JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

Musikspezifische Informationsextraktion aus Webdokumenten

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplomingenieur

im Masterstudium

Informatik

Eingereicht von:
Andreas Krenmair

Angefertigt am:
Institut für Computational Perception

Beurteilung:
Univ.-Prof. Dr. Gerhard Widmer

Mitwirkung:
Dipl. Ing. Peter Knees
Dr. Markus Schedl

Linz, Mai 2010

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Linz, am 23. Mai 2010

Andreas Krenmair

Danksagung

Vor allem bedanken möchte ich mich bei Peter Knees, welcher es mir ermöglicht hat, in diesem interessanten Gebiet zu arbeiten und mir dabei sehr viel Freiraum eingeräumt hat. Aber auch bei Markus Schedl, welcher während Peters Abwesenheit immer mit zuverlässigen Informationen zur Verfügung stand. Weiters möchte ich mich bei meinen Freunden bedanken, welche für den nötigen Ausgleich zu dieser Arbeit gesorgt haben. Ebenfalls bin ich meiner Tante Mag. [a] Heidemarie Gratz zu großem Dank verpflichtet, welche die aufwendige Arbeit des Korrekturlesens der ersten Version übernommen hat. Zuletzt möchte ich meinen Eltern ein herzliches Dankeschön ausprechen, die mir nicht nur finanziell dieses Studium ermöglicht haben.

Kurzfassung

Neben content-based features, also charakteristischen Kennzahlen, die aus dem Audiosignal extrahiert werden, kann man auch context-based features zur Gewinnung von Informationen über Musik verwenden. Diese Features für die Berechnung von Ähnlichkiten beruhen hauptsächlich auf kulturell geprägten Informationen. Um diese Informationen zu erhalten, werden geeignete Ansätze benötigt. Genau hier setzt das Forschungsfeld des Information Extraction (IE) an, welches verschiedene Strategien zur Verfügung stellt, um context based features aus umgangssprachlichen Texten zu erhalten.

Diese Arbeit bschäftigt sich mit der Entwicklung einer IE Architektur, welche automatisch Musik-Entitäten wie Interpreten, Bandmitglieder und Banveröffentlichungen mittels eines regelbasierten Ansatzes aus einer Menge von Webseiten extrahiert. Aufbauend auf der zugrundeliegenden Architektur, ermöglicht das System die Analyse von Webseiten und die Erstellung von Beziehungsnetzwerken. Um die Ergebnisse zu verbessern, besteht die Möglichkeit, automatisch neue Regeln, aus Webseiten, zu generieren.

Da die Verwendung von IE zu ziemlich fehlerbehafteten Resultaten führen kann, konzentriert sich ein wesentlicher Teil dieser Arbeit auf der Evaluierung der Ausgabe des entwickelten Systems. Vor allem die Relationen des Beziehungsnetzwerks werden im Zuge dieser Arbeit ausführlich ausgewertet.

Abstract

Besides content-based features for analysis of music, one can use context-based features to obtain music related information. These features rely heavily on culturally related information. The lack of proper information calls for new ways of retrieving context-based features. The research area of Information Extraction (IE) offers various concepts to obtain this particular kind of information from natural language texts.

This thesis discusses an IE architecture that automatically extracts music entities like artists, band members and band releases from a collection of web pages using a rule-based approach. On the basis of the underlying architecture, the system is capable of analysing webpages and building relational networks. In order to improve the results, a modified version of the system creates new extraction rules based on Web pages retrieved via a search engine.

Since IE can produce rather defective results, a major task of this thesis is to evaluate the output of the developed system. Especially the relations of the relational network are thoroughly evaluated.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface

GUI Graphical User Interface

GATE General Architecture for Text Engineering

HTML HyperText Markup Language

IE Information Extraction

IR Information Retrieval

ISO International Organization for Standardization

LE Language Engineering

LHS Left Hand Side

NLP Natural Language Processing

POS Part of Speech

RHS Right Hand Side

RTF Rich Text Format

SGML Standard Generalized Markup Language

UTF Unicode Transformation Format

XML Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Das Forschungsfeld des Music Information Retrieval (MIR) beschäftigt sich vor allem mit der Analyse von und Extraktion von Informationen aus Musikstücken. Anhand verschiedener Algorithmen können Lieder automatisch einem Genre zugeordnet oder auch deren Ähnlichkeit errechnet werden. Jedoch konzentrieren sich diese Verfahren hauptsächlich auf die Verarbeitung von Audiosignalen. Die kulturelle Komponente hinter den einzelnen Musikstücken wird dabei großteils vernachlässigt. Populäre Audioformate bieten die Möglichkeit einzelne Musikstücke mit diversen Metainformationen anzureichern. Häufig existieren aber nur sehr wenige Informationen zu dem jeweiligen Track. Eine Vielzahl zusätzlicher Informationen lässt sich allerdings im Internet finden. Genau hier setzt das Konzept des Information Extraction (IE) an. Dabei werden zahlreiche Dokument maschinell verarbeitet mit dem Ziel spezifische Informationen daraus zu erhalten. Beispielsweise lässt sich so das Genre anhand des Namens des Interpreten bestimmen. Dies ist aber nur eine der vielen Anwendungsmöglichkeiten des IE in diesem Kontext. Auch die Bestimmung von Ähnlichkeiten zwischen Tracks bzw. Interpreten ist damit möglich. Zudem können die extrahierten Informationen teilweise als Metadaten in Musikanwendungen verwendet werden. In dieser Arbeit konzentriert sich der Fokus auf die Extraktion von bandspezifischen Informationen. Aus einer Menge an relevanten Dokumente wird versucht Namen von Artists bzw. Bands, Bandmitglieder und Medien (Alben, Singles, etc.) zu extrahieren. Diese Namen bzw. Entitäten findet man grundsätzlich in den einzelnen Sätzen eines Dokuments (unstrukturierte Daten). Ein Besipiel dafür wäre folgender Absatz:

W. Axl Rose (born William Bruce Rose, Jr. on February 6, 1962 in Lafayette, Indiana) is an American musician, and the lead vocalist of hard rock band Guns N' Roses. The band's major label debut album, entitled Appetite for Destruction, was released in the United States on July 21, 1987. [1]

[1] `http://en.wikipedia.org/wiki/Axl_Rose`, besucht am 20. März 2010

Aus diesen zwei Sätzen lässt sich dabei einiges an Wissen ableiten. Zum einem der Name W. Axl Rose, dessen Rolle in der Band (lead vocalist), der Bandname (Guns N' Roses), sowie ein Album der Gruppe (Appetite for Destruction). Kombiniert man diese Fragmente, besteht die Möglichkeit ein Beziehungsnetzwerk oder ein Bandbiographie automatisch zu generieren.

In dieser Arbeit wird eine Architektur vorgestellt, welche diese musikspezifische Entitäten erkennt und weiterverarbeitet. Da aber diese Ausprägungen vor allem in ihrer Struktur sehr variieren, liegt der Schwerpunkt in der Erarbeitung einer geeigneten Erkennungsstrategie. Ausgehend von dieser Basisarchitektur werden zwei mögliche Anwendungen präsentiert, welche Grundfunktionalität der Basisarchitektur erweitern. Dem folgt eine Evaluierung des Systems. Abgeschlossen wird die vorliegende Arbeit mit einer kritischen Beurteilung der verwendeten Konzepte.

Kapitel 2

Information Extraction

In der heutigen Zeit sieht sich der Mensch mit einer schier nicht endend wollenden Flut an Informationen konfrontiert. In der Regel werden jene Informationen von uns verarbeitet und anschließend bewertet. Im Gegensatz zu Computern, stellt es für Menschen kein Problem dar, natürlichsprachige Texte sinnerfassend zu verarbeiten und relevante (implizite) Informationen zu extrahieren.

Allerdings wird dieser Vorgang durch verschiedene Aspekte beeinflusst. Jede Person bewertet Informationen unterschiedlich. Zudem nimmt nach einiger Zeit die Aufmerksamkeitspanne ab, welche Einfluss auf die Qualität der Informationen haben kann. Deswegen ist es nicht verwunderlich, dass man versucht diesen Prozess zu automatisieren, denn nur so ist es möglich eine Vielzahl von Dokumenten in einer angemessenen Zeit zu verarbeiten. Allerdings ist die formale Verarbeitung von Texten ein sehr komplexer Prozess. Das liegt vor allem daran, dass die Grammatik einer Sprache nahezu unendlich viele Möglichkeiten bietet, Informationen zu kodieren. Maschinensprachen hingegen unterliegen einen strikt und eng definierten Syntax, welche das Verarbeiten von Zeichen erst ermöglicht [41]. Eine vollständige Verarbeitung von natürlicher Sprache ist in den meisten Fällen also nicht sinnvoll bzw. auch nicht notwendig. Automatisierte Systeme konzentrieren sich in der Regel auf einzelne Eigenschaften von Texten.

2.1 Einordnung

Grundsätzlich lässt sich diese Problemstellung in den Bereich der Natural Language Processing, also der Verarbeitung von natürlichen Sprachen, einordnen. Natural Language Processing umfasst folgende Aufgabenbereiche [20]:

- Speech Synthesis

- Speech recognition
- Natural language unterstanding
- Natural language generation
- Machine translation

Dabei ist Information Extraktion in den Bereich der Natural Language Unterstanding einzuordnen. Allerdings werden einige NLP Techniken aus unterschiedlichen Bereichen im Extraktionsprozess verwendet, um die gegebenen Dokumente zu vorverarbeiten.

2.2 Definition

Eine ähnliche Kategorie von Information Extraktion ist das Gebiet des Information Retrieval. Obwohl sich diese beiden Kategorien vor allem durch ihre Ziele unterscheiden, besteht eine enge Verbindung zwischen diesen beiden Kategorien. Line Eikvil [28] grenzt diese beiden Begriffe folgendermaßen ab:

- ”IR retrieves relevant documents from collections”
- ”IE extracts relevant information from documents”

Sind jedoch keine relevanten Dokumente vorhanden, muss dem IE Prozess wiederum ein IR Prozess vorangehen. In der Regel sind die erforderlichen Dokumente aber bereits vorhanden. Grundsätzlich beschränkt sich der Information Extraktion Prozess auf eine Domäne, eine Applikation, die im Grunde genommen auf ausschließlich einen Fachbereich (also z.B. Medizin, Pharmazie, usw.) zugeschnitten ist.

Kapitel 3

Problemstellung

Generell beschäftigt sich die vorliegende Arbeit mit der Erstellung eines IE Systems, welches musikspezifische Daten aus einer Sammlung von Dokumenten extrahiert. Das Hauptaugenmerk liegt vor allem auf der Extraktion von Bandnamen, Bandmitgliedern und Medieninhalten einer Band. Letztere können in weitere Unterklassen (Album, Single, Compilation, etc.) unterteilt werden. Trotz der Fokussierung auf diese drei Hauptentitäten, muss das einfache Hinzufügen von neuen Entitäten unbedingt gewährleistet bleiben. Zudem muss die Architektur so ausgelegt sein, dass der Entwickler sowohl eine grafische Oberfläche als auch eine Applikation für eine rasche Verarbeitung von sehr vielen Dokumenten darüberlegen kann. Zusätzlich kann man die gewonnenen Informationen für weitere Verarbeitungsschritte speichern bzw. verwenden. Aus dieser groben Beschreibung kristallisieren sich folgende Problemstellungen:

Architektur: Generell existieren zwei Hauptkomponenten in jeder IE Architektur. Zum einem müssen die einzelnen Dokumente in adäquater Weise verwaltet werden. Diese Aufgabe übernimmt die **Corpus** Komponente, deren Texte im Anschluss vom System verarbeitet werden müssen. Diese Aufgabe wiederum übernimmt die **Pipeline** Komponente. Diese zweite Komponente setzt sich aus verschiedenen Verarbeitungsstationen zusammen, welche generell von den Anforderungen an das System diktiert werden. Die Eigenschaften einer IE Archietktur werden in Kapitel 4 gesondert besprochen.

Wissensbasis: Die Architektur wird um eine Wissensbasis erweitert. Jene speichert die extrahierten Informationen und unterstützt das System beim Extraktionsprozess.

Extraktion: Wohl die schwierigste Aufgabe ist die Extraktion der gewünschten Informationen. Dieser Prozess kann auf unterschiedliche Art und Weise durchgeführt werden. Es hängt aber immer davon ab, wie komplex diese Informationen sind.

Evaluierung: Das implementierte System muss in einer angemessenen Weise bewertet werden. Generell hängt die Qualität des Systems beinahe ausschließlich von der Extraktionskomponente ab. Man misst also, wie gut diese für eine vorliegende Menge an Dokumenten funktioniert.

Für diese Probleme werden im Anschluss verschiedene Lösungsansätze präsentiert. Aus der Menge an Strategien ergeben sich in Kapitel 5.1 die Eigenschaften des vorliegenden Systems. Der Beschreibung dieser Probleme geht aber noch eine Auflistung von Herausforderungen, mit denen sich ein IE System konfrontiert sieht, voraus.

3.1 Allgemeine Herausforderungen

Ein IE System unterliegt sehr vielen Faktoren, welche den Erfolg des Systems beeinflussen können. Diese werden in [46] veranschaulicht und im Folgenden kurz behandelt.

3.1.1 Treffergenauigkeit

Die wichtigste Herausforderung ist wohl die Qualität und Quantität der extrahierten Informationen. Das System muss sowohl so viele relevante Fakten wie möglich abdecken, als auch eine niedrige Fehlerquote aufweisen. Dieses Ziel wird durch unterschiedliche Faktoren beeinflusst.

1. Die Erkennung von Information beruht stark auf gewissen Mustern im Text. Je nachdem welche Struktur bzw. grammatikalische Eigenschaften die Wörter aufweisen, kann abgeschätzt werden, ob es sich um eine relevante Information handelt oder nicht. Vor allem die Komplexität eines Textes macht diese Aufgabe besonders schwierig.

2. Natürlich können die relevanten Teile eines Dokuments nicht zu 100 Prozent erkannt werden. Einige Informationen verstecken sich entweder in komplexen Strukturen bzw. existieren keine aussagekräftigen Anhaltspunkte, um diese zu erfassen. Um auch diese Teile abzudecken, muss zusätzlich zu den Extraktionsregeln, eine weitere Strategie verwendet werden.

3. Einige Informationen verstecken sich teilweise in mehreren Sätzen. Die Problematik besteht darin, diese Fakten zu finden und miteinander zu verknüpfen.

3.1.2 Laufzeit

Ein IE System besteht aus unterschiedlichen Verarbeitungseinheiten. Bei der Verarbeitung werden die Dokumente analysiert und mit Metainformationen versehen. Jene Schritte schlagen sich wiederum in der Laufzeitkomplexität nieder. Die Schwierigkeit besteht darin sowohl eine angemessene Performance als auch zufriedenstellende Ergebnisse zu liefern.

3.1.3 Daten

Die Eigenschaften des Internets erlauben dem Entwickler in moderater Zeit eine große Menge an Daten bzw. Dokumenten zusammen zu tragen. Generell kann man das Internet als riesiges Informationsdepot betrachten. Alleine Seiten wie Wikipedia beinhalten eine Menge an Fakten. In der Regel sind diese auch immer aktuell. Tritt ein gewisses Ereignis auf, wird die zugehörige Wikipedia Seite bereits nach kurzer Zeit aktualisiert. Dies führt oft zu regelrechten Wettkämpfen, wenn ein populärer Eintrag geändert werden muss. Aber auch Webseiten von Tageszeitungen und Nachrichtensendern stellen generell immer aktuelle Dokumente zur Verfügung. Jedoch führt dies auch zum Problem, dass zu einem früheren Zeitpunkt bezogene Daten womöglich schon überholt sein könnten. Zudem können die einzelnen Seiten auch Falschinformationen beinhalten.

3.2 Trainingscorpus

Ein fundamentales Element der Arbeit ist auch der Aufbau eines Trainingscorpus. Dieser kann für verschiedene Aufgaben herangezogen werden:

1. Er kann für regelbasierte Ansätze verwendet werden. Diese Regeln werden automatisch oder durch einen Menschen erstellt.

2. Er kann für statistische Ansätze verwendet werden. Dafür muss der vorerst Corpus von einem Menschen annotiert werden. In Folge dessen, ist es möglich statistische Modelle für die Erkennung von Strukturen wie Bandnamen und andere Entitäten zu trainieren.

Zum Zweck der Evaluierung dieser Arbeit gibt es zwei Arten von Korpora: Einen Trainings- und einen Testcorpus. Der Trainingscorpus wird für die Erstellung der Extraktionsregeln verwendet, wobei man den Testcorpus für die Evaluierung heranziehen

kann. Damit wird verhindert, dass die Regeln zu stark auf den vorliegenden Trainingscorpus ausgelegt sind ("overfitting"). Desweiteren muss überlegt werden, aus welchen Dokumenten der Corpus besteht. Je nachdem ob es sich um strukturierte, semi- oder gänzlich unstrukturierte Texte handelt, wirkt sich dies unterschiedlich auf den Prozess der Extraktion aus. Hinsichtlich der oben festgelegten Punkte, können folgende Anforderungen an den Testcorpus abgeleitet werden:

1. Er muss eine adäquate Größe haben. Dies ist sowohl für die Qualität der jeweiligen Extraktionsstrategie als auch für die Evaluierung wichtig.

2. Die Dokumente müssen von unterschiedlichen Quellen bezogen werden. Das verringert die Gefahr, dass die Regeln zu stark an eine Struktur adaptiert werden.

3. Zusätzlich muss der Testcorpus Entitäten mit unterschiedlicher Komplexität, sowie Artists aus unterschiedlichen Genres beinhalten.

4. Der Trainingscorpus sollte ähnliche Eigenschaften wie der Testcorpus aufweisen.

Zu den oben genannten Punkten kommen noch Qualitätsmerkmale wie Korrektheit und Aktualität hinzu.

3.3 Extraktion

Der Hauptzweck eines IE Systems ist das Erkennen und Markieren von einzelnen Strukturen. Wenn der Vorgang der Extraktion keine brauchbaren Ergebnisse liefert, kann der Anwender keinen Nutzen aus dem System ziehen. In diesem Kapitel werden die verschiedenen Möglichkeiten Informationen zu extrahieren vorgestellt. Zunächst werden aber die einzelnen Ausprägungen von Informationen, wie diese im Text vorkommen könnten, besprochen.

3.3.1 Informationskategorien

Die einzelnen Informationen können in mannigfaltiger Weise in den einzelnen Dokumenten vorkommen. Je nachdem welche Anforderungen an das IE System gestellt werden, können Informationen von unterschiedlicher Komplexität gewonnen werden. Sunita Sarawagi [46] unterscheidet dabei zwischen folgenden Kategorien:

3.3.1.1 Entitäten

Die einfachste Form von extrahierter Information sind Entitäten. Dabei kann man zwischen verschiedene Ausprägungen von Entitäten unterscheiden. Es kann sich sowohl um einfache numerische Ausdrücke, aber auch um komplexere Strukturen, wie Datumsangaben handeln. Am Häufigsten werden jedoch Namen extrahiert, welche grundsätzlich aus einer Sequenz von Eigennamen bestehen. Da der Begriff Name doch etwas allgemein formuliert ist, findet dabei eine Unterteilung in verschiedene Namensklassen statt. Welche Ausprägung eines Namens dann tatsächlich extrahiert wird, hängt aber wiederum von der Domäne ab.

3.3.1.2 Aufzählungen, Tabellen

Im Grunde genommen bestehen solche Strukturen aus einer Menge von Entitäten, welche nach einem bestimmten Schema angeordnet sind. Wobei Aufzählungen generell aus Entitäten der gleichen Ausprägung bestehen, wie zum Beispiel Listen von Bandmitgliedern. Tabellen wiederum, können aus verschiedenen Entitätsklassen bestehen. In Tabelle 3.1 [1] wird eine solche Struktur dargestellt.

Date	Venue	City
May 1, 2010	Sheffield Arena	Sheffield
May 2, 2010	Newcastle Arena	Newcastle
May 4, 2010	Liverpool Arena	Liverpool
May 5, 2010	LG Arena	Birmingham
May 7, 2010	O2 Arena	Dublin
May 9, 2010	SE+CC	Glasgow

Tabelle 3.1: Strukturierte Aufzählung der KISS Sonic Boom Tour 2010

Grundsätzlich besteht dabei eine gewisse Beziehung zwischen den einzelnen Feldern. Die Detektion und Extraktion von Verknüpfungen wird im nächsten Kapitel etwas näher erläutert. Die Hauptproblematik bei solchen Strukturen liegt in ihrer Form, denn sie kann von Dokument zu Dokument variieren. Eine adaptive Methode um Entitäten aus Listen zu extrahieren wird in [29] vorgeschlagen.

[1] `http://www.kissonline.com/tour/`, besucht am 23. März 2010

3.3.1.3 Verknüpfungen

Verknüpfungen erfordern eine komplexere Form der Informationsextraktion. Sie bestehen aus zwei oder mehreren Entitäten, wobei man auch hier zwischen Verknüpfungen unterschiedlicher Komplexität unterscheiden kann. Die einfachste Variante sind wohl die **isA** Beziehungen [50]. Dabei wird eine Entität gewissen Klassen zugeordnet, wie zum Beispiel das Muster:

drummer <Member>

Daraus kann nun die Beziehung "Lars Ulrich is a drummer" abgeleitet werden. Bei etwas komplexeren Verknüpfungen werden mehrere Entitäten durch fest vorgelegte Definitionen miteinander verbunden, wie zum Beispiel:

<Member> joined <Band>

Ein Beispiel für eine komplexere Verknüpfung wäre zum Beispiel ein Konzert einer Band. Für diese Situation müssen Entitäten, wie Datum, Ort und optional Supportkünstler zusammengeführt werden.

3.3.2 Extraktionsstrategien

Die Designentscheidungen beim Entwurf eines IE System werden maßgeblich von der verwendeten Extraktionsstrategie bestimmt. Allgemein kann man zwischen zwei verschiedenen Ansätzen unterscheiden. Dem Knowledge Engineering Approach und dem Automatic Training Approach, wobei jeder Ansatz seine Stärken und seine Schwächen hat. Zunächst werden die einzelnen Ansätze gesondert vorgestellt. Die beiden Strategien werden im Anschluss gegenübergestellt und ihre Stärken bzw. Schwächen identifiziert.

