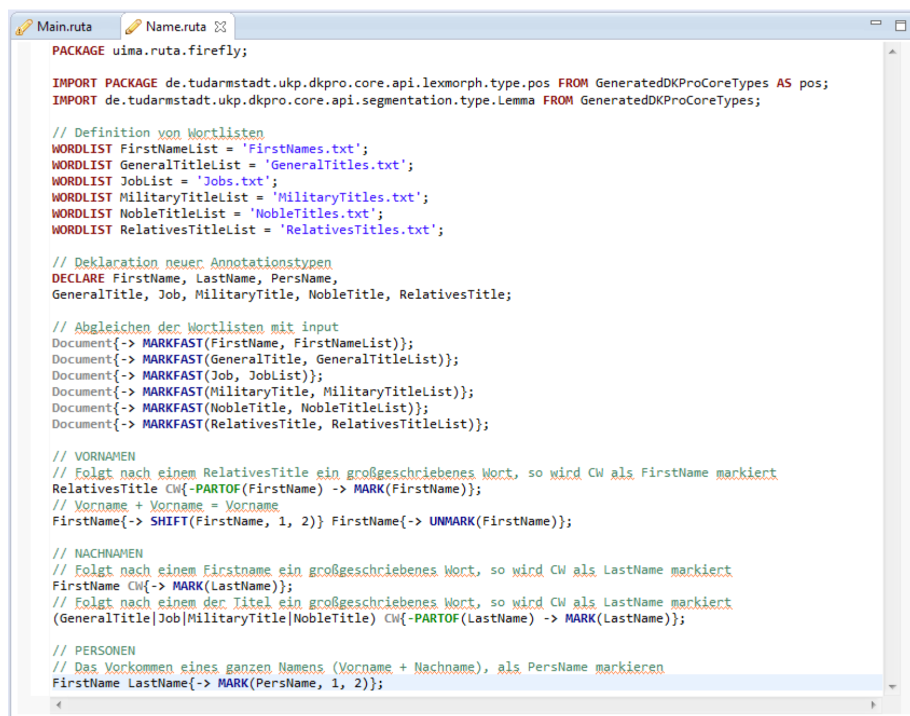

Abbildung 30: Die neuen Wortlisten werden eingefügt und die Regeln erweitert.

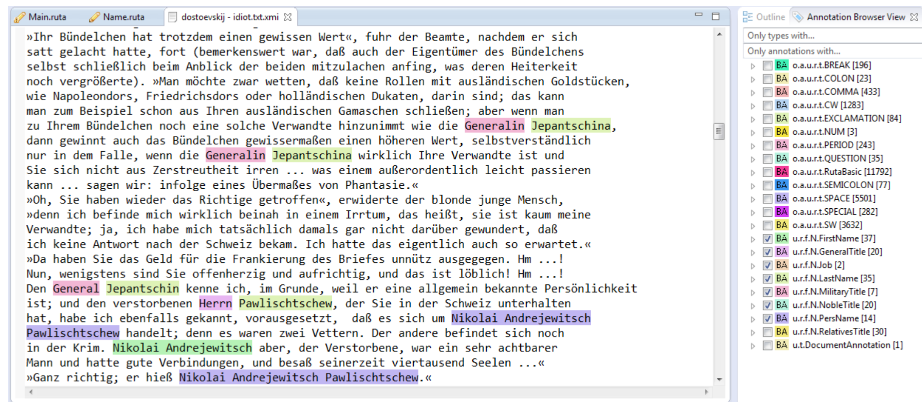

Abbildung 31: Das Ergebnis der Erweiterungen links im Fenster. Personen werden teilweise vollständig erkannt.

finden und beide Vornamen zu einem einzigen *FirstName* verbinden. `SHIFT` weitet dabei die Annotation des ersten *FirstName* auf das zweiten *FirstName* aus. Jetzt ist der zweite Vorname allerdings doppelt markiert und muss mit `UNMARK` wieder demarkiert werden. Im Nachnamenabschnitt suchen wir zunächst nach Vornamen mit einem angehängten, großgeschriebenen Wort. Trifft die Regel auf einen Abschnitt zu, wird das letzte Wort als *LastName* markiert. Mit der zweiten Regel suchen wir Personen, wie „Generalin Jepantschina". Folgt `CW` auf ein *MilitaryTitle* o.ä., wird das letzte Wort als *LastName* markiert. Dies geschieht wiederum nur, wenn das gefundene `CW` noch kein *LastName* ist (`-PARTOF`). Im letzten Abschnitt verbinden wir nun die *FirstName* und *LastName* zu einer Person als *PersName*. `MARK(PersName, 1, 2)` weitet die Annotation auf das erste und zweite Element in der Regel aus. Das Ergebnis unserer Entwicklung ist in Abb. 31 zu sehen.

Für den Ansatz, mit direkter Rede auf eine Person zu schließen, erstellen wir ein neues Skript. Darin befinden sich sämtliche Regeln, die eine direkte Rede erkennen und als *DirectSpeech* markieren (Abb. 32). Das Zeichen `#` stellt den gesamten Inhalt zwischen dem ersten und letzten Zeichen der Regel dar. Dies wäre also die gesamte Aussage einer Person. `MARK(DirectSpeech, 1, 2, 3)` markiert anschließend den gesamten Abschnitt als *DirectSpeech*.

Anschließend importieren wir das Skript mit `SCRIPT` in *Name.ruta* und führen es mit `CALL` aus. Für die Regel ist nun der zuvor eingerichtete TreeTagger notwendig. Mit diesem können wir nicht nur kleine und große Wörter erkennen, sondern sie auch einer Wortart zuordnen. Zusätzlich benötigen wir noch eine Liste mit Verben, die eine direkte Rede einleiten bzw. beenden. Dies wären beispielsweise Wörter wie „sagen" oder „antworten". Die Liste sollte als Wortliste definiert, aber nicht mit `MARKFAST` ausgeführt werden. Die Regel aus Abb. 33 wird dann am Ende des Personenabschnittes eingefügt. Diese markiert den Sprecher nach einer direkten Rede als *PersName*.

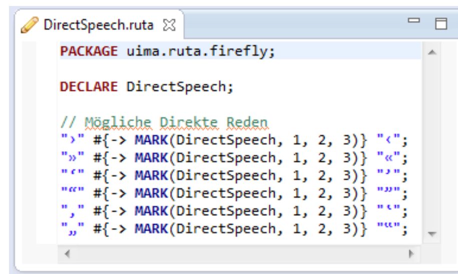

Abbildung 32: Das Skript zum Erkennen direkter Rede.

```
// Sprecher nach einer direkten Rede erkennen
DirectSpeech COMMA? pos.V{INLIST(VerbList, Lemma.value)} pos.ART CW+{-> MARK(PersName)};
```

Abbildung 33: Die Regel, um Sprecher nach einer direkten Rede zu erkennen.

Die Annotationstypen, die mit „pos.“ eingeleitet werden, sind von TreeTagger und reichen von Nomen und Verben, bis zu Personal- und Reflexivpronomen. Der gesamte Umfang der Annotationstypen des Part-of-speech Taggers ist in der entsprechenden Dokumentation[3] zu finden. Zunächst sucht UIMA Ruta nach einer direkten Rede und einem optionalen Komma. Danach muss ein Verb aus unserer Wortliste stehen. `INLIST` gleicht das Verb mit der Wortliste ab. Das Besondere dabei ist, dass der *Lemma.value* des gefundenen Wortes überprüft wird, nicht das Verb selbst. Somit steht in unserer Wortliste nur die Grundform eines Verbs, z.B. „sagen“, und nicht jede Deklination, z.B. „sagte“. Der *Lemma.value* eines Wortes ist also die Grundform. Jedes *Lemma.value* kann im *Selection View* eines geöffneten Dokumentes eingesehen werden, das mit dem TreeTagger bearbeitet wurde. Nach dem Verb folgt ein Artikel mit dem großgeschriebenen Wort. Das `CW` ist unsere gefundene Person und wird als *PersName* annotiert. Das Plus (`+`) ist ein Quantifizierer und drückt aus, dass mindestens ein `CW` folgt. Die Regel findet sowohl „Schwarzhaarige“, als auch „Schweizer Patient“.