3.3.2.1 Knowledge Engineering Approach

Der Knowledge Engineering Approach verlässt sich grundsätzlich auf die Fähigkeiten des Menschen. Jener versucht die gewünschten Entitäten bzw. Strukturen durch die Erstellung von Regeln zu erkennen. Der Knowledge Enginneer ist auch mit den Eigenschaften des zugrundeliegenden Systems vertraut. Zudem muss er die Fähigkeit besitzen, vielversprechende Regeln zu erkennen und diese, wenn nötig, zu verallgemeinern bzw. zu spezialisieren. In der Regel ist der Knowledge Engineer mit den Eigenheiten der Domäne vertraut. Ist dies nicht der Fall muss ein Experte der jeweiligen Domäne

konsultiert werden. Eine Alternative wäre, dass sich der Knowledge Engineer selbst in die Domäne einarbeitet. Der Aufbau einer Regelbasis beim Knowledge Engineering Ansatz sieht folgendermaßen aus: Ausgangspunkt ist ein Corpus mit domänenspezifischen Dokumenten. Der Corpus muss dabei über eine adäquate Größe verfügen. Adäquat bedeutet in diesem Kontext, dass die manuelle Analyse des Textes in einem angemessenen Zeitraum durchgeführt werden kann. In der Initialphase wird der gesamte Corpus durchgelesen und die häufigsten bzw. vielversprechensten Regeln in eine Regelbasis aufgenommen. Abhängig von den Ergebnissen der Regeln, werden diese schrittweise verfeinert. Dieser Prozess wird solange durchgeführt, bis die Extraktionsregeln ein angemessenes Ergebnis erzielen. Alleine das Geschick und die Fähigkeiten des Knowledge Engineers entscheiden über die Qualität der Regeln.

Unter einer Regel versteht man generell eine Sequenz von Wörtern. Eine einfache Regel zur Detektion von Bandmitgliedern könnte zum Beispiel so aussehen:

drummer <Vorname> <Nachname>

Eine Regel besteht grundsätzlich aus einem variablen und einem konstanten Teil. Der variable Teil, in diesem Fall das Konstrukt <Vorname> <Nachname>, steht als Platzhalter für die jeweilige Entität. Je nachdem welche interne Struktur die Information aufweist, sind diese Platzhalter unterschiedlich angeordnet. Der konstante Teil übernimmt die Aufgabe auf die gewünschten Entitäten hinzuweisen. Dieser ist in der Regel auf die Domäne zugeschnitten. Je allgemeiner diese konstanten Token gehalten werden, desto größer ist die Wahrscheinlichkeit, eine nicht relevante Information zu erhalten. Befindet sich nicht nur vor, sondern auch nach der Entität ein konstanter Teil, kann dieser auch die Aufgabe eines Delimiters erfüllen. Je mehr konstante Token sich in einer Regel befinden, desto präziser werden diese. Allgemein herrscht eine Diskrepanz zwischen Precision und Recall [28]. Der Knowledge Engineer muss sich entscheiden, ob seine Regeln eher wenige präzise Informationen oder viele eher unsichere Konstrukte zusammentragen möchten. Die Schwierigkeit besteht darin diese beiden Werte gegeneinander abzuwiegen. Eine einfache Alternative wäre die Regel auf eine hohe Präzision auszulegen und für jede Variation eine neue (präzise) Regel zu erstellen. Allerdings würde die Regelbasis mit dieser Strategie sehr schnell explodieren, was sich nun wieder negativ auf die Performanz des Systems auswirkt. Dies zeigt, dass der Prozess der Regelerstellung mit sehr vielen Schwierigkeiten verbunden ist. Eine fundamentale Designentscheidung bei diesem Prozess ist die oben angesprochene Diskrepanz zwischen Precision und Recall. In der Regel konzentriert man sich aber auf einen Wert und versucht den anderen sukzessive aufzubauen. Man hat die Wahl zwischen den zwei folgenden Ansätzen [6]:

- Der molekulare Ansatz zielt von Anfang an auf eine hohe Precision ab und versucht den Recall Wert im Laufe des Prozesses zu steigern

- Der atomare Ansatz hingegen startet mit einem hohen Recall Wert und inkrementiert nach und nach den Precision Wert.

Selbstverständlich hängt es von den Zielen des Systems ab, welche Variante man schließlich wählt. Generell pendeln sich die Werte nach der Initialphase dieses Verfahren eigentlich sehr schnell ein und beide Werte erzielen ähnliche Ergebnisse.

3.3.2.2 Automatic Training Approach

Der Automatic Training Approach ist eine grundsätzlich andere Herangehensweise an diese Problemstellung. Durch eine maschinelle Verarbeitung eines zur Verfügung gestellten Trainingscorpus werden Regeln bzw. Strukturen der gewünschten Informationen erlernt. Somit ist es nicht notwendig den Corpus von einem Experten analysieren zu lassen. Diese Aufgabe übernimmt der Automatic Training Ansatz. Der Mensch muss dabei die relevanten Dokumente zusammentragen um einen Trainingscorpus für die jeweilige Implementierung zu erstellen. Zusätzlich muss dieser Trainingscorpus von Menschenhand vorverarbeitet bzw. annotiert werden. Im Vorgang der Annotation werden die relevante Informationen hervorgehoben. Jene signalisiert dem Programm, welche Stellen im Dokument interessant sind und welche vernachlässigt werden können. Allerdings ist die Annotierung eines Trainingscorpus ebenfalls ein sehr langwieriger und zeitintensiver Prozess. Teilweise stehen Trainingsdaten für gewisse Domänen frei zur Verfügung und können für diesen Zweck herangezogen werden. Ist dies nicht der Fall, so muss der Trainingscorpus selbst erstellt werden. Das ist zwar in den meisten Fällen mit einer Menge Arbeit verbunden, aber mit den vorhandenen Daten können verschiedene Lernstrategien getestet werden. Einige dieser Strategien verfolgen einen Supervised Learning Ansatz [40]. Man kann diese Lernalgorithmen in diesem Kontext in zwei Klassen aufteilen:

- Ansätze, welche das Verhalten eines Knowledge Engineers mimen und aus dem vorhandenen Wissen **Regeln** erzeugen.

- Ansätze, welche anhand der analysierten Dokumente ein **statistisches Modell** generieren, welches die gewünschten Muster im Text erkennt

Die erste Kategorie deckt sich komplett mit dem Knowledge Engineering Ansatz, nur mit dem Unterschied, dass die Regeln automatisiert generiert wurden.

Die statistische Erkennung von Entitäten basiert in den meisten Fällen auf einem Hidden Markov Model [49, 32]. Allerdings werden für statistische Ansätze sehr viele Trai-

ningsdaten benötigt um ein angemessenes Ergebnis zu erzielen. In [6] wird von einem Maximum von 1,2 Millionen Wörtern an Trainingsdaten gesprochen.

3.3.2.3 Gegenüberstellung

In diesem Abschnitt werden die beiden Paradigmen zur Extraktion gegenübergestellt und die jeweiligen Vorteile bzw. Nachteile diskutiert. Auf den ersten Blick mögen die statistischen Ansätze klar im Vorteil liegen. Sie sind generell universell einsetzbar. Wie spezifisch sie auf eine Domäne zugeschnitten sind, kann durch die Trainingsdaten reguliert werden. Die Effektivität des Systems wächst mit der Quantität der zur Verfügung stehenden Daten. Zudem wird kein Knowledge Engineer benötigt, welcher hauptverantwortlich für die Qualität der Informationen ist. Allerdings darf man dabei nicht übersehen, dass die Ergebnisse der bisherigen Evaluierungen [6] für den Knowledge Engineering Approach sprechen. Natürlich hängt dies auch von der Qualität und der Quantität der jeweiligen Trainingsdaten ab. Welcher Ansatz nun endgültig verwendet wird, ist von den jeweiligen Anforderungen an das System abhängig.

Obwohl der Knowledge Engineering Approach die besseren Ergebnisse erzielt, kommen die Automatic Training Approach Ansätze immer näher an diese heran. Allerdings ist der Prozess des Knowledge Engineering sehr langwierig. Es müssen domänenspezifische Elemente, wie zum Beispiel Wortlisten erstellt werden. Aber vor allem die Regelerstellung und die damit verbundene sich wiederholende Adaptierung der einzelnen Regeln verlangt sehr viel Zeit. Die Flexibilität des Automatic Training Approach ist der größte Vorteil im Vergleich zur Knowledge Engineering Strategie. Wenn einmal die Daten analysiert worden sind, können aus den Trainingsdaten entweder Regeln generiert oder ein statistischer Tagger erzeugt werden. Vor allem die statistische Erkennung von Namen lässt sich damit sehr einfach realisieren, denn die Annotierung der Entitäten benötigt kaum spezielles Domänenwissen. Allerdings kann der Fall auftreten, dass die Menge der vorhandenen Trainingsdaten begrenzt ist. Zudem benötigen einige Klassen von Entitäten eine genaue Spezifikation. Diese Problematik tritt zum Beispiel bei Solokünstlern auf. Bands und Solokünstler können generell als "Artist" klassifiziert werden. Solokünstler haben aber meistens die Struktur eines Personennamens. Werden nun einige Solokünstler als Person annotiert und andere wiederum als Artist, so führt dies zu einer Inkonsistenz. Das wiederum wirkt sich negativ auf Ergebnisse des Automatic Training Approach aus. Die konsistente Haltung von Trainingsdaten ist aber auch ein Vorgang der sehr viel Zeit benötigt.

Generell spielt die Spezifikation bei statistischen Ansätzen eine größere Rolle als bei regelbasierten. Je nachdem welche Anforderungen geändert werden, kann dies negative Auswirkungen für den einen Ansatz haben, den anderen aber nur marginal beeinträchti-

gen. Sobald ein statistisches Modell für eine Menge von Entitätsklassen trainiert wurde, kann man nicht so einfach eine neue Klasse hinzufügen. Um diese Klasse in das Modell miteinzubeziehen, muss das Modell für die gänderten Anforderungen erneut trainiert werden.

Die Annotierungen im Corpus müssen erweitert werden und die Analyse der Daten muss nochmals durchgeführt werden. Der regelbasierte Ansatz ist hier toleranter. Besteht nun die Aufgabe weitere Entitäten zu extrahieren, fügt man diese einfach in die Regelbasis ein. Natürlich müssen diese auch auf die jeweiligen Eigenheiten angepasst werden. Ändert sich nun aber die Struktur der Dokumente sind die statistischen Strategien klar im Vorteil. Das erlernte Wissen kann sehr leicht an die neuen Bedingungen angepasst werden. Bei einem Knowledge Engineering Approach muss die Regelbasis im schlimmsten Fall verworfen werden und erneut für die geänderten Anforderungen erstellt werden.

Es ist also nicht einfach einen klaren Favoriten festzulegen. Um die Wahl zwischen den beiden Ansätzen zu erleichtern, stellen Appelt und Israel [6] diesen Leitfaden zur Verfügung:

Knoledge Engineering	**Automatic Training**
spezifische Ressourcen verfügbar	keine Ressourcen verfügbar
Knowledge Engineer verfügbar	Knowledge Engineer nicht vorhanden
wenig Trainingsdaten vorhanden	ausreichende Daten vorhanden
Spezifikation ändert sich	keine Änderungen in der Spezifikation

Tabelle 3.2: Leitfaden nach Appelt und Israel

Welcher Ansatz schließlich gewählt wird, ist von Situation zu Situation verschiedenen. Es hängt von den zu verfügenden stehenden Dokumenten, dem Wissen um die Domäne und nicht zuletzt von persönlichen Präferenzen ab. Zudem sei noch angemerkt, dass obwohl diese beiden Paradigmen grundverschieden sind, die Möglichkeit besteht sie im System miteinander zu kombinieren.

3.4 Wissensbasis

Bei einer Wissensbasis handelt es sich in der Regel um die Repräsentation von Wissen in einer strukturieren Form. Wissen kann dabei in unterschiedlicher Weise repräsentiert werden. Wie die einzelnen Informationen aufbereitet werden hängt sehr stark von der Aufgabe ab, ob man Wissen nur repräsentativ darstellen oder die unterliegende Struktur für weitere Verarbeitungsschritte nutzen möchte.

Dafür wird zunächst eine gewisse Grundstruktur benötigt um Wissen zu repräsentieren. Ein Ausgangspunkt für eine adäquate Form der Darstellung von zusammenhängenden Informationen bietet das Entity Relationship Modell [12, 39, 55]. Jene wurden eingeführt um die theoretischen Konzepte [17, 18] von Datenbanken zu modellieren. Grundsätzlich bestehen diese aus zwei Komponenten: Entitäten und deren zugehörigen Relationen. Dieses Konzept dient als Basis für die Wissensrepräsentation.

Die Struktur wird in diesem Bereich als eine Art Wissensgraph bzw. semantisches Netz angesehen. Dieses Netz verfügt über dieselbe Struktur wie ein Entity Relationship Modell. Jedoch wird die Struktur des Modells ausgenutzt um eventuell weiteres Wissen abzuleiten. Natürlich müssen dabei auch Relationen im Text erkannt und extrahiert werden, um ein solches semantisches Netz zu generieren. Generell existieren zwei Konzepte bei semantischen Netzen [43, 50]: Zum Einen die Generalisierung, zum Anderen die Aggregation. Die Generalisierung stellt eine hierarchische Anordnung von verschiedenen Entitäten dar. Das Konzept der Generalisierung repräsentiert grundsätzlich eine einfache **isA** Beziehung. Mit dieser Modellierung lassen sich vorerst einfache Aussagen bilden (zum Beispiel "Aerosmith is a Band").

Die Aggregation versieht die jeweiligen Entitäten mit Attributen. Jede Entität kann durch verschiedene Attribute beschrieben werden. Ab und zu überlappen sich diese Attribute, aber generell verfügt jede Entität eine spezielle Menge an Attributen, welche sie von den anderen Entitäten unterscheidet. Die Entität "Band" könnte folgendermaßen modelliert werden:

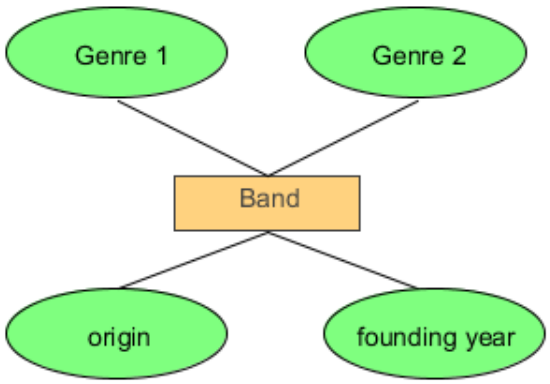

Abbildung 3.1: Beispiel für Aggregation im Entity Relationship Modell

Eine Band besteht aus einer endlichen Anzahl an Mitgliedern. Zudem verfügt sie über verschieden Medien (Singles, Alben, usw.) und eventuell über ein Label, welches die Medien wiederum vertreibt. Natürlich kann man dieser Entität noch mehrere Attribute

zuordnen. Dies hängt ganz davon ab wie detailliert die Informationen dargestellt werden müssen.

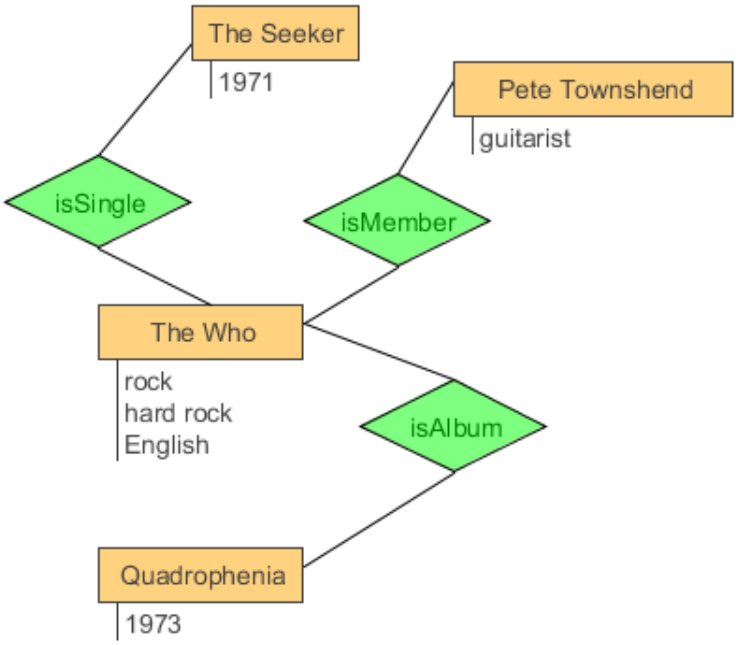

Abbildung 3.2: Einfaches semantische Netz

Je nachdem wie die Struktur aufgebaut ist, könnten eventuell auch einfache Fragestellungen beantwortet werden. Eine Frage könnte durch eine Traversierung der unterliegenden Struktur automatisch verarbeitet werden. Eine einfache Frage wäre zum Beispiel:

Who is the guitarist of Aerosmith?

Besteht ein Pfad zwischen den Entitäten Band und Bandmember, sowie eine Relation vom Typ isGuitarist zwischen den beiden Entitäten, so kann die Frage definitiv beantwortet werden [50]. Wie komplex die Daten nun tatsächlich angeordnet werden müssen, hängt aber grundsätzlich von den Anforderungen an die Wissensbasis ab.

3.5 Evaluierung

Ein wesentlicher Punkt ist die Evaluierung des entwickelten Systems. In der Regel unterteilt sich der Prozess der Evaluierung in zwei Aufgaben: Die Erstellung eines Testcorpus bzw. einer Ground Truth und die Messung der Ergebnisse anhand spezieller Metriken. Bei der Evaluierung der einzelnen Systeme haben vor allem die zahlreichen Message Unterstanding Conferences (MUC) [1, 2, 3, 4, 42] große Pionierarbeit geleistet. Da sich die Konferenzen vor allem mit der Evaluierung beschäftigten, wurden hier die Standards im Bereich des IE festgelegt. Die Maßzahlen für die Evaluierung von IE Systemen wurden in der zweiten MUC Konferenz [52] definiert.

Vor allem ermöglichen diese Maßzahlen es verschiedene Systeme oder nur einzelne Strategien miteinander zu vergleichen. Zudem sind sie ein guter Indikator für die Verbesserung der Technologie im Laufe der Zeit [19]. Die Maßzahlen wurden aus dem verwandten Bereich des Information Retrievals für diesen Kontext adaptiert. Die zwei Hauptmaße sind Precison und Recall. Diese setzen sich folgendermaßen zusammen:

$$P = \frac{|extracted \cap relevant|}{|extracted|} \tag{3.1}$$

$$R = \frac{|extracted \cap relevant|}{|relevant|} \tag{3.2}$$

Dabei umfasst *extracted* die Menge aller vom System extrahierten Informationen aus dem gegebenen Corpus. Die Menge *relevant* hingegen, steht für die Gesamtheit aller relevanten Informationen im vorliegenden Corpus. Die Formel für Precision und Recall werden in der Literatur [34, 6, 28] teilweise etwas unterschiedlich definiert, generell laufen sie aber auf dasselbe Ziel hinaus: Die Precision ist ein Maß für die Korrektheit der extrahierten Informationen. Der Recall Wert hingegen misst wie viele korrekte Informationen gefunden wurden. Die beiden Maßzahlen bewegen sich in einem Wertebereich zwischen 0 und 1. Da zwischen Precision und Recall immer eine gewisse Diskrepanz herscht [28], ist es schwieriger für beide Werte angemessene Ergebnisse zu erzielen. Grundsätzlich werden beide Werte benötigt, um ein System zu evaluieren. Eine Evaluierung anhand nur einer Maßzahl ist nicht immer aussagekräftig. Precision und Recall können auch miteinander zu einer neuen Maßzahl kombiniert werden. Die wohl am bekannteste Form davon ist der F-Measure Wert:

$$F = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R} \tag{3.3}$$

Mit der Variable β kann der Einfluss von Precision und Recall reguliert werden. Der Wertebereich liegt ebenfalls zwischen 0 und 1, wobei eine 1 zu einer Gleichgewichtung von Precision und Recall führt. Appelt und Israel schlagen in [6] einen Wert von 0.6 vor.

Um die Maßzahlen berechnen zu können wird die Ground Truth herangezogen. Die Struktur dieser Ground Truth hängt natürlich von der Spezifikation ab.

Die Berechnung von Precision und Recall ist nur ein wesentlicher Punkt der Evaluierung. Vor allem der Aufwand sowohl einen Trainingscorpus als auch einen Testcorpus zu erstellen, zwingt die Entwickler zu Kompromissen, wenn kein bereits aufbereiteter Corpus zur Verfügung steht, denn die Erstellung eines Test- bzw. Trainingscorpus ist mit einer Menge Arbeit verbunden. Zuerst müssen spezifische Texte von einer verlässlichen Quelle bezogen werden. Je nachdem um welche Quelle es sich dabei handelt, muss man diese eventuell vorverarbeiten. Zusätzlich muss im Testcorpus auch noch das relevante Wissen markiert werden, um eine aussagekräftige Evaluierung zu gewährleisten. Die markierten Wörter werden dabei mit dem extrahierten Wissen abgeglichen und die oben erwähnten Maßzahlen dafür berechnet.

Eine Alternative wäre die Evaluierung anhand eines einzigen annotierten Sets mittels Cross Validation. Allderdings besteht dabei immer noch die zeitintensive Notwendigkeit einer händischen Annotierung.

Eine Evaluierung kann also ein sehr langwieriger Vorgang sein, aber er ist notwendig um die Stärken bzw. Schwächen eines Systems aufzuzeigen.

Kapitel 4

Beschreibung eines IE Systems am Beispiel GATE

GATE ist die Kurzform für General Architecture for Text Engineering. Der Entwickler kann auf ein breites Spektrum an Komponenten zurückgreifen und diese für sein System adaptieren. GATE wurde 1995 an der Universität von Sheffield [24], konzipiert. Die aktuelle Version kann unter http://gate.ac.uk/ kostenlos bezogen werden. Die Architektur basiert vor allem auf den Erkenntnissen, welche aus der Entwicklung des TIPSTER [25] Systems gewonnen wurden.

4.1 Architektur

In diesem Kapitel werden die verschiedenen Anforderungen an eine allgemeine IE Architektur vorgestellt. Grundsätzlich besteht die Architektur grob aus zwei Teilen. Einer zentralen Verwaltungsstelle der Dokumente, dem Corpus und einer Pipeline, welche die einzelnen Dokumente verarbeitet. Diese setzt sich aus verschiedenen Verarbeitungseinheiten zusammen.

4.2 Verwaltung der Daten

Grundsätzlich besteht jedes IE System aus zwei fundamentalen Komponenten: Einer Verwaltungs- und einer Verarbeitungsstelle. Der Corpus umfasst die Gesamtheit aller Dokumente im IE System. Die einzelnen Verarbeitungsstationen (Tokenizer, POS Tagger. ...) erweitern die Dokumente um Metainformationen. Man bezeichnet diese Form der Anreicherung als Annotierung. Diese Annotierungen kennzeichnen später jene

Wörter bzw. Wortfolgen welche vom System extrahiert werden sollen. Die Dokumente kann man beim Informationsextraktion-Prozess von verschiedenen Quellen beziehen. Grundsätzlich kann dafür jedes textuelle Medium herangezogen werden. In [46] werden die verschiedenen Bezugsquellen noch etwas detaillierter besprochen. Egal ob die Daten vom World Wide Web oder aus einem Dokumentarchiv stammen, lassen sich die einzelnen Quellen anhand ihrer Struktur kategorisieren. Man unterteilt diese in drei unterschiedliche Klassen [28, 46]:

4.2.1 Unstrukturierter Text

Laut der gängigen Definitionen von Information Extraktion [28] werden die extrahierten Informationen von unstrukturiertem Text bezogen. Es handelt sich also dabei um die Rohform von IE. Dabei werden die Texte mittels einem Tokenizer und einem POS Tagger vorverarbeitet. Diese Schritte sind notwendig, um die einzelnen Fakten zu extrahieren. Die Natur von unstrukturierten Texten macht es trotz dieser Vorverarbeitungsschritte nicht leicht die gewünschten Informationen zu beziehen, denn es muss immer ein gewisses Indiz für Existenz einer Entität im Text vorhanden sein: Ein Schlagwort (zum Beispiel: band, released, usw), welches vermuten lässt, dass nun ein spezifische Information (also der Name einer Band) folgt. Die zweite Problematik liegt darin die gewünschte Information (den Namen einer Band) vollständig zu erkennen bzw. zu annotieren. Je nach Domäne kann eine Information einen unterschiedlichen Komplexitätsgrad aufweisen bzw. aus meherern Wörtern bestehen. Zwar kann die Erfolgsrate bei frei strukturiertem Text nicht mit der eines Menschen verglichen werden, liefert aber durchaus brauchbare Ergebnisse, wenn das IE System auf die gewünschten Informationen zugeschnitten wird.

4.2.2 Semistrukturierter Text

Ein semistrukturierter Text beinhaltet sowohl freie als auch strukturierte Textelemente. In den meisten Fällen, lassen sich die gängigen NLP Techniken nicht auf diese Art von Texten anwenden, da diese nur für freie (unstrukturierte) Texte entwickelt wurden [28]. Dies gilt jedoch nur eingeschränkt für HTML oder XML Dateien, wo freier Text in der Regel von den einzelnen Tags gekapselt wird. Zusätzlich weisen HTML Seiten gewisse Strukturen wie Listen und Tabellen auf, welche eine einfache Extraktion ermöglichen. Allgemein ist es aber sehr schwierig Dokumente aus dem Web einzuordnen, da diese, sowohl strukturierte als auch unstrukturierte Dokumente umfasst.

4.2.3 Strukturierter Text

Strukturierter Text wird generell in einem strikt definierten Format gespeichert, wie zum Beispiel einer Datenbank. Durch die starre Struktur der Daten können die Informationen ohne größere Probleme extrahiert werden, da allgemein keine komplizierten NLP Techniken notwendig sind. Allerdings muss das Format bekannt sein, in anderen Fällen sind die Informationen nicht verwertbar oder das zugrundeligende Format muss erlernt werden.

4.3 Verarbeitungseinheiten

Obwohl sich die verschiedenen IE Systeme grundsätzlich in ihren Zielen und Aufgaben unterscheiden, muss eine Basispipeline über eine gewisse Grundstruktur verfügen. Jede Komponente dieser Pipeline implementiert eine gewisse NLP Strategie, welche den Text für die Extraktion vorbereitet. Die Basispipeline lässt sich in vier Grundkomponenten unterteilen, welche in Abbildung 4.1 dargestellt werden. [6, 11]

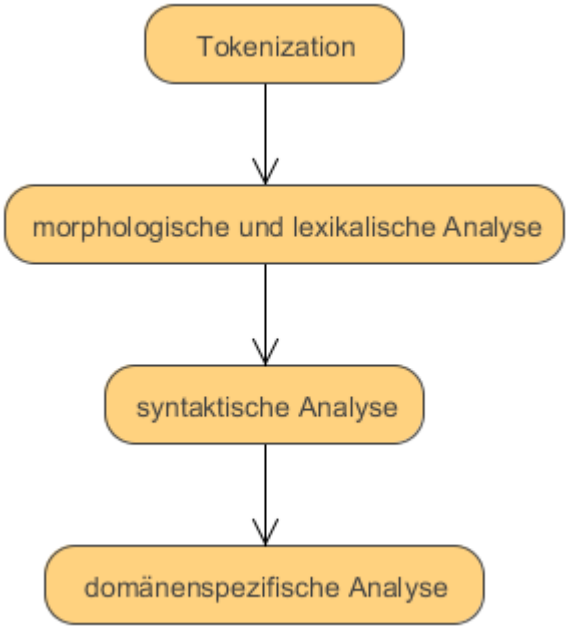

Abbildung 4.1: Basispipeline

Teilweise weichen die Bezeichnungen in der Literatur [34, 46] etwas ab, aber im Grunde genommen besteht jedes System mindestens aus einem Tokenizer, einer Art lexikalischer

und morphologischer Analyse, einer syntaktischen Analyse und einer domänenabhängigen Verarbeitung des jeweiligen Dokuments. Teilweise bestehen diese Einheiten wieder aus verschiedenen Subkomponenten. Die Elemente der vorgestellten Basispipeline werden im nun folgenden Abschnitt detaillierter besprochen.

4.3.1 Tokenization

Der erste Schritt beinhaltet eine Vorverarbeitung des Dokuments. Ein Dokument besteht aus einer endlichen Aneinanderreihung von Zeichen. Unter Tokenization versteht man den Prozess ein Dokument in eine Sequenz von Tokens bzw. Wörtern umzuwandeln. Dieser Prozess generiert eine Abfolge von Wörtern, welche später durch die anderen Komponenten der Pipeline leichter verarbeitet werden können. Ein Dokument kann man also als eine einfache Abfolge von Token betrachten. Der Vorgang der Tokenization läuft anhand eines Beispiels folgendermaßen ab:

Aerosmith is an American rock band, sometimes referred to as "The Bad Boys from Boston".

wird durch Tokenization zu

Aerosmith|is|an|American|rock|band|,|sometimes|referred|to|as| The|Bad|Boys|from|Boston|
.

Dieser einfache Satz wird in diesem Beispiel von allen Leerzeichen befreit. Zusätzlich wird der Punkt am Ende entfernt. Übrig bleiben die einzelnen Wörter, wobei man die Leerzeichen sowie den Punkt ebenfalls als eine spezielle Ausprägung eines Tokens ansehen kann. Grundsätzlich stellt die Unterteilung in Token keine größeren Probleme für europäische Sprachen dar. Man definiert lediglich eine Menge an Trennzeichen, welche eine Trennung in der Sequenz von Charactern symbolisiert. In der Regel handelt es sich dabei um typische Symbole, wie Leerzeichen, Beistriche, Punkten und andere Delimiter. Allerdings existieren solche offensichtliche Trennzeichen nicht in Sprachen wie Japanisch. Solche Sprachen benötigen eine darauf abgestimmte Form der Tokenization.

4.3.2 Lexikalische und morphologische Analyse

In diesem Kapitel werden die einzelnen Subkomponenten dieser Phase behandelt. Diese Komponenten sind ebenfalls von der verwendeten Sprache abhängig. Allgemein besteht

die Aufgabe dieser Einheiten darin, die einzelnen Elemente im Dokument zu klassifizieren. Dabei werden nicht nur einzelne Token in verschiedene Klassen eingeteilt, sondern auch komplexere Strukturen zusammengefasst. Jene Informationen können im späteren Extraktionsprozess wiederum von Nutzen sein.

4.3.2.1 Lexikalische Wörterbuchsuche

Eine Form dieser Analyse ist die lexikalische Wörterbuchsuche. Dabei werden die einzelnen Token mit den Einträgen eines vordefinierten Wörterbuchs abgeglichen, um diese einer spezifischen Annotierungsklasse zuzuweisen. Generell gibt es bei der lexikalischen Wörterbuchsuche zwei fundamentale Fragen:

1. Welche Elemente muss das Wörterbuch beinhalten?

2. Wie groß muss das Wörterbuch sein?

Vor allem bei unstrukturiertem Text ist es naheliegend ein breites Spektrum abzudecken. Somit ist es auch möglich, seltenere Wörter zu klassifizieren. Doch ein umfangreiches Lexikon muss nicht zwingend bessere Ergebnisse liefern. Zum einem leidet die Performanz des Systems an einem sehr großen Lexikon, zum anderen bringen umfangreichere Wörterbücher neue Probleme mit sich: Bei einem umfangreichen Wörterbuch ist die Gefahr von Ambiguitäten viel größer als bei spezifischen Wortlisten. Ein Wort mit mehreren Bedeutungen bedarf wieder einer Erweiterung, um dieses auch richtig zu klassifizieren. Allgemein empfiehlt es sich das Lexikon auf die jeweilige Domäne zuzuschneiden: Einerseits hat das Wörterbuch eine moderate Größe, andererseits werden nur domänenspezifische Begriffe klassifiziert, was wiederum die Problematik von Ambiguitäten größtenteils beseitigt.

4.3.2.2 POS-Tagging

Einige der zahlreichen Verarbeitungsschritte in der Pipeline markieren die jeweiligen Tokensequenzen mit speziellen Metainformationen. Diees Metainformationen können dabei helfen den Extraktionsprozess zu erleichtern. Der Prozess des Part of Speech Tagging versieht die einzelnen Token mit grammatischen Informationen. Diese Strategie klassifiziert die einzelnen Token bzw. Wörter in Wortklassen einer natürlichen Sprache, also Nomen, Verben, usw. Diese Wortarten bzw. -klassen werden generell in weitere Unterklassen unterteilt. So kann man ein Pronomen in Personalpronomen, Possessivpronomen, etc. einteilen. Wie detailliert die Tokens klassifiziert werden, hängt von

der Implementierung des Part of Speech Taggers, aber auch von der Zielsprache, ab. Eine solche Unterteilung für die englische Sprache ist in Tabelle 4.1 dargestellt. [1]:

POS Tags	Beispiele
Noun	Aerosmith
Verb	left, join
Participle	has joined, was joined
Interjection	Hi, Bye
Pronoun	I, He, She
Preposition	of, to, on
Adverb	quickly, again, often,
Conjunction	and, but, or

Tabelle 4.1: Englische POS Tags

Ein Part of Speech Tagger kann grundsätzlich auf zwei Arten realisiert werden. Die Kategorisierung kann entweder durch einen regelbasierten [10, 36] oder durch einen statistischen Ansatz [9] erzielt werden. Bei Letzterem wird die Klassifizierung durch einen annotierten Corpus erlernt. Allerdings muss dieser Trainingscorpus von einem Menschen vorklassifiziert werden, um daraus ein statistisches Modell zu erstellen. Generell werden sowohl bei einem regelbasierten als auch bei einem statistischen Ansatz zu 95 Prozent korrekt klassifiziert [6]. Jedoch darf man nicht außer Acht lassen, dass diese Ansätze mit allgemeinen Texten trainiert wurden. Bei Dokumenten mit domänenspezifischen Inhalten kann die Korrektheit des Taggers darunter leiden, da eventuell sehr spezielle domänenspezifische Begriffe falsch klassifiziert werden.

4.3.2.3 Tagging von Strukturen

Im Gegensatz zu den bereits vorgestellten Methoden der lexikalischen Analyse ist die Erkennung und Klassifizierung von Strukturen um einiges diffiziler. Verfahren wie Part of Speech Tagging konzentrieren sich auf die Klassifizierung eines Tokens, Strukturen bestehen aber in der Regel aus einer Aneinanderreihung von Tokens. Diese muss nicht zwingend Listen oder ähnliche Aufzählungen beinhalten. In diesem Kontext bedeuten Strukturen eher Namen oder Datumsangaben. Grundsätzlich verfügt ein Name oder ein Datum im Text ebenfalls über eine gewisse Substruktur[6]. Man nehme zum Beispiel das Format eines Datums:

02. 02. 1985

[1] http://www.usingenglish.com/glossary/parts-of-speech.html

Dieses einfache Format besteht generell aus Zahlen und Punkten. Verschiedene Strukturen können aber auch dieselbe Information beinhalten. So kann ein Datum auch folgendermaßen dargestellt werden.

2 Februar '85

Gerade diese Strukturen sind für den Anwender des Systems interessant, denn man will nicht nur einzelne Schlagwörter extrahieren, sondern Namen, Daten oder Beziehungen präsentiert bekommen. Allerdings bereiten diese Strukturen, vor allem Namen, einem IE System die größten Schwierigkeiten. Allgemein gesehen ist der Begriff oder die Struktur "Name" ein sehr weitläufiger Begriff. Die Klasse "Namen" kann wiederum in sehr viele Unterkategorien aufgespalten werden: Personenname, Produktname, Firmenname, Name eines Ortes, usw.

Es ist gut möglich, dass ein Name in zwei Kategorien fällt. Ein Beispiel aus der Musikdomäne dafür wäre die Unterscheidung zwischen dem Namen eines Solokünstlers und dem Namen einer Band. Von der Struktur her ist der Solokünstler eine Person, er könnte aber durchaus auch als Bandname kategorisiert werden. Ein weiteres Problem das dabei nicht vergessen werden darf ist, dass Namen keiner Konvention unterliegen. Sie können in jeder beliebigen Form vorkommen und genau dies macht es so schwierig sie zu erkennen.

Um diese Strukturen zu erfassen stehen zwei verschiedene Techniken zur Verfügung. Zum einem werden diese Strukturen durch selbstentwickelte Regeln, zum anderen durch Hidden Markov Models [49, 32] extrahiert. Man kann also wieder wie beim Part of Speech Tagger einen regelbasierten oder einen statistischen Ansatz wählen.

Dabei sind die Regeln das Resultat einer von Menschen durchgeführten Recherche. Es werden lediglich Dokumente mit der gewünschten Namenklasse durchgelesen und die häufigsten Muster in Regeln umgewandelt. Eine stark vereinfachte Regel für die Erkennung einer Person könnte folgendermaßen aussehen:

Mr. <Vorname><Nachname>

Hidden Markov Modelle hingegen verwenden dafür einen statistischen Ansatz. Diese Modelle werden im Kapitel 3.3.2.2 genauer besprochen.

Welche Strategie der Entwickler schließlich wählt, hängt ganz von den Anforderungen an das System ab. Allerdings wird in [6] behauptet, dass bisher die regelbasierten Konzepte immer ein wenig besser abgeschnitten haben, als selbstlernende Systeme.

4.3.3 Syntaktische Analyse

Eine Syntaxanalyse des jeweiligen Dokuments kann die nachfolgende domänenspezifische Analyse vereinfachen. Die syntaktische Verarbeitung von Textpassagen kann auf die Form der zu extrahierenden Daten hinweisen: "After all, the arguments to be extracted often correspond to noun phrases in the text, and the relationships to be extracted often correspond to grammatical functional relations." [34]

Jedoch ist diese detaillierte Vorverarbeitung ein sehr aufwendiger Prozess. Laut Appelt und Israel [6] ist die NLP Verarbeitung im Kontext von Informations Extraktion immer mit einem Kompromiss verbunden. Man muss sich stets bewusst sein, dass in der Regel große Mengen an Dokumenten verarbeitet werden. Diese Aufgabe muss natürlich auch in einer annehmbaren Zeit bewältigt werden. Einzelne Systeme verzichten daher gänzlich auf den Prozess der syntaktischen Analyse. Jedoch gibt es auch Vertreter des anderen Extrems, welche eine vollständige Analyse durchführen [5]. Die meisten Systeme gehen einen Kompromiss ein und verarbeiten nur vereinzelte Segmente des Dokuments. Diese konzentrieren sich vor allem auf Gruppierungen von Hauptwörtern. Hauptwörter verweisen in der Regel auf potentiell interessante Fakten. Im Schritt der domänenspezifischen Analyse können die einzelnen Segmente darauf überprüft werden, ob sie relevante Wörter bzw. Phrasen enthalten. Generell beschränkt man sich dabei auf folgende Ausprägungen in den Dokumenten [6, 34]:

- Sequenzen von Hauptwörtern (um Namen zu annotieren)
- Verben
- Partikel (Präpositionen, Konjunktionen,...)

Jene Ausprägungen ermöglichen es nun Fakten bzw. auch Relationen aus einfachen Strukturen zu extrahieren. Dies könnte folgendermaßen aussehen:

In 1999, <Hauptwort (-wörter)> left <Hauptwort (-wörter)>.

Aus diesem einfachen Konstrukt, könnte man nun eine Relation zwischen <Hauptwort (-wörter)> und <Hauptwort (-wörter)> herstellen. Diese beiden Hauptwörter kann man durch das Verb *left* verbinden. Befindet sich eine Abfolge wie <Hauptwort (-wörter)><Verb><Hauptwort (-wörter)> im Text, so kann man eine einfache Relation herstellen. Um Laufzeit einzusparen, beschränkt man sich in der Regel auf einzelne Sätze bzw. Absätze, welche auch über mindestens drei der angesprochenen Asuprägungen verfügen.

4.3.4 Domänenspezifische Analyse

In den vorangegangen Verarbeitungsschritten wurde lediglich eine allgemeine Klassifizierung von Tokens vorgenommen. Einzig beim Annotieren von Strukturen richtet man sich auf die unterliegende Domäne aus. Das Dokument beinhaltet also eine gewisse Menge an interessanten Namen bzw. Strukturen. Allerdings verbergen sich Informationen auch of in nicht so offensichtlichen Konstrukten wie Sequenzen von Eigennamen: Ein Mitglied einer Band könnte auch immer mit einem Personalpronomen angesprochen werden. Bands mit sehr langen Namen werden in zahlreichen Musikplattformen mit Kürzel angesprochen. So wird die Band "...And You Will Know Us by the Trail of Dead" oft einfach nur als "Trail of Dead" bezeichnet. Andererseits existieren Gruppen wie die "Dirty Pretty Things" und die "Pretty Things". Für einen Menschen mit dem entsprechenden Hintergrundwissen ist klar ersichtlich, dass es sich bei diesen unterschiedlichen Strukturen um dieselbe Information bzw. bei den ähnlichen Strukuren nicht um die selbe Band handelt. Eine Maschine hingegen sieht nur eine Struktur, welche sich in der Anzahl der Tokens unterscheidet. Bei der domänenspzeifischen Analyse wird versucht unterschiedliche Strukturen welche dieselben bzw. zusammenhängende Informationen beinhalten miteinander zu verbinden.

4.3.4.1 Koreferenzen

Die Koreferenzenanalyse beschreibt die Problematik von scheinbar unterschiedlichen Entitäten, welche aber dieselbe Bedeutung haben. Ein gutes Beispiel dafür sind Personennamen oder Abkürzungen. Diese Problematik kann folgendermaßen verdeutlicht werden: Ein Biographie über die australische Rockgruppe AC/DC könnte verschiedene Variationen ihres verstorbenen Lead Sängers beinhalten: Bon Scott, Ronald Belford "Bon" Scott oder einfach nur Scott. Ein anderes Problem tritt bei Abkürzungen auf. Die amerikanische Band Red Hot Chilli Peppers werden auch desöfteren mit dem Akronym RHCP abgekürzt. Eine einfache Möglichkeit Abkürzungen einer konkreten Entitäten zu zuordnen, wird von Appelt [6] beschrieben: Aus Entitäten mit großgeschriebenen Tokensequenzen werden die jeweiligen Akronyme gebildet. Im Anschluss wird das Dokument auf solche Konstrukte durchsucht.

Noch problematischer wird es, wenn eine Entität mehrere Bedeutungen hat, wie im Falle der amerikanischen Band R.E.M. Zum einem kann ein und dieselbe Entität variieren, zum anderen hat diese in einem anderen Kontext eine ganz andere Bedeutung.

R.E.M = Rapid Eye Movement

Ein weiteres Problem tritt auf, wenn ein Personalpronomen stellvertretend für den Namen verwendet wird. Dieses Problem wird als anaphorische Referenz oder kurz Anapher bezeichnet. Ein Beispiel für eine Anapher wäre:

Ronald Belford Scott was born on 9 July 1946. He formed his first band ...

Die Schwierigkeit besteht darin, zu erkennen, dass sowohl die Hauptwörter als auch das Personalpronomen auf dieselbe Person verweisen. Das oben präsentierte Beispiel gehört zu den einfacheren Problemen. Tritt ein Personalpronomen auf, so verwendet man einfach den Namen, welcher zuletzt aufgetreten ist. Bei komplexeren anaphorischen Referenzen ist dies allerdings nicht mehr so einfach. Appelt und Israel skizzieren in [6] einen Lösungsansatz für diese Probleme.

4.3.4.2 Verknüpfte Entitäten und deren Zusammenführung

Die bisherigen Verarbeitungsschritte haben lediglich das Dokument bzw. den Text auf den Vorgang der Extraktion vorbereitet. Die Erkennung von Namen und die syntaktische Analyse bildeten bisher die einzige Ausnahme. Hier wurden die jeweiligen Entitäten im vorliegenden Text markiert.

Je nachdem wie komplex die Ausgabe der bezogenen Informationen gestaltet wird, benötigt man weitere Verarbeitungsschritte. Die extrahierten Informationen werden in einem strukturierten Raster, dem Template, verwaltet. Ein komplexes Template verwendet geschachtelte Platzhalter, welche gefüllt werden müssen. Man kann sich das folgendermaßen vorstellen: Die Pipeline findet ein einfaches Ereignis im vorliegenden Text, wie zum Beispiel:

In 1999, <Member1> left <Band1>.

In der Regel lösen Ereignisse wiederum andere Ereignisse aus. So wie in diesem konkreten Fall, das Ereignis <Member1> left <Band1>, weitere Ereignisse auslöst. Zum einem benötigt <Band1> nun einen Ersatz für <Member1>. Somit tritt Folgeereignis Nummer 1 in Kraft:

<Member1> was replaced by <Member2> oder <Band1> hired <Member2>

Zudem wird <Member1> mit großer Wahrscheinlichkeit wieder in einer neuen Band sein Glück versuchen, welches Folgeereignis 2 auslöst:

<Member1> joined <Band2>

Das Initialereignis kann natürlich eine Vielzahl an Folgeereignissen auslösen. So könnte sich <Band1> gerade auf Tour befinden und müsste in Folgen dessen die Tour abbrechen.