6 Der Annotation Test

Der *Annotation Test* hilft uns dabei, unsere Regeln automatisch zu evaluieren. Bei sehr großen oder vielen Texten ist es nur sehr aufwendig, das Ergebnis manuell zu überprüfen. Der *Annotation Test* hingegen liefert uns einen genauen Überblick über unsere Trefferquote.

[3] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/stts_guide.pdf`

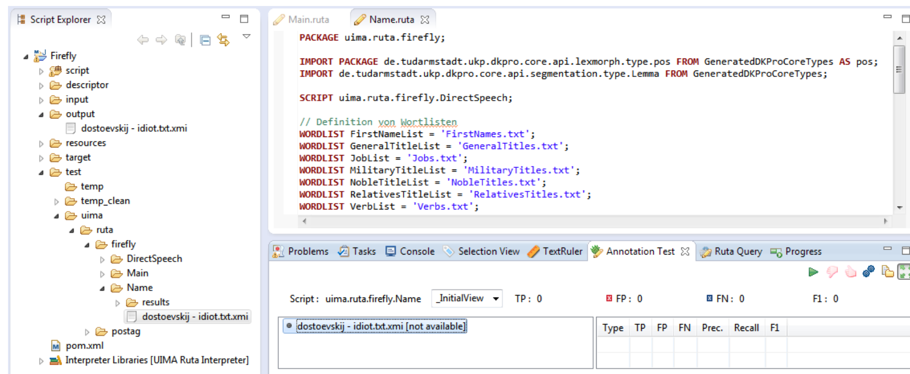

Abbildung 34: Im *Script Explorer* ist die Struktur des Testordner zu erkennen. Das relevante Skript muss angeklickt werden, damit unten der *Annotation Test* angezeigt wird.

6.1 Das Referenzdokument

Damit der Test überhaupt durchgeführt werden kann, benötigen wir ein Referenzdokument. In diesem Dokument müssen alle Annotationen vorhanden sein, die wir mit unserem Skript finden wollen. Bei der späteren Bearbeitung dieses Dokumentes müssen die entsprechenden Annotationstypen in der XMI-Datei vorhanden sein. Neue Annotationstypen können nicht erstellt werden. Deswegen starten wir das Skript *Main.ruta* und kopieren das Ausgabedokument aus *output* in den Unterordner *test* → *uima* → *ruta* → *firefly* → *Name*. Anschließend wählen wir das Skript *Name.ruta* an und klicken auf den Reiter *Annotation Test*. Die Arbeitsoberfläche sollte jetzt ähnlich der Abb. 34 sein.

Auf der linken Seite ist der *Script Explorer* mit allen Unterordnern von *test*. Das zu testende Dokument wird in den entsprechenden Ordner eingefügt. Testen wir also das Skript *Name.ruta*, kopieren wir das Referenzdokument in den Unterordner *Name*. Der Unterordner *results* enthält die Ergebnisdateien des *Annotation Test*. Neben den ursprünglichen Annotationstypen sind zusätzlich die Typen *TruePositive*, *FalsePositive* und *FalseNegative* gespeichert. *TruePositive* sind dabei die richtig markierten Abschnitte. Alle *FalsePositive* sind fälschlicherweise markiert, während hingegen alle *FalseNegative* fälschlicherweise nicht markiert sind. Die Anzahl dieser Typen ist im *Annotation Test* bei *TP*, *FP* und *FN* zu finden. Der *F1-Wert* liegt im Zahlenbereich zwischen 0 und 1. Umso höher der Wert, desto besser die Trefferquote. Wenn ein Test gestartet werden soll, wird der grüne Pfeil gedrückt. Da das derzeitige Referenzdokument eine exakte Kopie des Ausgabedokumentes ist, gibt es nur *TruePositives* und der F1-Wert ist genau 1. Wird einer der beiden Daumen gedrückt, erscheint das Auswahlfenster (Abb. 35), in dem sämtliche Annotationstypen ausgewählt werden können. Der „Daumen nach unten“ lässt uns Annotationstypen auswählen, die wir nicht testen wollen, wohingegen der „Daumen nach oben“ zwischen den relevanten Typen auswählen lässt.