Natürlich handelt es sich dabei nur um ein hypothetisches Beispiel, im Normalfall sind die vorhanden Verknüpfungen weit nicht so komplex. Vor allem würde ein System mit einer solchen Fülle an Zusammenhängen ohnehin nicht Zurecht kommen. Da diese Ereignisse miteinander zusammenhängen, müssen sie in irgendeiner Weise miteinander verknüpft werden. Dieser Vorgang wird als "Merging" bezeichnet. Grundsätzlich werden die Merging Regeln handgeschrieben. Jene sind also auf die spezifische Aufgabenstellung bzw. Domäne zugeschnitten.

4.4 Einführung in GATE

[22, 26] GATE verfügt über drei grundlegende Anwendungsmöglichkeiten im Kontext des Language Engineerings. Es kann als Architektur, Framework und Entwicklungsumgebung genutzt werden. Die Architektur regelt die Anordnung der einzelnen Komponenten. Zudem legt sie die Form der Interaktion zwischen den einzelnen Komponenten fest. Basierend auf der Architektur von GATE, wurden zwei konkrete Systeme realisiert. Das VIE (Vanilla Extraction System) [37] und LaSie. Im Kontext des Frameworks wird ein wiederverwendbares Design gewährleistet, das für die verschiedensten LE Anwendungen genutzt werden kann. Zudem stehen allgemeine Komponenten zur Verfügung. Diese können vom jeweiligen Entwickler hinsichtlich der gegebenen Anforderungen erweitert bzw. angepasst werden.

Die Infrastruktur von GATE wurde anhand folgender Anforderungen entwickelt [26, 19].

- Strikte Trennung von NLP Verarbeitungsschritten und den jeweiligen Komponenten, für die Visualierung oder Serialisierung der Daten
- Austausch von Daten zwischen den einzelnen NLP Komponenten
- Integration von Modulen, welche in unterschiedlichen Sprachen implementiert wurden.
- Die unterliegende Architektur wird mit offenen Standards, wie Java und XML realisiert

- Bereitstellung von einem Set an Basiskomponenten. Jene können nach Belieben erweitert bzw. adaptiert werden

- Automatisierung von Prozessen wie Benchmarking und Evaluierung

Im Sinne einer Entwicklungsumgebung unterstützt GATE den Anwender bei der Realisierung bzw. Modifizierung eines Systems. Für diesen Zweck stellt die Umgebung zahlreiche Tools, wie einen Debugger für neue Module, zur Verfügung. GATE unterliegt dabei einem komponentenbasierten Modell [33]. Der große Vorteil dieses Ansatzes ist die Flexibilität. Die einzelnen Module können je nach Zweck an das System ein- bzw. ausgekoppelt werden. Zusätzlich können die einzelnen Module um spezifische Implementierungen erweitert werden. So beinhaltet zum Beispiel das Tokenization Modul verschiedene Ausprägungen für verschiedene Sprachen. Zusätzlich bietet GATE die Möglichkeit die verarbeiteten Daten sehr einfach zu visualisieren. Grundsätzlich besteht das Framework aus einer Bibliothek, die die Kernfeatures bereitstellt, und den verschieden LE Modulen. Die zahlreichen Schnittstellen der Architektur werden vom Framework implementiert. Diese Ausprägungen stellen die Grundfunktionalität einer IE Anwendung zur Verfügung. Darunter fallen: Laden der Module, Visualisierung der Daten, usw. Die wiederverwendbaren Module implementieren in der Regel die grundlegenden Verarbeitungsschritte des NLP Prozesses. Somit stehen zum Beispiel die Komponenten der lexikalischen Analyse bereits zu Verfügung und der Entwickler kann sich auf die anderen Aspekte des Systems konzentrieren. Zudem wird eine Menge Zeit eingespart, welche zum Beispiel für die Realisierung einer einzelnen Komponente aufgewendet werden müsste. Die verarbeitenden Daten müssen nicht zwingend in Form einer GUI repräsentiert werden. Die grafischen Komponenten und die einzelnen LE Module sind strikt voneinander getrennt und können gesondert verwendet werden. Somit können auch puristische Kommandozeilensysteme ohne den Overhead einer GUI implementiert werden. Grundsätzlich operieren sowohl der Dokument- bzw. Annotierungs Manager, als auch die wiederverwendbaren Komponenten unabhängig voneinander.

4.5 Komponentenklassen in GATE

Grundsätzlich unterscheidet GATE zwischen drei Hauptkomponenten: Daten, Algorithmen und Visualisierung. Dies liefert die Grundlage für die folgende Klassifizierung der Komponenten:

- Language Resources (LR): Es handelt es sich dabei um Daten in Form eines Dokuments oder Corpus, aber auch um einer Ontologie

- Processing Resources (PR): beinhaltet die einzelnen Verarbeitungsstationen eines IE Systems, wie zum Beispiel Tokenization, Wörterbücher, POS Tagging, usw.

- Visual Resources (VR) ermöglichen die Annotierung, das Verwalten und Verarbeiten von Dokumenten und die Konfiguration der Komponenten in Form einer grafischen Oberfläche

Diese Ressourcen müssen nicht zwingend auf dem lokalen System vorhanden sein, sie können auch von anderen Systemen bezogen werden. Die API stellt Funktionen bereit, um die verschiedenen Ressourcen mit einzubeziehen. Eine komplette Anwendung besteht aus allen drei Klassen, wobei wie bereits erwähnt, eine GUI für eine Anwendung nicht zwingend notwendig ist. Sie kann aber dabei helfen, die vorliegenden Daten besser zu interpretieren. Vor allem die strikte Trennung zwischen Daten und die darauf angewendeten Algorithmen bringt einen großen Vorteil mit sich. Die Gesamtheit aller Ressourcen wird unter den Begriff CREOLE zusammengefasst. Dies ist die Kurzform für "Collection of Reusable Objects for Language Engineering". Jedes wiederverwendbare Objekt wird durch eine eindeutig zugehörige XML Datei spezifiziert. Diese besteht aus dem Namen der Klasse, dem dazugehörigen Parameter und anderen spezifische Informationen. Jene XML Datei ist für das Einkoppeln der Komponente notwendig. Vereinzelt bestehen auch Abhängigkeiten zwischen den einzelnen Ressourcen. Diese werden ebenfalls in der Beschreibung festgelegt. Der Entwickler legt fest welche Prozessressourcen verwendet werden und in welcher Abfolge diese Komponenten abgearbeitet werden. Zusätzlich kann der Entwickler die zu verarbeitenden Daten festlegen. Die Komponenten und deren Anordnung kann dabei in der grafischen Umgebung von GATE festgelegt werden. Die verarbeiteten Dokumente werden im Anschluss im zugehörigen Document Viewer betrachtet und verfügen abhängig von den verwendeten Prozessressourcen über verschiedene Metainformationen, wie zum Beispiel Part of Speech Tags. Zusätzlich hat der Anwender bzw. Entwickler die Möglichkeit die annotierten Texte in verschiedenen Formaten abzuspeichern.

4.6 Verarbeitung von Daten in GATE

GATE [25, 26] verwaltet die Daten, wie bereits erwähnt, zentral als Language Resources (LR). Das Framework unterstützt dabei eine breite Palette an Formaten, von plain text oder RTF, bis zu MarkUp orientierten Dokumenten (XML, HTML, SGML). Zusätzlich kann das vorliegende Dokument in die gängigsten ISO Standards [54, 57] konvertiert werden. GATE verarbeitet und analysiert die Daten und mappt diese auf ein einheitliches internes Modell.Dieses Modell verwaltet die einzelnen Annotierungen im Dokument. Die zugehörigen Annotierungen werden ebenfalls zentral in einem Modell

in GATE verwaltet. Generell fügt jede Prozessresource Annotierungen in das unterliegende Modell ein. So reichert der POS Tagger das Annotierungmodell mit dem Part of Speech Klassen an. Die Annotierungen können entweder automatisch durch die verschiedenen Verarbeitungseinheiten hinzugefügt werden, oder manuell durch Einwirkung des Anwenders. Die einzelnen Verarbeitungsschritte benötigen je nach Komplexität der Aufgabe eine gewisse Zeit. Um diese Schritte nicht immer und immer wieder ausführen zu müssen, können die Dokumente und das zugehörige Annotierungsmodell abgespeichert werden. GATE bietet dabei drei Formen der persistenten Speicherung an:

- ein internes XML Format
- eine Serialisierung mittels JAVA
- Ablage in einer relationalen Datenbank

Außerdem kann ein GATE Dokument wieder in sein Ausgangsformat zurücktransformiert werden. Der Anwender hat die Möglichkeit die annotierten Daten im Docuemnt Viewer von GATE zu betrachten. Die verschiedenen Annotierungsklassen werden aufgelistet, wobei jede Klasse durch eine unterschiedliche Farbe markiert wird. Bei Auswahl einer Klasse werden die zugehörigen Annotierungen im Text durch diese Farbe markiert. Die Visualisierung der annotierten Passagen wird in Abbildung 4.2 dargestellt.

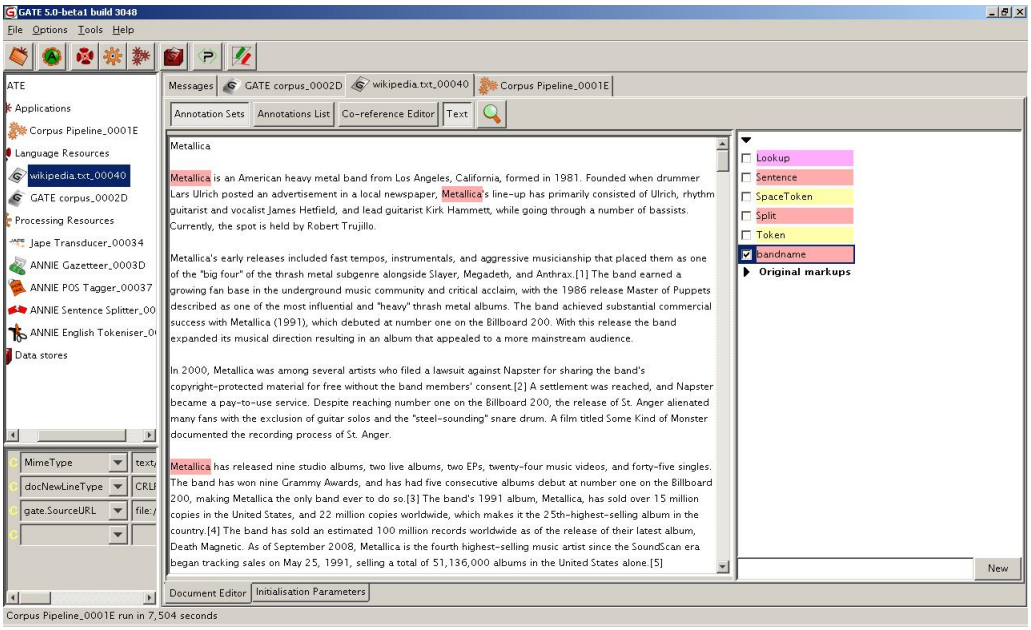

Abbildung 4.2: Hervorheben der Annotierungen in der GATE GUI

Der Prozess des IE hängt immer von zwei grundlegenden Dingen ab. Der Domäne und der unterliegenden Sprache. Jede Sprache hat dabei ihre Eigenheiten, welche bei der Verarbeitung berücksichtigt werden müssen. Um einer wirklichen *general purpose* Architektur gerecht zu werden, muss GATE auch für andere Sprachen tauglich sein. Um diese Unabhängigkeit zu gewährleisten, unterstützt GATE den Unicode Standard. Das ist das Standard Format für die Kodierung von Texten im Framework. Da aber nicht jede Sprache vom Betriebssystem bereitgestellt wird, bietet GATE die Möglichkeit, fremde Texte durch ein virtuelles Keyboard einzugeben. Falls eine Sprache nicht unterstützt wird, ermöglicht die Komponentenarchitektur von GATE eine einfache Erweiterung dieses Moduls. Eine weitere Schwierigkeit besteht in der Darstellung des Textes. GATE muss auch fähig sein die breite Palette von unterstützenden Zeichen anzuzeigen. Diese Aufgabe wird von den JAVA internen Utilities erledigt. Die grafische Oberfläche bietet dem Anwender die Möglichkeit die einzelnen Texte in einem Unicode Editor zu betrachten. Dieser unterstützt neben Unicode auch zahlreiche Ausprägungen von UTF und bietet alle rudimentären Funktionen eines Editors.

4.7 Spezifische Verarbeitungseinheiten in GATE

Die Verarbeitungseinheiten sind das Herz der Architektur. Gemeinsam formen diese die Pipeline, welche später die Dokumente verarbeitet und für die Extraktion bzw. Annotierung zuständig ist. Diese Menge an wiederverwendbaren Modulen kann vom Entwickler sowohl frei angeordnet, als auch beliebig erweitert werden. Die Menge dieser Prozessressourcen wird in diesem Framework unter dem Akronym ANNIE (A Nearly New Information Extraction System) zusammengefasst. Der Begriff steht stellvertretend für die Möglichkeiten, die mit dieser Ansammlung von Verarbeitungseinheiten erzielt werden können. Mit geringem Aufwand kann man die einzelnen Komponenten zu einem neuen System zusammenführen. In der Regel wird das System mit neuen spezifischen Modulen erweitert. Somit kann zum Beispiel die Standardpipeline für den IE Prozess aus Kapitel 4.3 realisiert werden. ANNIE besteht dabei aus folgenden Komponenten:

- Der **Tokenizer**: unterteilt den Text in SpaceTokens und Tokens. Dieser Prozess soll so effizient wie möglich ausgeführt werden. Deswegen werden weiterführende Verarbeitungsschritte auf die anderen Komponenten ausgelagert.

- Der **Sentence Splitter** generiert aus dem Text eine Sequenz an Sätzen. Dieser unterliegt der Struktur eines endlichen Automaten. Diese Einheit wird auch für den **Part of Speech** Tagger benötigt.

- Der **Part of Speech Tagger** klassifiziert die Sequenz von Sätzen bzw. Tokens. Einem Token wird dabei die jeweilige Wortart zugeordnet. Somit kann ein breiterer Bereich von Informationen einfacher abgedeckt werden.

- Der **ANNIE Gazetteer** ist eine Ansammlung von Wortlisten. Aus den Listen werden endliche Automaten generiert, welche die gespeicherten Wörter mit den Tokens abgleicht.

- Der **Transducer** besteht aus Regeln, welche von Menschen erstellt wurden. Diese Regeln annotieren in der Regel domänenspezifisches Wissen. Die Regeln werden in einer eigenen dafür konzipierten Sprache realisiert. Die Regeln bestehen aus regulären Ausdrücken, welche aus Tokens bestehen. Für den Prozess können Textelemente oder Annotierungen von vorherigen Verarbeitungsschritten (z.B. POS Tags) verwendet werden. Aus den Regeln werden wiederum endliche Automaten generiert.

- Die **orthografische Matchingeinheit** erkennt Koreferenzen bzw. Entitäten zwischen denen eine Verknüpfung besteht. Zusätzlich versucht er unklassifizierte Namen einer gegebenen Klasse zu zuordnen

- Der **Coreference Resolver** sucht Relationen vom selben Typ im gesamten Text.

Die einzelnen Ressourcen kommunizieren über die jeweiligen Annotierungen miteinander. Die Annotierungen werden in Form eines gerichteten Graphen angeordnet. Jeder Knoten stellt dabei eine Annotierung dar. Diese kann von unterschiedlichem Typ sein und verfügt über verschiedene Eigenschaften. Dies stellt jedoch nur einen kurzen Überblick über die möglichen Module dar, die Implementierungsdetails einiger Einheiten werden in einem späteren Kapitel noch detaillierter besprochen. Bei der Implementierung der Verarbeitungseinheiten wurde vor allem auf die Qualitätsmerkmale *Robustheit* und *Verwendbarkeit* geachtet. Ihr Verhalten wird durch externe Ressourcen festgelegt. In der Regel handelt es sich dabei um spezielle Grammatiken oder ein Set an Regeln. Somit können die einzelnen Komponenten ohne Programmierkenntnisse sehr einfach auf die jeweilige Aufgabe adaptiert werden. Da alle Prozessressourcen in endliche Automaten umgewandelt werden, erzielt das System auch eine akzeptable Laufzeit [26].

4.7.1 Tokenizer

Wie jeder andere Tokenizer auch, unterteilt die GATE Implementierung ebenfalls einen Text in eine Sequenz von Tokens. Der GATE Tokenizer teilt die Tokens in verschiedene Wortausprägungen, Symbole, Nummern, usw. ein. Wie diese Token unterteilt werden,

hängt von den unterliegenden Regeln ab. Jene befinden sich in einer ausgelagerten Regel Datei. Somit kann man den Tokenizer auf die jeweiligen Anforderungen abstimmen. Die Regeln des Tokenizer werden in einem bestimmten Format gespeichert. Dabei besteht jede Regel aus einem linken (Left Hand Side, LHS) und einem rechten (Right Hand Side, RHS) Teil [21]. Der linke Teil beinhaltet einen regulären Ausdruck, welcher für die eigentliche Klassifikation zuständig ist. Der rechte Teil umfasst lediglich die Zuweisung der Annotierung, welche im Dokument über den Typ entscheidet. Jede Annotierung in GATE verfügt über gewisse Eigenschaften. Diese werden in einer FeatureMap [53] gespeichert. Getrennt werden die beiden Teile durch ein " > " Symbol. Die Regel um ein Wort, welches mit einem Großbuchstaben beginnt sieht folgendermaßen aus:

```
1 "UPPERCASE_LETTER" (LOWERCASE_LETTER
2 (LOWERCASE_LETTER |DASH_PUNCTUATION|FORMAT)*)*
3 > Token;orth=upperInitial;kind=word;
```

Jene Regel befindet sich im Ruleset des Default Tokenizers von GATE. Für die regulären Ausdrücke im rechten Teil stehen die gängigen Operatoren zur Verfügung. Der Entwickler kann unter folgenden Symbolen unterscheiden:

- eine logische Oder Verknüpfung: |
- beliebig viele Elemente: *
- einmal oder keinmal: ?, [0, 1]
- mindestens einmal: +, [1,n]

Der linke Teil besteht aus dem Typ und seinen zugeordneten Eigenschaften. Der Typ muss dabei immer an erster Stelle sein. Getrennt werden die einzelnen Elemente durch einen Strichpunkt. In diesem Fall handelt es sich um einen Token, welcher über die orthografische Eigenschaft **upperInitial** und von der Art her als Wort einzuordnen ist. Grundsätzlich gibt es folgende Unterteilung an Tokens:

Word: Ein Wort besteht aus einer Sequenz von Klein- und Großbuchstaben. Je nachdem wie diese angeordnet sind, kann man das Wort noch weiter klassifizieren. Dies geschieht mit der ortografischen Eigenschaft **orth**. Generell unterscheidet man in GATE zwischen **upperInitial**, **allCaps**, **lowerCase** und **mixedCaps** Attributen.

Number: Eine Anordnung von aufeinanderfolgenden Zahlen. Diese Ausprägung verfügt über keine Zusatzeigenschaften.

Symbol: Diverse symbolische Character. GATE unterscheidet zwischen Währungssymbolen (€, $, usw.) und normalen Symbolen, wie " & ".

Punctuation: Generell handelt es sich dabei um Trennzeichen. Jedes Zeichen dieser Ausprägung wird als eigenständiger Token betrachtet. Grundsätzlich gibt es drei Gruppen von Trennzeichen: Startzeichen wie " (", Endzeichen wie ") " und andere Zeichen " : ".

Eine spezielle Ausprägung von Token ist ein **SpaceToken**. Eigentlich werden diese speziellen Tokens in GATE als eigenständiger Typ geführt. Dabei gibt es zwei Arten von SpaceTokens: Jene die nur Leerzeichen **(kind=space)** symbolisieren und jene, welche Controlcharacter **(kind=control)** beinhalten. Für die Implementierung wird der English Tokenizer von GATE verwendet, da englische Dokumente verwendet werden. Dieser Tokenizer unterscheidet sich von der Standardimplementierung durch einen nachgeschalteten Transducer. Jener fügt einzelne spezielle Konstrukte von Token zusammen, wie zum Beispiel " 'N " und " 'em".

4.7.2 Sentence Splitter

Im Grunde genommen übernimmt der Sentence Splitter eine triviale Aufgabe. Er unterteilt generell nur das Dokument in verschiedene Segmente in Form von Sätzen. Das wird mit einer Anordnung von endlichen Automaten bewerkstelligt. Zusätzlich wird diese Komponente von einer internen Gazetteer Liste unterstützt. Diese soll verhindern, dass Konstrukte wie Abkürzungen fälschlicherweise als Ende eines Satzes erkannt werden. Der Sentence Splitter erweitert das Dokument um zwei zusätzliche Typen: Durch den Typ Sentence werden die einzelnen Sätze annotiert. Der Split Typ markiert die einzelnen Delimiter, welche zur Segmentierung führen. Diese Trennzeichen beinhalten in der Regel Symbole wie Punkte oder Line Feeds. Zudem wird diese Komponente für den POS Tagger benötigt, welcher im folgenden Kapitel näher erläutert wird.

4.7.3 POS Tagger

Der Part of Speech Tagger ist Teil der lexikalischen Analyse. Diese Verarbeitungsstation kategorisiert die einzelnen Tokens in verschiedene Wortarten, wie Hauptwörter, usw. Ein POS Tagger wird entweder mit einem training- oder regel-basierten Ansatz realisiert. Die Standardversion von GATE ist eine Implementierung des Hepple Taggers[36], eine Erweiterung des Brill Taggers[10].

Der Kern des Taggers besteht aus einem Lexikon und einer Regelbasis. Diese werden durch einen statistischen Ansatz erstellt und in einer Textdatei gespeichert und können beliebig erweitert werden. Zusätzlich verfügt der Entwickler über zwei optionale Wörterbücher. Das gewünschte Lexikon kann zur Ladezeit des Moduls über einen Parameter verändert werden. Grundsätzlich verwendet die Implementierung des Systems die Standardkonfiguration der Komponente. Weitere Parameter des POS Taggers betreffen die Kodierung der Wörterbücher und die Regelbasis, die URL des Lexikons und den Regeln und das zu verarbeitende Dokument. Weiters kann man die verschiedenen Annotierungen des Taggers festlegen. In der vorliegenden Implementierung wird jedoch die Standardkonfiguration verwendet.

Die GATE Implementierung verfügt über 55 Part of Speech Tags. Diese decken die gängigsten Wortarten ab, wobei einige davon in verschiedene Symbole ($, (, #, , usw.). kategorisiert werden. Eine detaillierte Auflistung der einzelnen Tags befindet sich im Anhang des GATE Benutzerhandbuchs [21]. Werden englische Dokumente vom diesem Tagger verarbeitet, empfiehlt es sich immer den English Tokenizer von GATE vor zu schalten.

4.7.4 Gazetteer Listen

Bei den Gazetteer Listen handelt es ich um die Implementierung eines Wörterbuchs in GATE. Für jede Entitätsklasse, gibt es eine Liste von konkreten Ausprägungen. In GATE werden diese Wortlisten in Form einer Textdatei verwaltet. Jedes Wort befindet sich in einer eigenen Zeile. In der Regel ist diese Wortliste aufsteigend alphabetisch sortiert. Eine konkrete Liste könnte folgendermaßen ausschauen:

```
1 bassist
2 cellist
3 drummer
4 frontman
5 guitarist
6 keyboarder
```

Die einzelnen Listen werden in einer zentralen Datei (lists.def) verwaltet. Eine bestimmte Ausprägung wird durch eine einfache **isA** Beziehung einer Entität zugeordnet. Die Verwaltungsdatei könnte zum Beispiel folgendermaßen aussehen.

```
1 artist.lst:artist
2 genre.lst:genre
3 instrument.lst:instrument
4 role.lst:role
```

Die einzelnen Elemente werden dabei durch einen Doppelpunkt getrennt. Wobei der Dateiname immer vorangehen muss. Dieser wird vom Major Typ und vom Minor Typ (optional) gefolgt. Aus den einzelnen Listen erzeugt das Framework einen endlichen Automaten. Dieser erkennt die einzelnen Wörter der Liste später im vorliegenden Dokument. Wird eine Entität erkannt wird das Dokument um diese Annotierung erweitert. Diese beinhalten in der Regel den Typ der Annotierung, sowie Major und Minor Typ. Annotierungen, die von dieser Komponente hinzugefügt werden, können mit dem LookUp Typ angesprochen werden. Zudem besteht die Möglichkeit das Modul, durch eine breite Palette von Parametern, zu konfigurieren. Wobei nur zwei Parameter dieser Komponente bei der Implementierung eine Rolle spielen:

- listURL: Die URL der Verwaltungsdatei. Da es sich um eine auf die Domäne abgestimmte Liste handelt, ist diese viel kleiner als die GATE Standardliste.

- caseSensitive: bestimmt, ob bei der Erkennung von Wörtern zwischen Groß- und Kleinschreibung berücksichtigt wird.

Weitere Parameter betreffen die Zeichenkodierung der Listen, sowie die Festlegung eines alternativen Trennzeichens zwischen den verschiedenen Typen. Zusätzlich gibt es noch eine Menge an Laufzeitparametern, welche gesetzt werden können [21].

4.7.5 JAPE Transducer

Die Erkennung von Namen basiert auf der Integration von Regeln. Jene werden in GATE in Form eines JAPE Transducers bereitgestellt. Im folgenden Abschnitt werden die Details dieser Transducer näher besprochen. JAPE ist die Kurzform für *Java Annotation Pattern Engine*.

Generell transformiert JAPE im Kontext eines Transducers die Regeln in einen endlichen Automaten, welcher später die jeweiligen Informationen im Text erkennt. Die Implementierung von JAPE basiert auf der Common Pattern Specification Language (CPSL). Jene wurde von Appelt in seinem TextPro System [7] konzipiert. Eine Regel in GATE wird durch einen regulären Ausdruck repräsentiert. Jedoch handelt es sich dabei nicht um einen regulären Ausdruck in konventioneller Form [56]. Die Regeln erkennen vielmehr eine Abfolge an verschiedenen Tokens. Dies wird durch die interne Verwaltung eines Dokuments in GATE ermöglicht. Ein GATE Dokument unterliegt der Datenstruktur eines gerichteten Graphens. Jeder Knoten des Graphen steht dabei stellvertretend für eine Annotierung. Der generierte Automat wird intern auf diese Sequenz von Annotierungen angewendet. Da der Graph in gerichteter Form vorliegt, kann dieser

als einfache Sequenz von Tokens (eine Annotierung eines bestimmten Typs) angesehen werden und mit einem deterministischen Automaten verarbeitet werden.

JAPE Regeln sind ähnlich strukturiert wie jene eines Tokenizers. Beide bestehen aus einer Left und einer Right Hand Side (LHS, RHS). Die linke Seite beinhaltet die regulären Ausdrücke, welche für die Erkennung von Entitäten zuständig sind. Die rechte Seite hingegen ist für die Zuweisung einer Annotierung zuständig. Diese beinhaltet den Typ und eine Menge an Attributen. Getrennt werden die beiden Teile durch eine "–>" Abfolge. Die Regeln werden in Form einer Textdatei gespeichert. Jede Regeldatei besitzt dabei die Endung ".jape". [23] Grundsätzlich gliedert sich die Struktur eines JAPE Dokuments folgendermaßen:

1. Konfiguration
2. Definition von Makros
3. Regeldefinition

Jede JAPE Datei verfügt über einen Konfigurations-, einen Macro und einem Regel - Teil. Die Konfiguration regelt wie die Annotierungen gehandhabt werden bzw. welche Elemente zur Annotierung herangezogen werden. Es gibt drei verschieden Konfiguration, welche vorgenommen werden können:

- **Phase**: legt generell nur einen zentralen Namen für die Datei fest.
- **Input**: Gate verwaltet verschiedene Typen von Annotierungen (Token, SpaceToken, LookUp für Lexika, Split, usw.) Alle diese Typen können in den Regeln verwendet werden. Allerdings müssen diese im Input Teil explizit angeführt werden. Getrennt werden die einzelnen Elemente durch ein Leerzeichen.
- **Options**: Zusätzlich können einige Einstellungen bezüglich dem Verhalten der Regeln vorgenommen werden. Generell gibt es zwei mögliche Optionen: Die Control Option regelt wie die einzelnen Annotierungen abgedeckt werden. Die Debug Option zeigt bei einer bestimmten Control Option an, wenn sich mehrere Annotierungen überdecken.

Ein Beispiel für eine Konfiguration einer JAPE Datei könnte folgendermaßen aussehen:

```
1 Phase: Band
2 Input: Token Lookup Split
3 Options: control = appelt
```

Dabei werden **Token**, **LookUp** und **Split** Typen in den Regeln verwendet. Diese setzten sich in diesem Fall aus einer Anordnung dieser drei Typen zusammen.

Bevor Makros und Regeln thematisiert werden, müssen einige grundlegende Aspekte von JAPE besprochen werden. Wie bereits erläutert, handelt es sich bei Regeln um reguläre Ausdrücke von Tokens unterschiedlichen Typs. Die einzelnen Elemente können auf verschiedene Weise spezifiziert werden. Prinzipiell gibt es drei Möglichkeiten einen Ausdruck anzuordnen:

- durch einen String Vergleich, {*Token.string == "Hello"*}
- durch einen anderen Typ, { *Split* }, { *SpaceToken* }, usw.
- durch ein Attribut eines Typs, { *LookUp.majorType == instrument* }

Die einzelnen Möglichkeiten können aber auch kombiniert werden, um den Bereich etwas einzuschränken. Ein großgeschriebenes Verb, könnte folgendermaßen erkannt werden:

```
1 {Token.category = VB, Token.orth == upperInitial}
```

Grundsätzlich bietet JAPE die gängigste Vergleichsoperatoren (==, !=, <, <=, >, >=) für die Realisierung von Regeln an. Zusätzlich gibt es noch die Prädikate **contains** und **within**. Diese überprüfen ob ein Typ X einen anderen Typ Y beinhaltet. Diese können folgendermaßen in ein Element eingebunden werden:

```
1 {Token.orth == upperInitial, !Token contains {Lookup.majorType == date}}
```

In diesem konkreten Fall wird verhindert, dass ein Token fälschlicherweise einen Monatsnamen markiert. Die bisher vorgestellten Sprachelemente konzentrieren sich aber nur auf einzelne Elemente. Um mehrere Wörter anzusprechen, gibt es mehrere Möglichkeiten. Die einfachste Version um mehrere Tokens anzuordnen, ist eine Aneinanderreihung (Token.category == NNPToken.category == NNP). Dieses Konstrukt funktioniert gut, solange es sich um zwei Wörter vom POS Tag NNP handelt. Variiert jedoch die Länge der Token, erreicht man mit dieser Methode schnell seine Grenzen. Deshalb werden auch hier die gängigen Operatoren für reguläre Ausdrücke (|, *, ?, +) zur Verfügung gestellt. Nun können Token mit denselben Attributen zusammengefasst werden. Eine Anordnung von einem oder mehreren Token vom selben Typ sieht folgendermaßen aus:

```
1 ({Token.category == NNP})+
```

Kann die Anzahl der aufeinanderfolgen Token jedoch irgendwie eingegrenzt werden, bietet sich der sogenannte Kleeene Operator an. Jener definiert einen Wertebereich in dem sich die Anzahl der Wörter befindet:

```
1 ({Token.category == NNP})[1, 5]
```

Mit diesen Operatoren ist es möglich Konstrukte wie Makros oder Regeln zu bilden. Im Grunde genommen ist ein Makro eine Regel ohne zugehörigen Annotierungstyp. Makros werden grundsätzlich in Regeln verwendet, wo sich gewisse Strukturen wiederholen. Ein Beispiel dafür wäre eine Aufzählung von Personennamen im Format *Name1, Name2, ..., NameN*. Ein Name einer Person besteht allgemein betrachtet aus Vorname und Nachname. Da diese beiden Komponenten normalerweise großgeschrieben werden, könnte ein Makro für einen Namen folgendermaßen konzipiert sein:

```
1 Macro: Name
2 (
3   {Token.orth == upperInitial}{Token.orth == upperInitial}
4 )
```

Dieses Makro kann nun beliebig in den einzelnen Regeln verwendet werden. Eine einfache Regel für die Annotierung von Personennamen kann folgendermaßen konzipiert sein:

```
1 Rule: rule1
2 (
3   {Token.string == "Mr"}{Token.string == "."}
4   (
5     (Name)
6   ):Person
7 )--> :Person.PersonName = {kind = "Person", rule = "rule1"}
```

Diese Regel erkennt Muster mit der struktur "Mr. <Name>". Da aber in diesem Fall nur der Name interessant ist, muss dieser im RHS Teil für die Annotierung an der LHS gekennzeichnet werden. Dies geschieht durch die Abfolge ":Person ". Diese kann nun im linken Teil der Regel angesprochen werden (:Person.PersonName), wobei Member den Annotierungstyp darstellt. Diesem Typ können nun Attribute hinzugefügt werden. In diesem Fall sind das **kind** und **rule**. Jedem Attribut wird ein bestimmter Wert zugewiesen. Der zugeordnete Typ kann später so in der grafischen Oberfläche von GATE im Document Viewer angesprochen werden.

Bei sehr vielen Regeln kann der Fall auftreten, dass eine Entität von mehreren Regeln annotiert wird. Für diese Situationen kann ein unterschiedliches Verhalten konfiguriert werden. Dies geschieht im Konfigurationsteil mit der Control Option. Der Entwickler kann zwischen folgenden Verhalten wählen: **all**, **appelt**, **brill**, **first** und **once**. **First**

und **once** sind eigentlich ziemlich selbsterklärend. **Brill** und **all** triggern alle Regeln. Allerdings wird die Abfolge im Graphen der einzelnen Annotierungstypen unterschiedlich verwaltet. Das **Appelt** Verhalten hingegen ist etwas komplizierter. Es unterliegt drei Regeln:

- Bei überlappenden Annotierungen wird die längste ausgewählt
- Annotieren mehrere Regeln dieselbe Entität wird jene mit der höchsten Priorität ausgewählt.
- Bei Regeln mit derselben Priorität wird jene ausgewählt, welche zuerst definiert wurde.

Um diesen Prozess zu beeinflußen kann der Entwickler jeder Regel eine Priorität zuweisen. Diese kann dazu verwendet werden um Mehrdeutigkeiten zu vermeiden:

```
1 Rule: PriorityRule
2 Priority: 20
```

Wird einer Regel keine Priorität zugeordnet, hat diese den Standartwert -1. Ein Prioritätsvergleich wird aber immer nur zwischen den Regeln einer JAPE Datei durchgeführt. Um eine schnelle Ausführung der Regeln zu gewährleisten, empfiehlt das GATE Benutzerhandbuch [21] Operatoren wie + und * so sparsam wie möglich einzusetzen. Wenn die Anzahl der Token abgeschätzt werden kann, besteht die Möglihkeit folgende Konstrukte zu verwenden:

```
1 ({Token})[0, 3] //oder
2 ({Token})?({Token})?({Token})? //anstatt ({Token})*
```

Ein weiterer Weg um Laufzeit einzusparen, ist die Reduzierung der Input Konfiguration. Je mehr Typen sich in den Regeln befinden, desto komplexer wird der endliche Automat. Dies betrifft vor allem Typen wie SpaceToken. SpaceToken können dazu verwendet werden um Zeilenumbrüche zu erkennen. Somit müssen alle Leerzeichen im Text berücksichtigt werden. Eine Alternative für dieses Problem, bietet der Split Typ des Sentence Splitters. Jener erkennt einfach und unkompliziert Zeilenumbrüche ohne die Laufzeit stark zu beeinflussen.

Kapitel 5

Implementierung

Im vorigen Kapitel wurden die verschiedenen Eigenschaften von IE Systemen besprochen und die Probleme der vorliegenden Arbeit identifiziert. In diesem Kapitel wird die konkrete Implementierung des Systems besprochen. Für diese Probleme wird in diesem Kapitel der gewählte Lösungsansatz skizziert. Der detaillierten Beschreibung der einzelnen Komponenten im System, geht eine präzisierte Auflistung der jeweiligen Anforderungen voran. Diese konkretisieren die Ziele des Systems. Danach werden die einzelnen Aspekte des Trainingscorpus, sowie die Einzelheiten der Extraktionsregeln besprochen. Dabei wird weniger auf die spezifischen Aspekte der Implementierung eingegangen, sondern der Prozess der Regelerstellung dokumentiert.

Aufbauend auf GATE wird die Architektur des vorliegenden Systems näher erläutert, welche sich grob in Corpus, Pipeline und Wissensbasis gliedert. Dabei werden die Designentscheidungen für die Konfiguration der Pipeline besprochen, sowie technische Details der jeweiligen Verarbeitungseinheit. Zudem werden die Eigenschaften der verwendeten Regelsprache ausführlich behandelt. Dem folgt eine Beschreibung der Wissenbasis, welche den Extraktionprozess unterstützt.

Am Ende dieses Kapitel verfügt man über eine Basisarchitekur, welche mit der Unterstützung der erforderlichen Regeldateien musikspezifische Entitäten annotiert. Diese dient als Ausgangspunkt um verschiedene Strategien zu realisieren. Darunter fallen Komponenten zur automatischen Regelextraktion, aber auch Module zur Klassifikation einzelner Seiten oder zur Extraktion von Relationen.

5.1 Anforderungen an das System

Die Aufgabe ist die Implementierung eines IE-Systems für die Domäne Musik. Das Hauptaugenmerk liegt auf der Extraktion von Entitäten, welche bandspezifische Informationen beinhalten, wie zum Beispiel Labels, Mitglieder einer Band, usw. Die Arbeit beschränkt sich auf Bandnamen, Mitglieder und alle Subkategorien von Bandveröffentlichungen (Alben, Singles, LPs, DVDs, Compilations). Trotz des Fokus auf diese drei Entitäten, muss die Erweiterbarkeit hinsichtlich der Extraktion von Information unbedingt gewahrt bleiben. Der Anwender kann die einzelnen Module für die jeweilige Extraktion beliebig hinzufügen bzw. entfernen. Das extrahierte Wissen wird in einer einfachen Form gespeichert und für weitere Extraktionsprozesse genutzt. Das Eigenschaften dieses Systems führen zu folgenden Designentscheidungen:

5.1.1 Sprache

Der Extraktionsprozess beschränkt sich auf die Sprache Englisch. Das hat vor allem folgende Gründe. Zum einem ist Englisch wohl die am Weitesten verbreitete Sprache im Internet. Somit kann sehr einfach ein großes Set an spezifischen Dokumenten aufgebaut werden. Vor allem existieren sehr viele englischsprachige musikorientierte Medien. Zum anderen konzentrieren sich die meisten Implementierungen von Verarbeitungseinheiten auf die englische Sprache. Nicht zuletzt ermöglicht es dem Autor dieser Arbeit, die Inhalte zu verstehen, was zum Beispiel bei Französisch nicht der Fall wäre.

5.1.2 Grad der Strukturierung

Die extrahierten Informationen können je nach Zweck und Aufgabe des Systems von Dokumenten mit einem unterschiedlichen Grad an Strukturierung bezogen werden. Grundsätzlich konzentriert sich diese Anwendung auf die Annotierung von freien Texten. Da das Internet aber Unmengen an Informationen größtenteils frei zur Verfügung stellt, soll dem Anwender aber auch die Möglichkeit geboten werden, Websites in den Corpus zu integrieren. Jene Seiten müssen jedoch in einem Vorverarbeitungsschritt von sämtlichen Markup Elementen befreit werden.

5.1.3 Art der Extraktion

Nachdem die Struktur der Dokumente bereits spezifiziert wurde, können nun die Anforderungen der Extraktion näher spezifiziert werden. Diese ist in der Regel abhängig von

der vorliegenden Struktur. Wie bereits oben erwähnt, konzentriert sich diese Arbeit auf die Extraktion von einfachen Entitäten. Dabei steht die Erkennung von Bandnamen, Mitgliedern und Medien im Vordergrund. Generell soll das System bzw. die Extraktionskomponente offen gehalten werden, um auch die Extraktion von einfachen Verknüpfungen zu ermöglichen.

Die grobe Beschreibung im oberen Teil hat die Design Entscheidung für diese Anforderung eigentlich gleich vorweg genommen. Da weitere Regeln für die Extraktion von anderen Informationen hinzugefügt werden sollen, wird der Knowledge Engineering Ansatz bevorzugt. Das System kann je nach Aufgabe um eine beliebige Anzahl von Regeln erweitert werden. Diese können dann auf den Typ der Extraktion zugeschnitten werden.

5.1.4 Struktur der Wissensbasis

Ein großes Problem bei regelbasierten Ansätzen liegt am Design der jeweiligen Regeln. Diese sind in dieser vorliegenden Arbeit auf einem hohen Precision abgestimmt. Dabei sollte jedoch der Recall Wert nicht vernachlässigt werden. Daher wird in der Wissensbasis ein molekulare Ansatz implementiert. Die Struktur der Wissensbasis soll so einfach wie möglich gehalten werden und auch für Menschen lesbar sein. Auf komplexe Prozesse wie Inferenz, usw. [38] wird bewusst verzichtet. Das sichere Wissen kann gespeichert werden und bei weiteren Extraktionsprozessen wieder abgerufen werden. Die Schnittstelle wird bewusst offen gehalten, um auch eine Art Regelbasis zu implementieren. Mit der Implementierung dieser Schnittstelle lassen sich auch Hybrid- oder reine Automatic Training-Ansätze realisieren und in die Anwendung miteinbinden. Aber auch eine Implementierung einer Komponente, welche Relationen erekennt ist damit möglich. Die verschiedenen Ausprägungen der Schnittstelle können dann nach Belieben in das System ein- bzw. ausgeklinkt werden.

5.2 Trainingscorpus

Ein Trainingscorpus beinhaltet in der Regel domänenspezifische Inhalte. In diesem Kontext umfasst das alle Dokumente, welche einen Musikbezug haben. In dieser Arbeit wird ein Trainingscorpus für eine fundamentale Arbeit benötigt: Den Aufbau einer Regelbasis. Für diese Aufgabe bedient man sich dem vorhandenem Wissen (Bandnamen, Bandmitglieder und Medien) im Trainingscorpus.

Die langwierigste Aufgabe besteht bei der Erstellung eines Trainingscorpus im Sammeln und Annotieren von relevanten Dokumenten, da aktuell kein vorannotierter Corpus oder eine Kollektion von Dokumenten in dieser Domäne zur Verfügung steht. Somit müssen die einzelnen Texte händisch zusammengetragen werden. Ein Möglichkeit wäre Artikel aus verschiedenen Musikmagazinen zusammen zu tragen. Die wohl bessere und einfachere Alternative besteht darin, die Dokumente aus dem Internet zu beziehen. Es existieren vor allem Unmengen an Radiosendern und Musikmagazinen im Internet, welche ihre Artikel auch online zur Verfügung stellen. Zudem kommen noch Billboard Charts, offizielle Bandseiten, Fanseiten und Wikipedia Einträge.

Grundsätzlich werden Bandbiographien für den Corpus verwendet, da diese wohl die meisten Ausprägungen von Bandname, Bandmitglieder und deren Medien beinhalten. Sie werden von Konzertberichten, Rezensionen und allgemeinen Nachrichtenartikeln ergänzt. Allgemein werden die Dokumente nach unterschiedlichen Bands kategorisiert. Das Spektrum an Bands soll verschiedene Genres (alternative, electronic, metal, pop, punk, rock, world) abdecken. Für die Erstellung eines Trainingscorpus werden hauptsächlich drei unterschiedliche Quellen herangezogen:

- **en.wikipedia.org:** Es existiert eigentlich für jede bekannte Band ein umfangreicher Biographieeintrag. Lediglich bei kleineren, unbekannten Bands sind die Artikel etwas kürzer.

- **allmusic.com:** stellt ebenfalls Biographien für jeden erdenklichen Künstler zu Verfügung. Diese sind aber in der Regel etwas kürzer, als jene von wikipedia. Allerdings werden diese von professionellen Redakteuren verfasst.

- **last.fm:** Für jeden Bandnamen existieren Seiten für Biographie, Alben, Konzerte, usw. Dabei werden wiederum die Biographien eines Artists verwendet. Jeder registrierte Benutzer kann dabei das Bandprofil bearbeiten.

- ein Mix aus Billboard Charts, FM4 und Rolling Stone Magazine Artikeln

Die Betrachtung der Daten zeigt, dass last.fm Einträge nicht wirklich gut für diesen Zweck geeignet sind. Dies hat vor allem zwei Gründe: Für jede Entität (Bandname, Album, Song) existiert nur eine Seite. Existieren mehr Ausprägungen einer Entität (gleicher Name), werden diese einfach in diese Seite integriert. Außerdem sind die meisten Biographie-Einträge zusammenkopierte Fragmente von Wikipedia. Da es sich um Webdokumente handelt und das System auf freie, unstrukturierte Texte ausgelegt ist, müssen die Dokumente zuerst von den einzelnen HTML Tags befreit werden. Diese Aufgabe übernimmt die Entwicklungsumgebung von GATE, welche es ermöglicht, Dokumente aus dem Internet zu beziehen und dabei alle Markupelemente zu entfernen.