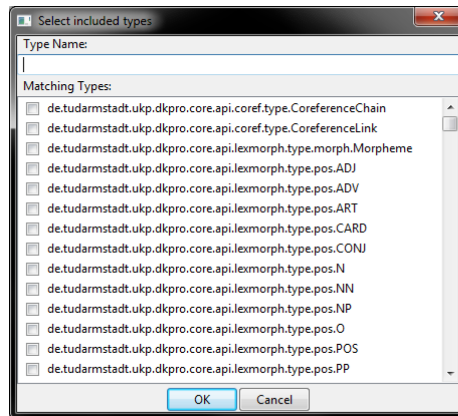

Abbildung 35: Auswahl relevanter oder irrelevanter Annotationstypen.

Daneben können bei den beiden Zahnrädern weitere Einstellungsmöglichkeiten vorgenommen werden (Abb. 36). Das Aktivieren von *Use all types* stellt sicher, dass zunächst alle Typen zum Test genutzt werden, außer man spezialisiert seine Suche mit Hilfe der Typenwahl (Abb. 35). Der Punkt *Extend classpath* sollte aktiviert sein, wenn wir TreeTagger nutzen, da sonst kein Test durchgeführt werden kann. Unter *Evaluator* kann die Art der Übereinstimmung gewählt werden. Bei *Exact Match* muss das Ergebnis exakt der Referenzannotation entsprechen, während *Partial Match* auch bei teilweise richtiger Markierung ein *TruePositive* liefert.

Das vorletzte Symbol aus Abb. 34 exportiert das Testergebnis als Tabelle in einer CSV-Datei, während das Letzte die Option *Extend classpath* symbolisiert. Im Moment haben wir noch kein nutzbares Referenzdokument, da wir lediglich das Ausgabedokument in den Testordner kopiert haben. Deswegen müssen wir das Dokument im Testordner mit allen richtigen Annotationen versehen. Dazu öffnen wir das Dokument mit einem Doppelklick und wählen im *Annotation Browser View* alle Annotationstypen aus, die wir bearbeiten wollen. In unserem Beispiel ist das *FirstName*, *LastName* und *PersName*, weil wir später nur die Trefferquote dieser Typen überprüfen wollen. Finden wir eine Person, müssen wir zunächst bestimmen, zu welchem Annotationstyp diese gehört. In Abb. 37 ist „Rogoschin“ weder als *LastName*, noch als *PersName* markiert. Wir annotieren das Wort mit **Doppelklick** → **Rechtsklick** → **Annotate** → **LastName** als *LastName*.

Es werden nur diejenigen Wörter als *PersName* markiert, die keine Namen sind, z.B. „Schwarzhaarige“. Dies spart uns in diesem Text viel Zeit, da wir im Anschluss mit *Quick Annotate* schneller arbeiten. Sind also alle Namen als *FirstName* bzw. *LastName* und jedes Synonym als *PersName* markiert, verändern wir den Annotationsmodus. Dazu klicken wir im Text auf **Rechtsklick** → **Mode** und navigieren zu *PersName*, wie in Abb. 38. Jetzt kann das gesamte Dokument überflogen werden, ohne jeden Satz lesen zu müssen. Jede vorhandene

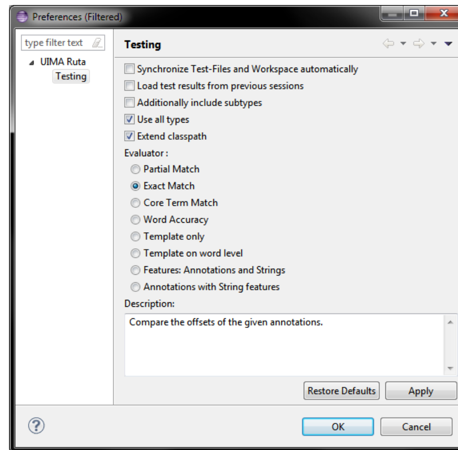

Abbildung 36: Einstellungen des *Annotation Test*.