Diese werden in einer einfachen Ordnerstruktur nach Bandnamen kategorisiert. Für jede Band gibt es drei verschiedene Dokumente. Bei einer Anzahl von 82 Bands kommt man somit auf 246 Dokumente. Dazu kommen noch allgemeine Dokumente, bestehend aus Konzertberichten und anderen relevanten Artikeln (insgesamt 14 zufällig ausgewählte Dokumente, die sich im Zuge der Dokumentenrecherche inhaltlich als ïnteressant"herausgestellt haben). Somit wird eine Gesamtanzahl von 260 Dokumenten erreicht.

5.3 Extraktionsregeln

In diesem Kapitel wird der Prozess der Regelerstellung detaillierter beschrieben. Die einzelnen Regeln werden mit einem Knowledge Engineering Ansatz erstellt und werden in Form von JAPE Regeln in das System eingebunden. Grundsätzlich muss sich der Knowledge Engineer mit folgenden drei grundlegenden Problemen auseinandersetzen: Erstens, die Hauptschwierigkeit besteht darin überhaupt Strukturen zu finden, welche auf die jeweilige Entität schließen lässt. Der Entwickler muss sich also mit der vorliegenden Domäne beschäftigen. Zweitens, benötigen solche Strukturen gewisse spezifische Indikatoren, welche auf eine gesuchte Entität hinweisen. Drittens, verfügen die eigentlichen Entitäten über ein eigenes Format, welches von den Regeln vollständig abgedeckt werden muss. Das kann in diesem Fall aber sehr stark divergieren. Zudem existieren keine publizierten Ansätze, welche sich mit dieser Domäne beschäftigen. Einzig in [48] werden einige Extraktionsregeln für Bandmitglieder vorgestellt. Die anderen Strategien [45, 31, 51] verwenden alle Standardansätze um Strukturen zu annotieren, die für diese Problemstellung nur bedingt funktionieren. In den folgenden Abschnitten werden die Lösungsansätze für diese drei Thematiken etwas näher erläutert:

5.3.1 Aufbau der Regelbasis

Ein Knowledge Engineer muss über ein bestimmtes Expertenwissen in der jeweiligen Domäne verfügen, bevor er die einzelnen Regeln vom Text herauslesen kann. In diesem konkreten Fall muss sich der Entwickler mit den einzelnen Mitgliedern und Medien einer Band vertraut machen, um diese später auch im Text zu erkennen. Der Aufbau der Regelbasis erfolgt in einem zyklischen Modell, ähnlich dem bekannten Spiralmodell [13] und läuft bei jeder Domäne gleich ab. Dabei unterscheidet man zwischen drei Phasen:

1. Lernphase: Der Entwickler liest die Dokumente durch und betrachtet die Struktur von den Abschnitten in denen Entitäten vorhanden sind. Daraus kristallisieren

sich die gängigsten Muster. Diese werden in der nächsten Phase in die Regelbasis eingebunden.

2. Regelphase: Die auffälligsten und häufigsten Regeln werden in das System eingebunden. Diese werden im Anschluss auf den Corpus angewendet. Die Resultate diese Regeln werden in der nächsten Phase evaluiert.

3. Debugphase: Je nachdem wie gut eine einzelne Regel in den Testläufen abschneidet, wird in dieser Phase entschieden ob sie gänzlich verworfen, adaptiert oder permanent in die Regalbasis eingebunden wird. Ist die Regel sehr präzise, wird diese gelockert, um ein breiteres Spektrum abzudecken. Jedoch macht dieser Schritt die Regel anfälliger für nichtrelevante Informationen.

Dieser Zyklus wird solange wiederholt bis die Regelbasis angemessene Ergebnisse liefert. Allerdings unterliegt dieser Prozess sehr vielen subjektiven Faktoren. Der Erfolg der Regelbasis hängt sehr stark von den Fähigkeiten des Entwicklers ab, einzelne Regeln zu bewerten und diese in angemessener Form zu adaptieren. Eine Regel besteht aus zwei Teilen: Einem variablen und einem konstanten Teil. Ein Beispiel dafür wäre diese Regel:

drummer <Member>

Der konstante Teil ist grundsätzlich immer der Indikator für eine gewisse Entität der Domäne. Der variable Teil steht stellvertretend für die konkrete Ausprägung der Entität im Dokument und verfügt zum Teil auch über eine eigene Substruktur. In der vorliegenden Arbeit wurde der erstellte Testcorpus mit dieser Strategie durchlaufen. Die spezifischen Einzelheiten der variablen bzw. konstanten Teile einer Regel werden in den nächsten Abschnitten detaillierter beschrieben.

5.3.2 Indikatoren

Ein Indikator in diesem Kontext lässt auf eine möglicherweise interessante Passage schließen. Zusätzlich übernehmen Indikatoren bzw. Schlag- oder Hinweisewörter auch die Aufgabe, die variablen Teile bzw. die Entität zu begrenzen. In der idealen Form wird der variable Teil bzw. die Entität einer Regel von einem Wort oder Symbol sowohl am Anfang als auch am Ende begrenzt (zum Beispiel: ”guitarist” <Member> ”,”). Dabei muss nicht jeder Delimiter auch einen Hinweis auf eine Entität geben. Ein Beispiel dafür könnte folgendermaßen aussehen:

album ”<Media>”

In diesem Fall ist das Wort ”album” der Indikator und die beiden Anführungszeichen die begrenzenden Elemente. Solche Konstrukte sind ideal, da sie dem Entwickler ermöglichen die Entität optimal abzudecken. Allerdings wird das Feld dabei sehr stark eingeschränkt. Manchmal ist es gar nicht möglich, dass eine Regel sowohl am Anfang als auch am Ende über einen konstanten Teil (in diesem Fall die Anführungszeichen) verfügt, wie zum Beispiel:

left <Band>

Diesem Konstrukt könnte alles Mögliche folgen. Die einfachste Variante für den Nachfolger der Enität <Band> wäre ein Trennzeichen (” .”, ” ,”). Es könnte aber auch durch ”to join” oder andere Phrasen ergänzt werden. Außerdem beinhaltet die oben angeführte Regel eine weitere Schwierigkeit, da in diesem Fall der Unsicherheitsfaktor größer ist. Es muss dabei nicht zwingend notwendig sein, dass es sich im vorliegenden Fall (left, join) bei der nachfolgende Struktur tatsächlich um eine Band handelt. Es könnte auch eine Firma oder ein Sportverein folgen. Jeder Indikator verfügt über einen unterschiedlichen Faktor an Unsicherheit. Generell ist es aber sehr schwierig sehr sichere Phrasen in freien Texten zu finden, da diese oft Variationen beinhalten, welche gar nicht in den Sinn des Regelerstellers kommen. Die Indikatoren werden auf drei verschiedene Arten in die Extraktionsregeln eingebunden: In der einfachsten Form, werden konkrete Phrasen mit Strings verwendet:

drummer <Member>

Jedoch bietet diese Form nicht allzu viele Freiheiten. Es müsste eine Regel für drummer, frontman, etc. konzipiert werden. Für solchen Fälle werden die Gazetteer Listen von GATE herangezogen. In diesem konkreten Fall werden alle generischen Bezeichnungen von Bandmitgliedern in der Liste role.lst verwaltet. Diese kann folgendermaßen in die Regel integriert werden:

{ LookUp.majorType == role } <Member>

Außerdem werden auch POS Tags als Trennzeichen verwendet. Ein Beispiel dafür wäre die Phrase ”best Queen album ”. Dieses Adjektiv kann aber auch variieren (worst, successful, usw.). Um alle Varianten abzudecken, wird einfach das Adjektiv durch seinen zugehörigen POS Tag ersetzt:

{ Token.category == JJ } <Band> { Token.string == album }// best Pearl Jam album

In diesem Fall könnte auch noch das Wort album durch eine Gazetteer Liste ersetzt werden, welche Wörter wie track, single, usw. beinhaltet. Generell werden für Bandnamen, Member und Medientitel folgende Indikatoren verwendet:

- Bandname: band, bands, headlined, recorded, usw.
- Member: left, joined, replaced by, bassist, ...
- Media: album, single, compilation, EP, LP, ...

Wie diese Indikatoren in den einzelnen Regeln tatsächlich verwendet werden, kann dem Anhang entnommen werden. Dieser beinhaltet alle Regeln, welche im System verwendet werden.

5.3.3 Annotierung von Namen

Neben der Bestimmung von adäquaten Indikatoren ist die Annotierung von Namen wohl die schwierigste Aufgabe. Dies hängt stets sehr stark von der Domäne ab. In diesem Kontext handelt es sich um eine nicht ganz so triviale Aufgabe einen Namen im Dokument zu erkennen. Vor allem Bandnamen und Medientitel können sehr stark variieren. Ein Beispiel dafür wäre:

- Bandnamen: No Use For A Name, We Butter The Bread With Butter, ...
- Medientitel: Killing Is My Business... and Business Is Good!, Whatever People Say I Am, That's What I'm Not, ...

Standardansätze [35, 51] empfehlen für die Namenserkennung eine Sequenz an Eigennamen (NNP bzw. proper noun POS Tag). Dies funktioniert in diesem Fall aber nur bei sehr einfachen Ausprägungen (Bad Religion, Queen, usw.) und bei Namen von Bandmitgliedern, da es sich dabei um Abfolgen von Eigennamen handelt. Bei Bandnamen wie The Rolling Stones, welche nicht nur aus NNP POS Tags ("The") bestehen, würde dieser Ansatz aber schon an seine Grenzen stoßen. Dieses Problem betrifft also grundsätzlich nur Bandnamen und Medientitel, da diese aus Variationen von verschiedenen Wortarten bestehen. Um diese stark variierenden Strukturen zu erfassen, wird ein endlicher Automat in einem "Trial and Error" Prozess mit zahlreichen Eigenschaften (POS Tags bzw. Groß- oder Kleinschreibung) auf diese Problemstellung zugeschnitten. Der Eigennamenansatz liefert dabei einen guten Ausgangspunkt. Bei den oben angeführten Beispielen fällt eines auf: Die Elemente eines Bandnamens werden durchwegs

großgeschrieben. Diese Eigenschaft trifft ebenfalls auf englische Eigennamen zu. Somit könnte die Sequenz aus Eigennamen durch eine Sequenz aus großgeschriebenen Wörtern ersetzt werden. Komplexere Bandnamen weisen in den Dokumenten allerdings oft unterschiedliche orthografische Eigenschaften auf. Der Mittelteil der Band "System Of A Down" wird in vielen Texten oft kleingeschrieben. In diesem Fall würde der aktuelle Ansatz versagen. Betrachtet man die Struktur von komplexeren Entitäten in dieser Domäne, fällt auf, dass vor allem der Mittelteil sehr stark variiert. Für einen komplexen Bandnamen lässt sich folgendes Format festlegen:

<StartToken>(<MiddleToken>)*<EndToken>

Der Starttoken unterliegt nur einer fundamentalen Regel. Er muss großgeschrieben werden. Diese Einschränkung gilt auch für den Endtoken. Allerdings kann dieser auch Nummern und Symbole (!, ?) beinhalten. Der Mittelteil ist der Knackpunkt dieser Struktur. Dieser soll so offen wie möglich gehalten werden, um möglichst viele Entitäten abzudecken. Allerdings stößt man bei der Konzipierung auf folgende Probleme:

Beistriche: Diese werden prinzipiell in einem Text als Delimiter verwendet. Jedoch können in diesem Kontext Beistriche auch als Teil des Bandnamens bzw. Medianamens angesehen werden. Zum Beispiel das aktuelle Dredg Album: The Pariah, The Parrot, The Delusion

Diese Struktur könnte durchaus als Liste durchgehen. Da eine Regel nicht zwischen einem Beistrich als Trennzeichen und als Teil einer Entität unterscheiden kann, muss man einen Kompromiss eingehen. In diesem Fall wird ein Beistrich immer als Delimiter behandelt. Entitäten, welche Beistriche beinhalten, können nur erfasst werden, wenn sie zwischen zwei konstanten Teilen gekapselt werden ("<Media> ").

Bindewörter: Dieselbe Problematik besteht bei Konjunktionen wie und bzw. oder. Diese Wörter können ebenfalls als Teil eines Bandnamens angesehen werden. Ein Beispiel wäre der Black Sabbath Abkömmling Heaven and Hell. Auch hier muss ein Kompromiss gefunden werden. Wird ein Bindewort kleingeschrieben, so ist es unmissverständlich als Trennzeichen zu verstehen. Werden diese jedoch großgeschrieben oder als Symbol (&) dargestellt, betrachtet man diese in diesem Fall als Teil der Entität.

Symbole: Zusätzlich zur Problematik von Beistrichen und Bindewörtern kommen noch Symbole, wie /, !, ?, :, etc. hinzu. Ein guter Vertreter wäre hier wohl die Guns N' Roses EP "Live ?!*@ Like a Suicide". Für Bandnamen werden die Symbole !, / verwendet, um Bands wie AC/DC abzudecken. Bei Medientitel wird auf das " / " Symbol verzichtet. Jedoch kommen noch Doppelpunkte hinzu.

Zusätzlich zu diesen Sonderfällen besteht der Mittelteil aus folgenden Token:

1. großgeschriebene Verben, Nomina, Eigennamen, Adjektive, Adverben und allen möglichen Pronomina

2. klein- oder großgeschriebene Artikel und Präpositionen

Dieser endliche Automat deckt die gängigsten unkonventionellen Entitäten ab. Allerdings deckt diese Struktur nicht Bandnamen wie R.E.M oder Albentitel wie St. Anger ab. Um auch solche Ausprägungen abzudecken, gibt es eine Sonderbehandlung für Abkürzungen in <StartToken>, <MiddleToken> und <EndToken>: Wird ein großgeschriebener Token mit maximal zwei Buchstaben von einem Punkt gefolgt ist er Teil der Entität. Da diese Struktur aber nur Entitäten mit mindestens zwei Wörtern abdeckt, muss diese um einen Sonderfall erweitert werden:

((<StartToken> (<MiddleToken>)* <EndToken>) | <SingleToken>)

Ein SingleToken unterliegt speziellen Einschränkungen: Er muss ebenfalls großgeschrieben werden und besteht nur aus den verschiedenen POS Tags (NN, NNP, NP, ...). Ein SingleToken darf nicht aus POS Tags wie Artikel oder kleingeschriebenen Token bestehen. Bei Strukturen von Bandmitgliedern sieht die Sache anders aus. Personennamen bestehen in der Regel aus Vorname und Nachname. Diese werden in fast allen Fällen als Eigenname getaggt. Allerdings werden die Namen oft mit einem Spitznamen oder Künstlernamen erweitert. Ein Beispiel dafür wär der ehemalige Gittarist der Band KoRn, Brian "Head" Welch. Hinzu kommen noch Namen wie Nick O'Malley, aber auch Abkürzungen (Schlagzeuger D.H. Peligro) und zweite Vornamen (John Paul Jones). Grundsätzlich führen diese Eigenschaften wieder zur folgenden bekannten Struktur:

<StartToken> (<MiddleToken>)? <EndToken>

Start und Endtoken sind wiederum großgeschriebene Eigennamen. Der Mittelteil beinhaltet wiederum verschieden Variationen: "<Token>(<Token>)? ", O', Abkürzungen und einen großgeschriebenen Eigenamen. Einfache Namen, welche nur aus einem Token bestehen (zum Beispiel das Red Hot Chilli Peppers Mitglied "Flea"), werden in der vorliegenden Form des Systems vernachlässigt. Die vorgestellten Strukturen wurden als Makro in JAPE implementiert. Zusammen mit den gefundenen Indikatoren bilden sie die Regelbasis für das vorliegende System.

5.4 Komponenten des Systems

Die Architektur des vorliegenden Systems besteht aus drei Komponenten: Dem Corpus, der Pipeline und der Wissensbasis. Die ersten beiden Module werden von GATE bereit gestellt. Diese werden um die Struktur der Wissensbasis erweitert. In den folgenden Abschnitten werden die Implementierungsdetails des Systems besprochen. Dem geht noch das Design der Architektur voraus, welche über die GATE Architektur gelegt wird. Diese modifizierte Architektur wird textuell und in Form von UML Diagrammen skizziert. Allerdings ist die API von GATE so umfangreich, dass die erforderlichen Komponenten nur mehr durch einige Schnittstellen zusammengeführt werden müssen. Wie diese Einheiten gekapselt werden und welche Designüberlegungen dahinter stehen, wird im folgenden Abschnitt näher besprochen. Außerdem wird auf die Aspekte des GATE Standard Corpus eingegangen. Jener ermöglicht eine einfache und flexible Verwaltung von Dokumenten mit unterschiedlichen Eigenschaften. Die Pipeline ist wohl der wichtigste Teil der Implementierung. Sie verwaltet die einzelnen Prozessressourcen und bestimmt deren Abfolge. Anhand der Designüberlegungen wird eine auf die Problemstellung zugeschnitte Pipeline konfiguriert, die auch den Extraktionsprozess durchführt. Dieser Vorgang wird durch die Wissensbasis unterstützt, welche die extrahierten Entitäten in einfacher Form verwaltet und für weitere Annotierungen verwendet werden kann.

5.5 Architektur des Systems

In diesem Abschnitt wird die Einbettung der GATE Architektur in das vorliegende Musik IE System beschrieben. Die Komponenten der allgemeinen Architektur werden dabei um die Struktur einer Wissensbasis erweitert. Zusätzlich werden die Verarbeitungsstationen von GATE in eine einfache Schnittstelle gekapselt und fundamentale Elemente wie Entitäten und die Wissensbasis an die Anforderungen der Anwendung angepasst.

Die API von GATE stellt alle erforderlichen Komponenten wie Pipeline und Corpus zur Verfügung. Diese müssen für die Eigenschaften der Anwendung konfiguriert werden. Zusätzlich existieren Hilfsklassen um die erforderlichen Module in die Anwendung zu laden oder auf Annotierungselemente eines Texts zuzugreifen. Grundsätzlich wird eine einfache Architektur über das durchaus vielfältige und komplexe Konstrukt von GATE gelegt. Die vorliegende Architektur erfüllt dabei die identifizierten Anforderungen an das System. Die Architektur muss die Implementierung einer IE Anwendung für musikspezifische Daten ermöglichen. Dabei kann es sich sowohl um ein Kommandozeilenprogramm oder eine Anwendung mit GUI als Frontend handeln. Die annotierten

Daten werden über eine zentrale Einheit angesprochen. Die Wissensbasis sorgt dafür, dass bereits extrahiertes Wissen im weiteren Extraktionsprozess automatisch in einem neuen Dokument erkannt wird. Die vorliegende Architektur lässt sich folgendermaßen darstellen:

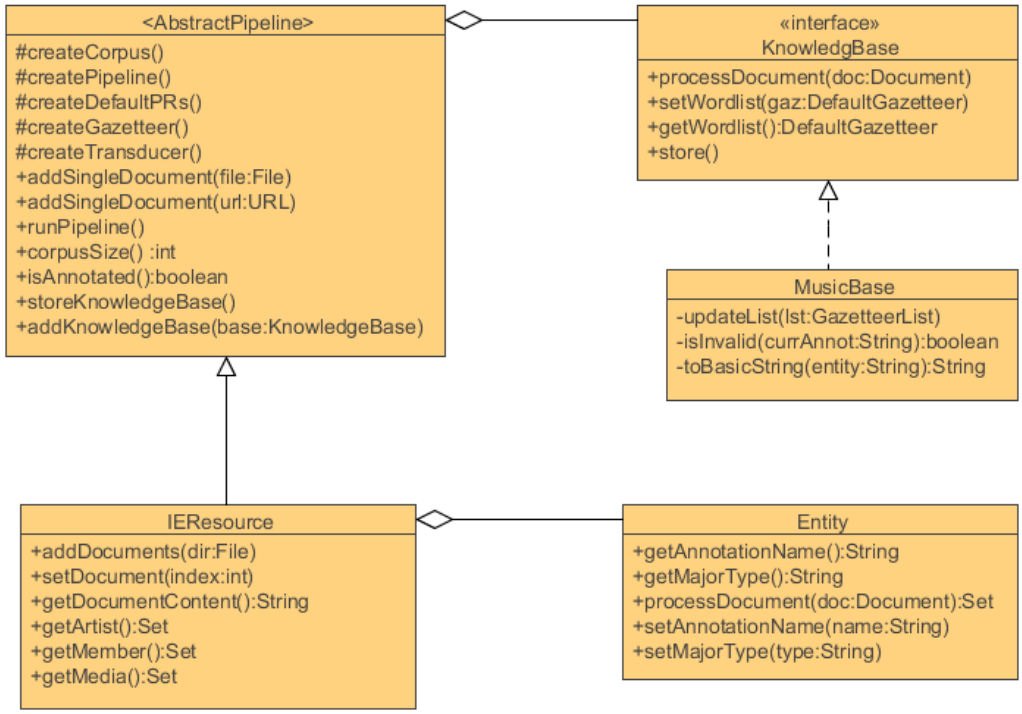

Abbildung 5.1: Klassendiagramm der Basiskonfiguration

Die **AbstractPipeline** Klasse führt alle erforderlichen Komponenten zusammen. Sie legt die Struktur der Pipeline fest und ermöglicht die Erstellung des Corpus. Zusätzlich kann man verschiedene Ausprägungen der Wissensbasis hinzufügen. Wie die einzelnen Dokumente schließlich abgearbeitet werden, hängt von der gewählten Implementierung ab, da in der GATE API verschiedene Arten von Pipelines existieren. Um die erforderlichen Komponenten in die Anwendung einzubinden, stellt GATE Funktionen bereit. Die Annotierungen werden mittels der Entity Klasse ausgewählt. Eine Implementierung dieser Komponente traversiert den Annotierungsgraph von GATE. Grundsätzlich konzentriert sich die Entität auf zwei Arten von Annotierungen: Annotierungen der JAPE Regeln und jene der Wissensbasis bzw. Gazetteer Listen. Jene umfassen die Gesamtheit einer speziellen Entität in einem Dokument. Die Einbindung einer Wissensbasis ist allerdings optional. Die Struktur einer Wissensbasis kann über die Funktion addKnowledgeBase in die Anwendung integriert werden. Nachdem die Verarbeitungselemente (Tokenizer, POS Tagger, etc) von GATE ein Dokument verarbeitet haben, werden die implementierten Operationen der Wissensbasis daruf ausgeführt. Diese verfügt über

eine Funktion um ein GATE Document zu verarbeiten. Dafür kann ebenfalls die Entity Klasse verwendet werden. Die extrahierten Wörter werden hier ebenfalls durch eine einfache Strategie bewertet und anschließend entweder permanent in die Wissensbasis aufgenommen oder verworfen. Zusätzlich kann die persistente Speicherung des bezogenen Wissens implementiert werden. Aus dieser Architektur lassen sich nun verschiedene Anwendungen erstellen. Im Zuge der vorliegenden Diplomarbeit wurde eine grafische Anwendung für die Extraktion von Bandnamen, Bandmitglieder und Medien entwickelt. Das Design der grafische Oberfläche orientiert sich dabei an GATE.

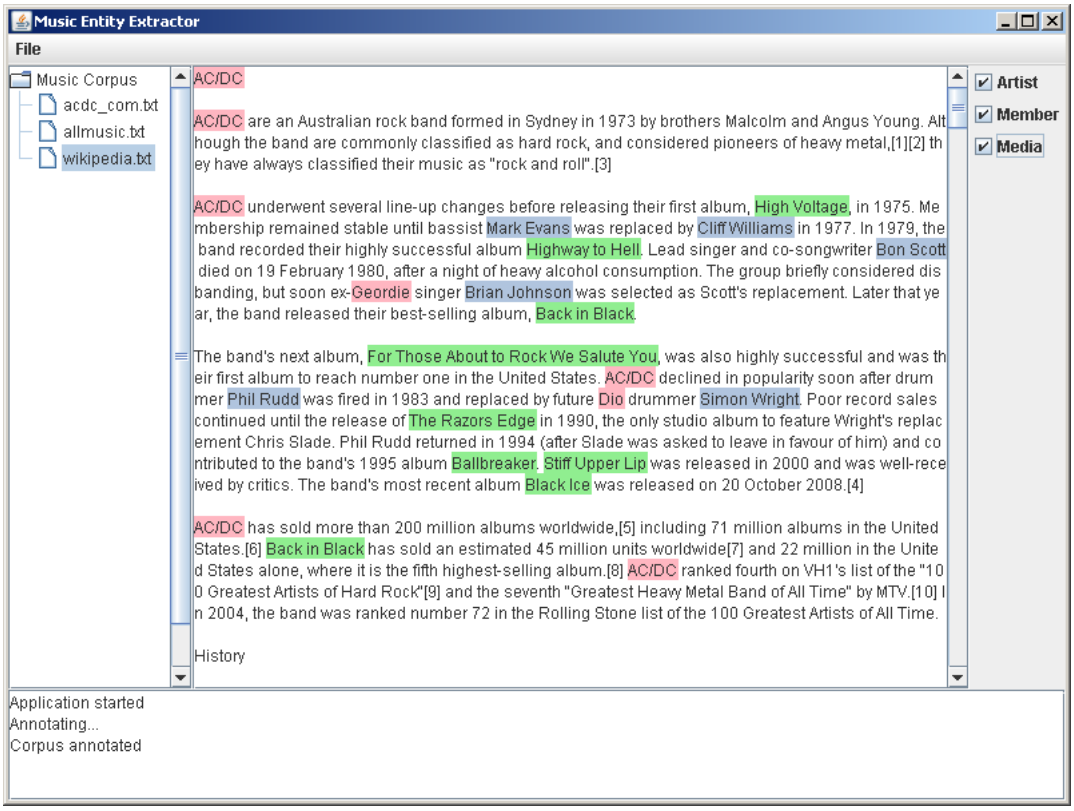

Abbildung 5.2: Grafische Oberfläche der Applikation

Das Hauptfenster zeigt den ausgewählten Text an. Rechts werden alle zugehörigen Dokumente des Corpus in einer JTree Komponente angezeigt. Wird dabei ein Dokument ausgewählt, ändert sich der Inhalt des Hauptfensters. Auf der rechten Seite befinden sich die einzelnen Ausprägungen einer Entität. Diese können wieder durch eine Checkbox angeklickt werden. Die jeweiligen Annotierungen werden dann im Text durch einen farbigen Balken hervorgehoben. Im Menü kann man Dokumente in den Corpus einfügen, sowie die gewonnenen Informationen der Wissensbasis für weitere Sitzungen speichern. Dieses Wissen ist beim Neustart der Anwendung sofort vorhanden.

5.6 Verwaltung der Dokumente im GATE Corpus

Da das System größere Mengen an Daten bzw. Dokumenten verarbeitet, müssen diese an einer geeigneten zentralen Stelle verwaltet werden. In GATE gibt es zwei Möglichkeiten Dokumente zu verarbeiten. Entweder werden die Verarbeitungsstationen auf das Dokument oder den Corpus angewendet. Dieser umfasst eine Ansammlung von verschiedenen Dokumenten. Die einzelnen Dokumente brauchen dabei nur mehr dem Corpus zugeordnet werden. Jedes Dokument wird eindeutig durch ein URL identifiziert. Somit können sowohl lokale Daten, als auch Dokumente aus dem Internet bezogen werden. Das Laden von externen Ressourcen wird von der **Document** Klasse übernommen. Zudem lassen sich verschiedene Parameter festlegen: Der Anwender bzw. Entwickler hat die Möglichkeit die Textkodierung festzulegen. Die Dokumentklasse unterstützt die gängigsten Standards. Für die Implementierung wurde die ISO 8859-1[1] gewählt. Außerdem bietet die GATE Dokument Klasse die Funktionalität, im Text vorhandene MarkUps herauszufiltern. Da sich das System auf freien Text konzentriert, werden sämtliche Tags aus dem Dokument entfernt. Zusätzlich kann auch noch der MIME Typ festgelegt werden. Es besteht auch die Möglichkeit das Originalformat des Dokuments zu speichern und zu einem späteren Zeitpunkt wieder zu verwenden. Jene Konfigurationen sind aber für das vorliegende System nicht relevant und werden nicht verändert.

5.7 Adaption der GATE Pipeline

In diesem Abschnitt werden die einzelnen Verarbeitungsmodule der Pipeline behandelt. Dabei handelt es sich um die GATE Basiskomponenten. Diese werden in der Reihenfolge beschrieben, wie sie auch später im System angeordnet werden. Grundsätzlich werden die Implementierungs- und Konfigurationsdetails der einzelnen Komponente erläutert. Zuvor werden die Designentscheidungen besprochen, welche zur vorliegenden Konfiguration des Systems führten.

5.7.1 Designüberlegungen

Das vorliegende System konzentriert sich auf die Annotierung von Bandnamen, Bandmembernamen und Namen von Medien einer Band (also CDs, DVDs, Alben Singles, Compilations, etc.). Diese Problemstellung lässt sich auf jene eines Namenserkennungssystems eingrenzen. Dies betrifft im Wesentlichen den Tokenizer und die lexikalische- und morphologische Analyse. Letztere Komponente kann man in verschiedene Sub-

[1] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-094.pdf

verarbeitungsschritte unterteilen. Vor allem sind aber die lexikalischen Verarbeitungsschritte sehr wichtig für die Namens- bzw. Entitätserkennung. Zusätzlich zu der Tokenization Einheit, wird ein domänenspezifisches Wörterbuch verwendet. Dieses wird in verschiedene Kategorien, wie Genrenamen, Instrumente, usw. eingeteilt. Die Wörterbuchimplementierung von GATE wird als Gazetteer bezeichnet. Jene erlaubt es, das sichere Wissen eines Wörterbuchs in die Extraktionsregeln mit einzubeziehen. Die Erkennung von Bandnamen, usw. bringt eigentlich die schwierigste Problematik mit sich: Sie bestehen nicht nur aus einer Vielzahl von Symbolen, sondern umfassen zum Teil auch komplette Satzkonstrukte. Beispiele für solche Ausprägungen der künstlerischen Freiheit sind zum Beispiel:

- !!! (Chk Chk Chk)
- ...And You Will Know Us By The Trail of Dead
- Me First and the Gimme Gimmes

Weder Tokenizer, noch Wörterbuch können dabei helfen, diese Konstrukte zu erkennen. Um solche stark divergierende Variationen zu erkennen, wird eine Unterteilun der Tokens in die verschiedenen Wortarten (POS Tags) benötigt. Diese Aufgabe wird von einem POS Tagger erledigt. Diesem muss jedoch ein Sentence Splitter vorgeschaltet werden, da teilweise auch Abhängigkeiten zwischen den GATE Komponenten herrschen. Tokenizer, Sentence Splitter und POS Tagger bereiten den vorliegenden Text nur auf die Erkennung der Entität vor. Der Annotierungsprozess hingegen steht aber noch bevor. Da in diesem Fall ein Knowledge Engineering Ansatz verfolgt wird, geschieht dies mit handgeschriebenen Extraktionsregeln. In GATE können solche Regeln in Form eines JAPE Transducer in das System integriert werden. Das System besteht also aus folgenden Komponenten: Tokenizer, Gazetteer Listen, Sentence Splitter, POS Tagger und Transducer. Die Einzelheiten dieser Komponenten wurden bereits in Kapitel 4.7 besprochen. Komponenten, welche im Zuge dieser Arbeit um spezifische Inhalte erweitert wurden, werden in den folgenden Abschnitten ausführlicher diskutiert.

5.7.2 Gazetteer Listen

Für die Implementierung wird eine domänenspezifische Wortliste verwendet. Einige dieser Entitäten werden auch bei der Namenserkennung verwendet und fungieren als Indikatoren für darauf folgende Namen.

Für die vorliegende Arbeit wurde eine spezifische Liste erstellt. Jene enthält musikrelevante Wörter wie Genrebezeichnungen oder Musikinstrumente. Die anderen Lis-

ten beinhalten verschiedene Hinweiswörter, welche eventuell auf interessante Passagen schließen können. Ein Beispiel dafür wäre das Konstrukt "frontman <Vorname><Nachname> ". Generell weisen Wörter wie frontman, guitarist, usw. auf ein Bandmitglied hin. Jene Wörter können unter den allgemeineren Begriff role zusammengefasst werden, wie in [48] vorgeschlagen wird. Diese Liste beinhaltet typische Ausprägungen von Bandrollen:

```
1 bass player
2 bassist
3 cellist
4 drummer
```

Zusätzlich existieren noch Listen für Datumsangaben (Monatsnamen, Jahreszahlen), Länder, Instrumente und Genrebezeichnungen. Dabei sind die einzelnen Listen so konfiguriert, dass sie zwischen groß- oder kleingeschriebenen Einträgen in Texten unterscheiden. Dies liegt vor allem daran, dass die Wissensbasis die extrahierten Entitäten ebenfalls in Gazetteer Listen (**artist.lst**, **member.lst**, **media.lst**) verwaltet und bei den Entitäten zwischen Groß- und Kleinschreibung unterscheiden muss. Andernfalls würde das System bei einigen Spezialfällen falsche Ergebnisse liefern ("One"-Titel der Band Metallica, "one"-eine Nummer). Die vollständigen spezifischen Wortlisten des Systems können dem Anhang entnommen werden.

5.7.3 JAPE Transducer

Um die Entitäten zu erkennen werden die manuell erstellten Regeln in einzelne JAPE Dateien gekapselt. Für die Annotierung der unterschiedlichen Namenstrukturen werden zuerst Makros erstellt. Diese müssen im Anschluss in die Regel integriert werden. Die Implementierung stellt dabei folgende Makros zur Verfügung:

1. **BandName** um Bandnamen bzw. Artists zu erkennen
2. **BandMember** um Bandmitglieder zu taggen
3. **Media** für die Annotierung von sämtlichen Medientiteln

Die exakte Struktur der einzelnen Makros wird im Kapitel 5.3.3 näher beschrieben. Um nun eine konkrete Ausprägung zu erkennen, müssen diese Makros nur mehr in eine Regel eingebettet werden. Eine Regel zur Erkennung eines Schlagzeugers sieht folgendermaßen aus:

```
1 Rule: drummer
```

```
2 (
3   {Token.string == "drummer"}
4   (
5     (BandMember)
6   ):BandMember
7 )--> :BandMember.Member = {kind = "Member", rule = "drummer"}
```

In diesem konkreten Fall wird jedoch nur ein Schlagzeuger erkannt. Da aber eine Band nicht nur aus einem Schlagzeuger besteht, muss diese um verschidene Rollen (frontman, bassist, usw.) erweitert werden. Eine Möglichkeit wäre der **or** Operator (|) von JAPE. So könnte man alle gewünschten Rollen einfach aneinanderreihen. Ab einer gewissen Anzahl an Rollen wird das aber sehr unübersichtlich. Deswegen besteht die Möglichkeit ganze Gazetteer Listen in eine Regel einzubinden:

```
1 Rule: role
2 (
3   (
4     {LookUp.majorType == role}
5   ):BandRole
6   (
7     (BandMember)
8   ):BandMember
9 )-->
10   :BandRole.Role = {rule = "BandRoleTitle"}
11   :BandMember.Member = {kind = "Member", rule = "role"}
```

Um Verknüfungen zu bilden besteht auch die Möglichkeit mehrere Typen zuzuordnen. Im oben angeführten konkreten Beispiel können beide Typen (**role** und **member**) in einem späteren Schritt miteinander verknüpft werden. Allerdings bietet die Möglichkeit mehrere Typen in dieser Form zu annotieren nur sehr wenige Freiheiten. Bei Aufzählungen, welche durch ein beliebiges Symbol getrennt sind, muss akzeptiert werden, dass die Trennzeichen mitannotiert werden. Diese einzelnen Elemente der Aufzählung müssen nun irgendwie getrennt werden. Jedoch bietet GATE die Möglichkeit JAVA Code in den RHS Teil einzubinden. Diese bietet dem Entwickler weit mehr Flexibilität bei komplexeren Strukturen wie Listen oder Tabellen. Dabei können nicht nur Klassen aus dem GATE Framework verwendet werden, sondern auch jene des J2SE Standard API. Eine Aufzählung von einfachen Bandnamen (in diesem Fall nur ein Wort) im Format "bands like Band1, Band2, ..., BandN-1 and BandN", kann folgendermaßen realisiert werden:

```
1 Rule: SimpleBandList
2 (
3   {Token.string == "bands"}
4   {Token.string == "like"}
5   (
```

```
6       {Token.category == NNP}
7       ({Token.string == ","}{Token.category == NNP})[0, 5]
8       ({Token.string == "and"}{Token.category == NNP})
9     ):list
10 )-->
11 {
12   List annList = new ArrayList((AnnotationSet)bindings.get("list"));
13   Collections.sort(annList, new OffsetComparator());
14   for(int i = 0; i < annList.size(); i++){
15           Annotation anAnn = (Annotation)annList.get(i);
16     if(anAnn.getFeatures().get("category").equals("NNP")){
17       FeatureMap features = Factory.newFeatureMap();
18             features.put("rule", "SimpleBandList");
19       annotations.add(
20         anAnn.getStartNode(),
21         anAnn.getEndNode(),
22         "bandname", features
23       );
24     }
25   }
26 }
```

Die Aufzählung wird dabei wie gewohnt mit einer Markierung (**:list**) versehen. Jene kann nun im LHS Teil mit **bindings.get("list")** angesprochen werden und liefert eine Liste von Annotierungen zurück. Die Annotierungen von der Kategorie NNP müssen nur mehr in die Annotierungsliste des Dokuments hinzugefügt werden.

Anhand dieser Beispiele wurden ein Vielzahl an Regeln implementiert und in das System eingebunden. Konkret existieren folgende Dateien in der Regelbasis:

1. **SimpleBandRules.jape** für Bandnamen

2. **SimpleMemberRules.jape** für Bandmitglieder

3. **SimpleMediaRules.jape** für Alben, Singles, usw.

4. **SecureMediaRules.jape** für Medientitel welche in Anführungszeichen gekapselt sind

5. **SimpleListRules.jape** für Aufzählungen von sämtlichen Entitäten

Die Einzelheiten der spezifischen Implementierung der Regeln für das System werden im Kapitel 5.3 detaillierter ausgeführt.

5.8 Wissensbasis

Die Wissensbasis beinhaltet jenes Wissen, das durch die im letzten Kapitel besprochenen Verarbeitungsschritte aus den Dokumenten extrahiert wurde. Dabei werden keine typischen Konzepte wie Inferenz, usw. auf die Daten ausgeführt, um wiederum neues Wissen abzuleiten. Die Wissenbasis hat einzig und alleine die Aufgabe den Extraktionsprozess zu unterstützen. Dabei werden die Regeln so konzipiert, dass ein möglichst hoher Precision Anteil erzielt wird. Der anfangs schlechte Recall Wert soll dabei nach und nach gesteigert werden. Die Wissensbasis besteht dabei aus bereits extrahierte Entitäten, die in spezifischen Gatezetteer Listen (**artist.lst**, **member.lst**, **media.lst**) verwaltet werden. Somit ist es möglich alle Inhalte der Listen im weiterführenden Extraktionsprozess automatisch in einem Dokument zu erkennen ohne die Regelbasis zu verwenden. Die Wissensbasis wird während des Extraktionsprozesses nach jedem Dokument um neue Entitäten erweitert. Im nächsten Kapitel werden die Implementierungsdetails dieser Wissensbasis näher erläutert.

5.8.1 Implementierung im System

Bei der Implementierung wurde bewusst eine offene Schnittstelle gewählt. Die Wissensbasis besteht aus zwei fundamentalen Funktionen: Eine Funktion ist für die Verarbeitung des annotierten Dokuments zuständig, die andere ist für die persistente Sicherung des Wissens in Form von extrahierten Informationen zuständig. Dieses Wissen muss natürlich in irgendeinem Format verwaltet werden. Dies geschieht in Form von spezifischen Gazetteer Listen. Da es sich bei den Informationen lediglich um Entitäten handelt und keine Verknüpfungen zwischen den einzelnen Entitäten hergestellt werden, können die jeweiligen Fakten einfach einer Klasse zugeordnet werden. Diese Klassifizierung kann ganz einfach aus der vorliegenden Annotierung entnommen werden. Das neu gewonnen Wissen wird dabei sofort in den Verarbeitungsprozess integriert. Um dieses Verhalten zu erzielen, durchläuft der Corpus folgenden Zyklus:

1. **Initialphase**: Der Corpus wird sequentiell durchgearbeitet. In dieser Phase wird das nächste nicht annotierte Dokument im Corpus ausgewählt. Wurden alle Texte verarbeitet, endet der gesamte Zyklus und man springt zur Speicherphase.

2. **Verarbeitungsphase**: Das Dokument durchläuft die einzelnen Verarbeitungsschritte (Tokenizer, Gazetteer Liste, Sentence Splitting, POS Tagging, JAPE Transducer). Die gewünschten Informationen werden durch den JAPE Tranducer markiert.

3. **Lernphase**: Die Wissensbasis evaluiert die annotieren Information basierend auf einer beliebigen Strategie (zum Beispiel: alle Entitäten die mindestens in zwei unterschiedlichen Dokumenten gefunden wurden). Wird die jeweilige Information als sicher eingestuft, erfolgt eine Aufnahme in die Wissensbasis. Existieren weitere Dokumente, welche noch nicht verarbeitet wurden, wechselt man zurück in Phase 1. Ansonsten wechselt man in folgende Phase

4. **Speicherphase**: Dieser Schritt beinhaltet die Sicherung der extrahierten Entitäten. Das gespeicherte Wissen kann in einer späteren Extraktionsphase wiederverwendet werden.

Relevant für die Wissensbasis sind eigentlich nur Lern- und Speicherphase. Die Verarbeitungsphase kann man eventuell auch als relevant betrachten, da sie die Entitäten der Wissensbasis in Form der Gazetteer Listen nutzt, um so viele Entitäten wie möglich im Dokument abzudecken. Es gibt also grundsätzliche zwei Subkategorien für jeden Typ (Bandname, Member, usw.). Jene, die durch die Regeln und jene, die durch Gazetteer Listen markiert werden. Die Gazetter Listen von GATE stellen eine einfache Struktur für eine Wissensbasis zur Verfügung. Sobald eine extrahierte Entität als sicheres Wissen angesehen wird, wird sie im jeweiligen Wörterbuch gespeichert. Ab dem nächsten Dokument wird dieses Wort in jedem folgenden Text gefunden. Bevor eine Information als gesichertes Wissen angesehen werden kann, muss es allerdings in irgendeiner Form evaluiert werden. Die handgeschriebenen Regeln funktionieren ausreichend, sind aber nicht perfekt. Die meisten Dokumente enthalten Annotierungen, in denen die Entität zwar gefunden wird, aber auch nachfolgende Token in die Annotierung aufgenommen werden. Dazu kommen noch Strukturen, welche zwar markiert werden, aber nicht relevant sind ("false positves"). Um zu verhindern, dass falsche Informationen aufgenommen werden, wird in der vorliegenden Implementierung eine sehr einfache Regel verwendet: Sobald eine Entität von zwei unterschiedlichen Regeln erkannt wird, wird diese in die Wissensbasis bzw. Gazetteer Liste aufgenommen. Dies ist eine recht rudimentäre Methode, die sich in den Testläufen bewährt hat. Eventuell könnte auch noch die Priorität der Regel für die Evaluierung herangezogen werden. Um die Anzahl an nicht korrekten Entitäten zu verkleinern, wird überprüft, ob eine Entität ausschließlich aus einem spezifischen Hinweiswort besteht. Allerdings hat diese Form der Verwaltung auch nicht nur Vorteile. Einfache Entitäten wie der Song "One" können zu Problemen führen. Solche Wörter werden dementsprechend oft im Text vorhanden sein und führen zu *false positives*. Dies wird durch die Konfiguration **Case Sensitive=false** noch verstärkt. Deshalb wird in der vorliegenden Implementierung zwischen Groß- und Kleinschreibung unterschieden. Die Wissensbasen für Bandname, Bandmitglieder und Medien sind bereits in die Implementierung der Gazetteer Liste der Pipeline eingebunden. Zusätzliche Annotierungsklassen (zum Beispiel: Namen von Plattenfirmen) müssen in die Liste und in die Implementierung der Wissensbasis mit eingebunden werden. Die Schnittstelle er-

möglich aber auch andere Formen von Wissen zu erfassen. Somit wäre es auch möglich mit Hilfe der Wissensbasis alle Annotierungen zu erfassen, welche nur von dem Wörterbuch und nicht von den Extraktionsregeln markiert wurden. Mit den **pretokens** und den **posttokens** dieser Annotierungen, können nun wieder neue Regeln generiert werden. Zusätzlich kann statistisch erfasst werden, welche Regeln am häufigsten in einem Corpus vorkommen.

Kapitel 6

Erweiterungen

Die Extraktion von einfachen Entitäten alleine liefert nur wenig interessante Ergebnisse. Man kann lediglich Aussagen darüber treffen, dass es sich bei der konkreten Entität E1 um eine Band handelt. Um komplexere bzw. interessantere Informationen zu extrahieren, muss die Anwendung um verschiedene Komponenten erweitert werden. Die allgemeine Strukturierung der Architektur erlaubt es dem Entwickler jene Komponenten einfach in die Anwendung ein- oder auszukoppeln. Die jeweilige Komponente wird durch die **KnowledgeBase** Schnittstelle in das System integriert. Jene ermöglicht eine weitere Verarbeitung des Dokuments und den darin enthaltenen Annotierungen. So kann man zum Beispiel die Ergebnisse des Extraktionsprozesses verwenden, um eine vorliegende Seite zu klassifizieren. Die aber wohl interessanteste Implementierung, wäre ein Modul zur Extraktion bzw. Bildung von Relationen. Die einzelnen entwickelten Erweiterungen können auch gemeinsam in die Anwendung integriert werden, falls die Resultate einer Erweiterung weiterverarbeitet werden müssen. Im Zuge dieser Diplomarbeit wurden einige dieser Erweiterungen realisiert. Diese werden in den folgenden Abschnitten detaillierter besprochen.