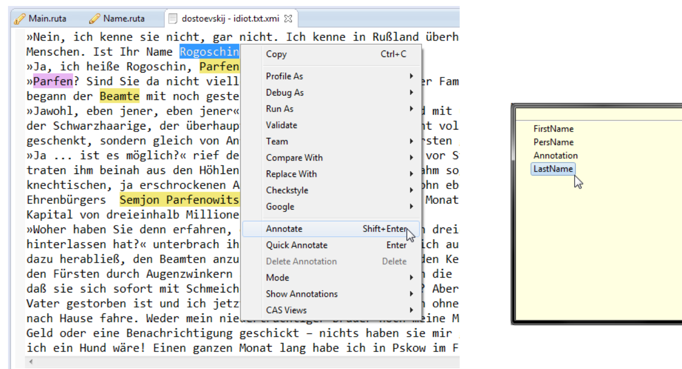

Abbildung 37: Manuelles Annotieren der Referenzdatei mit *Annotate*.

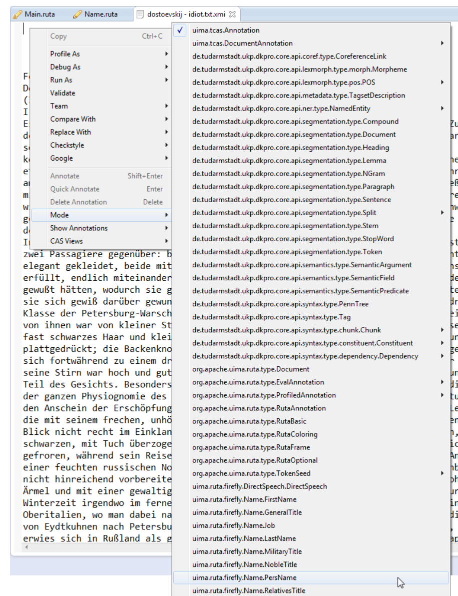

Abbildung 38: Ändern des Annotationsmodus über *Mode*.

Annotation wird markiert und mit **Enter** als *PersName* markiert. Natürlich sollen die zuvor markierten *PersName* nicht wiederholt markiert werden. Außerdem ist darauf zu achten, dass beispielsweise „Parfen Rogoschin“ als Ganzes markiert wird, nicht einzeln. Auch ist beispielsweise „Generalin Jepantschina“ ein *PersName*, da der General und die Generalin zwei unterschiedliche Personen sind.

Problems | Tasks | Console | Selection View | TextRuler | Annotation Test | Ruta Query | Progress

Script: uima.ruta.firefly.Name _InitialView TP: 95 FP: 14 FN: 165 F1: 0.515

dostoevskij - idiot.txt.xmi [95|14|165]

Type	TP	FP	FN	Prec.	Recall	F1
Total	95	14	165	0.872	0.365	0.515
uima.ruta.firefly.Name.FirstName	33	4	2	0.892	0.943	0.917
uima.ruta.firefly.Name.LastName	30	6	44	0.833	0.405	0.545
uima.ruta.firefly.Name.PersName	32	4	119	0.889	0.212	0.342

Abbildung 39: Der erste Test mit dem unangepassten Skript findet gerade 32 Personen.

Zusätzlich gibt es die Möglichkeit, eine Annotation zu entfernen. Dies ist notwendig, da wir das Referenzdokument aus dem *output* Ordner kopiert haben und dort möglicherweise falsche Annotationen vorhanden sind. Die erste Option ist es, in den jeweiligen Annotationsmodus zu wechseln und die Annotation mit **Entf** bzw. **Del** zu entfernen. Bei der zweiten Option müssen wir die jeweilige Annotation im *Annotation Browser View* suchen und ebenfalls mit **Entf** löschen.

6.2 Fehleranalyse und Verbesserung

Ist das gesamte Dokument vollständig mit Annotationstypen versehen, kann der *Annotation Test* gestartet werden. Dazu wählen wir nur die Typen *FirstName*, *LastName* und *PersName* aus und starten den Test. Das Ergebnis ist in Abb. 39 zu sehen.

Der F1-Wert liegt bei über 0.5. Wenn wir jedoch den Wert in der Zeile von *PersName* betrachten, sehen wir, dass wir nur knapp 20 % der Personen gefunden haben. Wenn wir wissen wollen, woran das liegt und wie das verbessert werden kann, öffnen wir das Ergebnisdokument mit einem Doppelklick im Fenster links. Im *Annotation Browser View* markieren wir die Typen *TruePositive*, *FalsePositive* und *FalseNegative*, um uns einen ersten Überblick des Ergebnisses zu verschaffen. Außerdem ordnen wir die Ansicht der Fenster per Drag&Drop um, sodass wir schnell eine Liste der Treffer haben (Abb. 40).

Wir sehen, dass viele Namen zuverlässig gefunden wurden, aber die gleiche Person an einer anderen Textstelle nicht. Das liegt daran, dass wir beispielsweise „Herr Rogoschin“ finden, nicht aber ein einzeln stehendes „Rogoschin“. Des Weiteren gibt es auch noch keine Regel die eine Abwandlung der Namen, wie „Rogoschins“, findet. Um das zu beheben gibt es nicht nur statische Wortlisten, sondern auch dynamische Stringlisten. Diese werden mit dem Befehl `STRINGLIST` erstellt und beinhalten `STRING` Elemente, die wir in die Regeln einbauen werden. Das vollständige Skript ist im Quelltext 1 zu sehen.

UIMA Ruta Quelltext 1: Das finale Skript Name.ruta.

```
PACKAGE uima.ruta.firefly;

IMPORT PACKAGE de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos
FROM GeneratedDKProCoreTypes AS pos;
IMPORT de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.Lemma
FROM GeneratedDKProCoreTypes;

SCRIPT uima.ruta.firefly.DirectSpeech;

// Definition von Wortlisten
WORDLIST FirstNameList = 'FirstNames.txt';
WORDLIST GeneralTitleList = 'GeneralTitles.txt';
WORDLIST JobList = 'Jobs.txt';
WORDLIST MilitaryTitleList = 'MilitaryTitles.txt';
WORDLIST NobleTitleList = 'NobleTitles.txt';
WORDLIST RelativesTitleList = 'RelativesTitles.txt';
WORDLIST VerbList = 'Verbs.txt';

// Definition von Stringlisten und Stringvariable
STRINGLIST FirstNames;
STRINGLIST LastNames;
STRINGLIST PersNames;
STRING MatchedName;

// Deklaration neuer Annotationstypen
DECLARE FirstName, LastName, PersName,
GeneralTitle, Job, MilitaryTitle, NobleTitle, RelativesTitle;

// Importierte Skripte ausfuehren
Document{-> CALL(DirectSpeech)};

// Abgleichen der Wortlisten mit input
Document{-> MARKFAST(FirstName, FirstNameList)};
Document{-> MARKFAST(GeneralTitle, GeneralTitleList)};
Document{-> MARKFAST(Job, JobList)};
Document{-> MARKFAST(MilitaryTitle, MilitaryTitleList)};
Document{-> MARKFAST(NobleTitle, NobleTitleList)};
Document{-> MARKFAST(RelativesTitle, RelativesTitleList)};

// VORNAMEN
// Folgt nach einem RelativesTitle ein grossgeschriebenes Wort,
// so wird CW als FirstName markiert und in die Liste FirstNames gespeichert
RelativesTitle pos.N{-PARTOF(FirstName) ->
MARK(FirstName), MATCHEDTEXT(MatchedName), ADD(FirstNames, MatchedName)};
// An alle Vornamen ein "s" haengen und in FirstNames speichern
FirstName{-> MATCHEDTEXT(MatchedName), ADD(FirstNames, MatchedName + "s")};
// Vornamen aus der Liste FirstNames markieren
CW{-PARTOF(FirstName), INLIST(FirstNames) -> MARK(FirstName)};
// Vorname + Vorname = Vorname
FirstName{-> SHIFT(FirstName, 1, 2)} FirstName{-> UNMARK(FirstName)};

// NACHNAMEN
// Folgt nach einem Firstname ein grossgeschriebenes Wort,
// so wird CW als LastName markiert und in die Liste LastNames gespeichert
FirstName pos.N{->
MARK(LastName), MATCHEDTEXT(MatchedName), ADD(LastNames, MatchedName)};
// Folgt nach einem der Titel ein grossgeschriebenes Wort,
// so wird CW als LastName markiert
(GeneralTitle|Job|MilitaryTitle|NobleTitle) pos.N{-PARTOF(LastName)
-> MARK(LastName), MATCHEDTEXT(MatchedName), ADD(LastNames, MatchedName)};
// An alle Nachnamen ein "s" haengen und in LastNames speichern
LastName{-> MATCHEDTEXT(MatchedName), ADD(LastNames, MatchedName + "s")};
// Nachnamen aus der Liste LastNames markieren
CW{-PARTOF(LastName), INLIST(LastNames) -> MARK(LastName)};

// PERSONEN
// Das Vorkommen eines ganzen Namens (Vorname + Nachname),
// als PersName markieren
(RelativesTitle|GeneralTitle|Job|MilitaryTitle|NobleTitle)* @FirstName
LastName{-> MARK(PersName, 1, 2, 3)};
(RelativesTitle|GeneralTitle|Job|MilitaryTitle|NobleTitle)* @FirstName
{-PARTOF(PersName) -> MARK(PersName, 1, 2)};
(RelativesTitle|GeneralTitle|Job|MilitaryTitle|NobleTitle)* @LastName
{-PARTOF(PersName) -> MARK(PersName, 1, 2)};
// Sprecher nach einer direkten Rede erkennen
DirectSpeech COMMA? pos.V{INLIST(VerbList, Lemma.value)} pos.ART CW+
{-> MARK(PersName), MATCHEDTEXT(MatchedName), ADD(PersNames, MatchedName)};
// Personen aus der Liste PersNames markieren
CW{-PARTOF(PersName), INLIST(PersNames) -> MARK(PersName)};
```

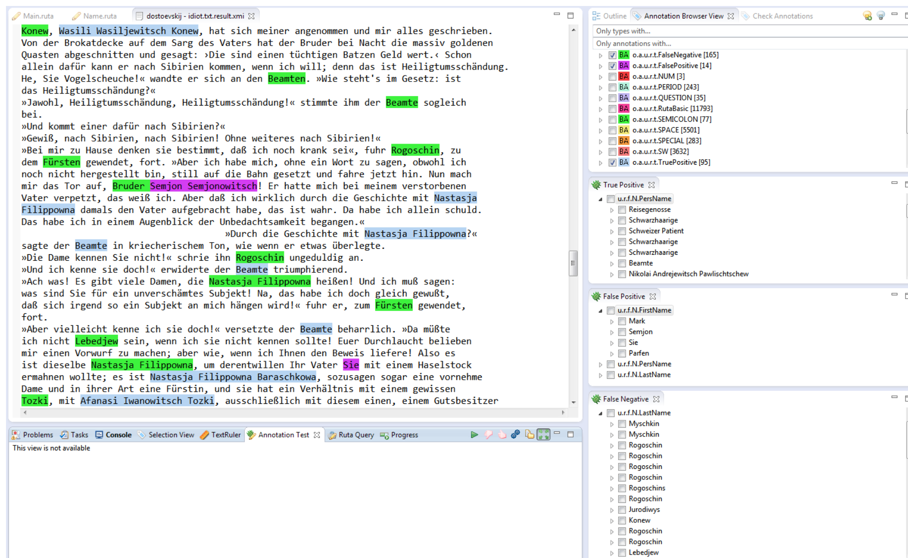

Abbildung 40: Im linken Fenster sind alle *TruePositive*, *FalsePositive* und *FalseNegative* markiert. In der rechten Ansicht ist eine detaillierte Übersicht der (nicht) gefundenen Vornamen, Nachnamen und Personen.

Für jeden Abschnitt haben wir eine Stringliste definiert, da wir diese am Ende jedes Abschnitts verwenden. In die Stringvariable schreiben wir einen gefundenen Namen bzw. Person, um ihn in eine Liste speichern zu können. Die erste Regel wurde so erweitert, dass mit `MATCHEDTEXT` der potenzielle Vorname in den String *MatchedName* gespeichert wird. Anschließend fügen wir mit `ADD` den gefundenen Vornamen in unsere Stringliste, in die alle Vornamen gespeichert werden, die nicht in der Wortliste *FirstNameList* sind. Zusätzlich wurde `CW` durch `pos.N` ersetzt, da wir sonst auch Wörter wie „Sie" finden würden, das zu den Pronomen gehört und nicht zu den Nomen. Im zweiten Schritt suchen wir nochmals nach Vornamen, schreiben diese aber mit einem angehängten „s" in die Stringliste, um so auch kleine Abwandlungen von Namen zu finden. Die nächste neue Regel untersucht alle `CW` im Text und prüft mit `INLIST`, ob es in unserer Liste *FirstNames* gibt. Bei jeder Regel ist zu beachten, dass wir einen schon markierten Abschnitt nicht doppelt annotieren. Im Skript *Name.ruta* verhindern wir mögliche Probleme, da `-PARTOF` dies ausschließt. Der zweite Abschnitt ist nach dem selben Prinzip erweitert worden, wie der Erste. Potenzielle Nachnamen werden in eine Liste geschrieben, Abwandlungen werden gespeichert und zum Schluss annotiert. Der letzte Abschnitt fügt Titel, Vornamen und Nachnamen zu einer Person zusammen. Jeder Annotationstyp, der mit einem Stern (`*`) endet, kann keinmal oder mehrmals vorkommen. Das Zeichen `@` vor einem Annotationstyp heißt, dass UIMA Ruta zuerst nach diesem im Text sucht und dann prüft, ob die restliche Regel zutrifft. Bei richtiger Nutzung führt dies zu

Problems Tasks Console Selection View TextRuler Annotation Test Ruta Query Progress

Script: uima.ruta.firefly.Name _InitialView TP: 216 FP: 21 FN: 44 F1: 0.869

dostoevskij - idiot.txt.xmi [216|21|44]

Type	TP	FP	FN	Prec.	Recall	F1
Total	216	21	44	0.911	0.831	0.869
uima.ruta.firefly.Name.FirstName	33	3	2	0.917	0.943	0.93
uima.ruta.firefly.Name.LastName	63	9	11	0.875	0.851	0.863
uima.ruta.firefly.Name.PersName	120	9	31	0.93	0.795	0.857

Abbildung 41: Das Endergebnis des Beispiels zeigt eine Steigerung der (korrekt) gefundenen Personen von 32 auf 120.

einer Geschwindigkeitssteigerung. Personen nach direkten Reden werden ebenfalls in einer Liste gespeichert und anschließend im gesamten Text gefunden. Wenn das Skript erneut getestet wird, können wir eine erhebliche Steigerung des Ergebnisses erkennen (Abb.41).

Mit den vorgestellten Techniken von UIMA Ruta, der Textanalyse, TreeTagger und dem *Annotation Test* kann sehr schnell eine hohe Trefferquote erzielt werden. Mit Hilfe des *Annotation Test* muss zum Schluss jede Regel weiter verfeinert werden, bis ein akzeptables Niveau erreicht ist.