6.1 Automatische Regelextraktion

Ein wichtiger Punkt ist die Erweiterung der Regelbasis. Bei einem Knowledge Engineering Ansatz hängt die Qualität ganz von den Fähigkeiten des Menschen ab. Welche Regeln schließlich verwendet werden, beruht daher auf subjektive Entscheidungen. Bei der Verarbeitung von so vielen Dokumenten, kann aber die Fähigkeit Regeln zu finden durch diese monotone Aufgabe beeinträchtigt werden. Automatisierte Verfahren unterliegen nicht diesem Nachteil. Jede gefundene Regel wird anhand von bestimmten Metriken evaluiert. Allerdings benötigen solche lernenden Verfahren einen vorannotierten Corpus, um die relevanten Passagen zu markieren. Eine Alternative wird in [15, 14]

vorgeschlagen. Dabei werden spezifische Queries an Google [1] abgesetzt und die resultierenden Seiten auf bestimmte Muster untersucht, um neues Wissen zu extrahieren. Um neue Regeln zu finden wird dieser Ansatz mit [27] kombiniert und verwendet eine ähnliche Struktur wie in [16] beschrieben. Das System wird dabei in zwei Phasen unterteilt: Einer Retrievalphase und eine Extraktionsphase. Die Retrievalphase benötigt ein Modul, dass relevante Seiten zur Verfügung stellt. Idealerweise werden diese Seiten aus dem Internet bezogen, da diese in den meisten Fällen auch aktuell sind. Um relevante Seiten zu erhalten, wird ein spezifischer Request an eine Suchmaschine gesendet. Für diese Aufgabe wird das CoMIRVA [47] Framework verwendet. Dieses stellt die Klasse **AnySearch** zur Verfügung, welche ein beliebiges Query an Google sendet. Anhand einer spezifischen Artist Gazetteer Liste *artist* ist es möglich folgende Query zu bilden:

$artist_i$+ "band biography"

Nachdem alle Anfragen an Google gestellt wurden, verfügt man über eine Liste mit Suchergebnissen in Form von URLs. Mittels GATE können die URLs im Anschluss in den Corpus der Basisarchitektur geladen werden und stehen zur Weiterverarbeitung in der Extraktionsphase bereit. Für die Extraktionsphase kann die vorgestellte Basisarchitektur verwendet werden. Allerdings werden dabei keine Extraktionsregeln bzw. kein JAPE Transducer benötigt. Für diesen Zweck werden relevante bzw. spezifischen Indormationen (zum Beispiel Bandnamen) benötigt. Spezifische Gazetteer Listen verwalten diese Informationen bzw. Entitäten. Der Anwender erstellt dabei ein beliebiges Set an relevanten Entitäten (Artists, Members, Media, usw.). Wird eine Entität von den Gazetteer Listen erkannt, wird eine festgelegte Anzahl von umliegenden Token verarbeitet. Diese ergibt folgende Struktur:

<pretokens> <tagged entity> <posttokens>

Wie viele umliegende Pre- oder Posttokens tatsächlich verarbeitet werden, kann vom Anwender beliebig konfiguriert werden. Somit verfügt man über eine begrenzte Liste von Vorgänger- (*pre*) und Nachfolgertoken (*post*) der gefundenen Entität. Da einer Regel sowieso immer mindestens ein Token vorangehen bzw. folgen muss ist es möglich mit diesen Listen verschiedene Variationen von Regeln zu erezugen. Für den konkreten Fall, dass nur der Vorgänger und Nachfolger der Entität verarbeitet wird, können folgende Regeln erstellt werden:

1. <preToken> <tagged entity>

2. <preToken> <tagged entity> <postToken>

[1] http://www.google.com/

3. <tagged entity> <postToken>

So könnte die generierte Regel <Artist> <postToken> eventuell die konkrete Regel <Band> ”released” abdecken. Die Komponente für die automatische Regelextraktion verwaltet sowohl die konkrete Regel, als auch die Häufigkeit ihres Auftretens in den Dokumenten. Der komplette Prozess läuft dabei folgendermaßen ab:

Konfiguration der Menge an Vorgänger $preSize$
Konfiguration der Menge an Vorgänger $sucSize$
Laden der Bandnamenliste
while weitere Bandnamen vorhanden **do**
  Query bilden: $artist_i$ + ”band biography”
  Request an Google
  Laden der resultierenden URLs
  while nicht annotiertes Dokument vorhanden **do**
    Annotierung durch Gazetteer Listen
    while weitere Annotierungen vorhanden **do**
      i = Index der Annotierung im Tokengraphen $tokens$
      $pre = \{tokens_{i-1}, \ldots, tokens_{i-preSize}\}$
      $post = \{tokens_{i+1}, \ldots, tokens_{i+sucSize}\}$
      pre Variationen: $\{pre_0, pre_0 - pre_1, \ldots, pre_0 - pre_{preSize-1}\}$ + <entity>
      suc Variationen: <entity> + $\{suc_0, suc_0 - suc_1, \ldots, suc_0 - suc_{sucSize-1}\}$
      pre und suc Variation: pre_0 + <entity> + suc_0
      Verwaltung der generierten Regeln
    end while
  end while
end while
Weiterverarbeitung der verwalteten Regeln

Aus den gespeicherten Resultaten kann man nun die vielversprechendsten Exemplare auswählen. In diesem konkreten Fall werden jene N Regeln ausgewählt, welche am häufigsten im Corpus gefunden wurden. Jene könnte man durch ein Zusatzmodul gleich in Regeln mit JAPE Syntax umwandeln.

6.2 Extraktion und Verwaltung von Relationen

Das Erkennen und Verarbeiten von Beziehungen in unstrukturierten Dokumenten ermöglicht dem Entwickler die Bildung von Beziehungsnetzwerken, usw. Allerdings ist die Verknüpfung von unterschiedlichen Entitäten noch um einiges diffiziler als eine ein-

fache Annotierung von Namen. In [34, 46] wurden verschiedene Ansätze besprochen um konkrete Entitäten miteinander zu verknüpfen. Die Standardansätze konzentrieren sich auf Trigramme der Form (NNP, Verb, NNP). Jene Form kann für diese Domäne adaptiert werden. So kann man zum Beispiel das Triple (Member, Verb, Band) bilden, um konkrete Muster wie <Member> left <Band> zu erkennen. Allerdings kommen diese Strukturen eher selten in Sätzen in den einzelnen Dokumenten vor, deswegen muss ein gewaltiges Set an Dokumenten verarbeitet werden, um eine ausreichende Menge von Beziehungen aufzubauen. Außerdem trifft dies nur immer auf jeweils zwei konkrete Entitäten zu. Bei Aufzählungen wie "<Band> consists of <Member1>, <Member2>, ... and <MemberN> " muss die Bildung der Relation (Band, Member) auf die Struktur dieser Aufzählung zugeschnitten werden. Vor allem in Bandbiographien wird der konkrete Künstler oft nur als "The Band" (nicht zu verwechseln mit der Band "The Band") bezeichnet. Ähnliches gilt auch für Medien. Zudem müssen die einfachen Entitäten im Vorhinein annotiert werden. Diese Aufgabe übernimmt aber sowieso die entwickelte Basiskomponente. Vor allem wegen der oben angeführten Probleme, wird hier ein anderer Ansatz verfolgt. In einem ersten Schritt wird zuerst das Dokument anhand der erkannten Bandnamen analysiert. Lässt sich ein Dokument D einer konkreten Band X zuordnen, so wird das vorliegende Dokument als relevant für diese Band X eingestuft. Ist dies der Fall, so werden automatisch alle anderen Entitäten $\{Y\}$ im Dokument dieser Band $X \leftrightarrow \{Y\}$ zugewiesen. Somit trifft man die Annahme, dass eine gewisse Beziehung zwischen der Band und einer anderen Entität besteht. In diesem konkreten Fall werden **Artist-Member** und **Artist-Media** Relationen gebildet. Je häufiger diese Beziehung gefunden auftritt, desto sicherer ist es, dass diese konkrete Relation auch tatsächlich korrekt ist. Für diesen Zweck werden zwei Komponenten benötigt: Ein Modul, welches ein Dokument analysiert bzw. nach Bandnamen klassifiziert und ein Relationsgraph, welcher die Relationen verwaltet. In dem folgenden Kapitel werden die Designüberlegungen für die vorliegende Implementierung näher beleuchtet.

6.2.1 Klassifizierung von Seiten

Bevor eine Entität einer anderen zugeordnet werden kann, muss das vorliegende Dokument verarbeitet und analysiert werden. Dazu müssen die einzelnen Entitäten im Dokument annotiert worden sein. Somit muss zuerst die Basisextraktionskomponente diesem Modul vorgeschaltet werden. Als Implementierung der Schnittstelle KnowledgeBase kann nun diese Komponente in das System eingebunden werden.

Da sich aber musikrelevante Artikel bzw. Seiten zumeist auf eine einzelne Band konzentrieren, liegt eine Analyse mittels vorhandenen Bandnamen sehr nahe. Darunter fallen unter anderem Konzertberichte, Rezensionen von Alben und nicht zuletzt Bandbiographien. Allerdings wird zuerst eine angemessene Strategie benötigt, um ein Dokument

D hinsichtlich der extrahierten Menge von Bandnamen *annotatedbands* zu klassifizieren. Kann man ein Dokument D eindeutig einer konkreten Band X zuordnen, werden alle anderen Entitäten (in diesem Fall **Member** und **Media**) in diesem Dokument der Band X zugewiesen. Im Zuge dieser Arbeit wurden zwei unterschiedlich Strategien für diesen Zweck erarbeitet:

1. Das Dokument wird der ersten vorkommenden Band zugeordnet

2. Das Dokument wird dem am häufigsten vorkommenden Bandnamen zugeordnet

Die Ergebnisse der Evaluierungen beider Ansätze befinden sich in Kapitel 7.3.2. Anhand der daraus gewonnenen Erkenntnise wurde eine neue Strategie konzipiert. Bei den oben angeführten Strategien handelt es sich um ein boolsches Klassifizierungsmodell: Entweder kann man die vorliegenden Seite eindeuig einer Band zuordnen oder nicht. Ein Modell welches auf einer relativen Häufigkeit beruht, wäre in diesem Fall besser geeignet. Deshalb wurden diese rudimentären Strategien kombiniert und auf die weiteren Bedürfnisse angepasst. Das resultierende Modell funktioniert folgendermaßen: Zuerst wird mit der oben angeführten zweiten Stratgie (häufigster Bandname im Dokument) eine relative Häufigkeit p berechnet:

$$p = \frac{count(mostoccuringband)}{count(annotatedbands)} \tag{6.1}$$

Da vor allem bei bandspezifischen Dokumenten der Künstlername am Anfang des Textes genannt wird, wird überprüft, ob die erste annotierte Band mit der häufigsten annotierten Band übereinstimmt. Ist dies der Fall, so wird die relattive Häufigkeit p erhöht:

$$p = p + \frac{1 - p}{3} \tag{6.2}$$

Die oben angeführte Formel wurde anhand von zahlreichen Testläufen während den ersten Evaluierungsversuchen erarbeitet. Erreicht dieser Wert p nun einen gewissen Schwellenwert t, so gilt das vorliegende Dokument als relevant für die jeweilige Band X (=most occuring band). Dieser werden im Anschluss alle anderen Entitäten zugeordnet. Empirisch wurde ein Wert von $t = 0.6$ für adäquat befunden.

6.2.2 Relationsgraph

Da nicht nur einzelne Dokumente, sondern ganze Korpora verarbeitet werden, besteht der Bedarf an einer geeigneten Datenstruktur um das bereits vorhandene Wissen bzw. Entitäten und deren Relationen persistent zu halten. Die Grundproblematik bei der automatischen Extraktion von Informationen liegt in der Unsicherheit der jeweiligen Entität. Existieren aber Entitäten bzw. deren Relationen sehr häufig und eventuell auch noch in unterschiedlichen Dokumenten, besteht ein geringeres Risiko, dass die vorliegende Information falsch ist.

Für diesen Zweck wird ein Beziehungsnetzwerk verwendet. Diesem liegt die Datenstruktur eines bidirektionalen Graphen [30] zu Grunde. Diese besteht grundsätzlich aus zwei Komponenten: Knoten und deren Relationen.

Dabei wird eine Entität als Knoten angesehen und speichert die konkrete Ausprägung der jeweiligen Entitätsklasse. Da aber gleiche Entitäten in unterschiedlichen Dokumenten öfters anders geschrieben werden, kann der Fall eintreten, dass man für dieselbe Entität mehrere unterschiedliche Stringausprägungen mitführt. In [48] wird von ähnlichen Problemen berichtet. Um diese Fälle zu reduzieren, wird der String der Entität standardisiert. Dafür schaltet man eine einfache Methode vor, welche verschiedene Character einer Entität auf eine Stammform reduziert. So werden alle Großbuchstaben auf Kleinbuchstaben gemappt, sowie à, á, â, usw. einfach auf 'a' reduziert.

Jeder Knoten verwaltet seine Relationen zu anderen Knoten, sowie den zugewiesenen Annotierungstyp. So kann zum Beispiel Dokument D_X behaupten, dass die Entität E eine Band ist. Dokument D_Y behauptet aber E ist ein Medium. Somit wird das Problem von gleichnamigen Interpreten und Alben beseitigt (zum Beispiel: Rage Against The Machine - Rage Against The Machine).

Eine Relation zwischen zwei Entitäten ist dabei immer bidirektional. Für jede Relation wird außerdem die absolute Häufigkeit der Relation n gespeichert. Die absolute Häufigkeit n gibt an, wie oft die konkrete Relation im laufenden Prozess gefunden wurde.

Um den erarbeiteten Relationsgraphen nicht jedes Mal verwerfen zu müssen, hat der Anwender die Möglichkeit die Ergebnisse am Ende bzw. wäherend einer Sitzung zu speichern. Für diese Funktionalität wurde die Serializable Schnittselle von JAVA [8] implementiert. Die gespeicherten Ergebnisse können nun wiederum für weitere Sitzungen verwendet, aber auch für die Evaluierung dieses Moduls herangezogen werden.

Kapitel 7

Evaluierung

Ein wesentlicher Teil der Arbeit konzentriert sich auf die Evaluierung des Systems. Eine Evaluierung besteht aus zwei Komponenten: Die tatsächlichen Ergebnisse des Systems und einer Referenz. Bei dieser Referenz kann es sich um ein anderes System handeln, welches als Baseline dient. Um Referenz und vorliegendes System miteinander vergleichen zu können, wird eine Ground Truth benötigt. Dabei handelt es sich um eine (angenommene) fixe "Wahrheit", welche für die Evaluierung herangezogen wird. Die Ergebnisse und die Referenz werden anhand von standardisierten Maßzahlen miteinander verglichen.

7.1 Vorbereitungen

Bevor der Evaluierungsprozess gestartet werden kann, muss man eine Ground Truth erstellen bzw. jene von bestehenden Evaluierungen benutzen. Dabei kann es sich um einen manuell annotierten Corpus handeln, welcher als Referenz für die Ergebnisse des Systems herangezogen wird. Jener wird mit speziellen Markierungen versehen, welche alle notwendigen Ausprägungen kennzeichnen. In der Regel umfasst die Ground Truth die Menge aller relevanten Informationen in einem Dokument bzw. Corpus. Um ein Dokument manuell zu annotieren kann die Entwicklungsumgebung von GATE herangezogen werden [26]. Diese bietet dem Anwender die Möglichkeit Texte mit beliebigen Annotierungen zu versehen.

Allerdings wird für diesen Zweck ein selbstgeschriebenes Tool verwendet. Das hat vor allem folgende Gründe: Subjektiv betrachtet ist die GUI von GATE ziemlich sperrig. Um eine neue Annotierung hinzuzufügen, muss die Entität mit dem Cursor markiert werden. Im Anschluss öffnet sich ein Fenster, wo der Name der Annotierung festgelegt werden kann. Wird eine andere Annotierung benötigt, muss diese in einer Dropdown

Liste ausgewählt werden. Da aber der Prozess einer manuell durchgeführten Annotierung sehr langwierig und vor allem mühsam ist, soll dieser Vorgang mit so wenig Aufwand wie möglich verbunden sein. Zudem ist die GUI von GATE auch sehr überladen. Ein weiterer Grund, der gegen die GATE Utilities spricht, ist die Durchführung der Evaluierung auf verschiedenen Ebenen. Das Annotation Diff Tool von GATE bietet nur die Standardebene an. In dieser Arbeit werden aber zwei unterschiedliche Level an Maßzahlen berechnet. Bevor man die Evaluierung startet, muss man sich noch Gedanken machen, in welcher Form die einzelnen Entitäten markiert werden. In den zahlreichen MUC Evaluierungen [19] wurden einfache MarkUp Elemente verwendet. Diese haben folgende Struktur:

<ENAMEX TYPE=PERSON >Brian May</ENAMEX>
<NUMEX TYPE=MONEY >1000 EUR</NUMEX>

Für diese Evaluierung werden ähnliche Tags verwendet. Da sich die Entitäten lediglich auf Bandnamen, Mitglieder und Medien einer Band beschränken, werden momentan keine Attribute verwendet. Grundsätzlich werden folgende Tags verwendet:

- **Artist**: Anstatt von Bandname wird die etwas allgemeinere Formulierung Artist verwendet. Dies liegt vor allem daran, dass ein Solokünstler nicht als Band eingestuft werden kann, aber auch kein reines Bandmitglied ist. Dieser Tag umfasst also sowohl Solokünstler als auch komplette Bands.

- **Member**: Markiert alle Mitglieder einer Band. Dabei werden aber nur vollständige Namen annotiert. Vornamen oder Nachnamen alleine werden nicht als vollständige Entität betrachtet.

- **Media**: Umfasst alle physischen Medien in der Musikbranche: Album, Single, LP, EP, Compilation, DVD und Musikvideo

Wie bereits erwähnt, wird ein eigenes Programm für die Aufgabe der manuellen Annotierung verwendet. Grundsätzlich sollte dieser Vorgang immer durch ein Tool unterstützt werden. Eine Annotierung mittels eines herkömlichen Texteditors ist keines Falls zu empfehlen, da vor allem bei sehr langwierigen Annotierungen die Konzentration sehr schnell nachlassen kann. Dies resultiert generell in fehlerhafte Annotierungen und man benötigt sehr viel Zeit dafür diese Fehler zu korrigieren. Allgemein werden nur zwei fundamentale Anforderungen an das Annotierungstool gestellt: Zum einem soll es möglich sein, in einfacher Weise Annotierungsklassen hinzuzufügen bzw. zu entfernen. Zum anderen muss der Vorgang der Annotierung mit so wenigen Schritten wie möglich durchführbar sein. Im Zuge dieser Arbeit wurde ein einfaches Tool geschrieben, welche den Anwender bzw. Entwickler bei dieser langwierigen Arbeit unterstützt.

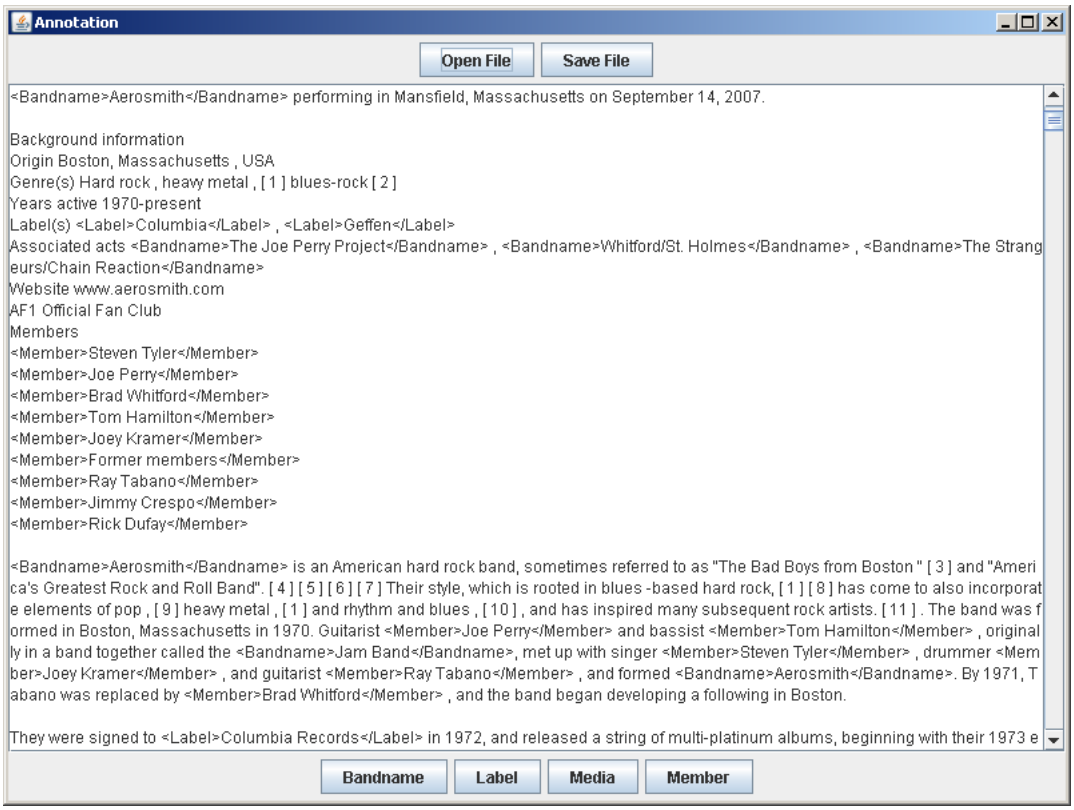

Abbildung 7.1: Grafische Oberfläche des Annotierungsprogramm

Generell unterteilt man das Programm in drei verschiedene Einheiten: Die oberste beinhaltet zwei Buttons um ein Dokument zu öffnen und ein Dokument zu speichern. Wird ein modifiziertes Dokument gespeichert wird dies durch ein zusätzliches Fenster bestätigt. Die manuellen Annotierungen werden im geöffneten Dokument verwaltet. Daher empfiehlt es sich zwei Versionen eines Dokuments zu verwenden. Eine davon bleibt unverändert, die andere wird mit dem Tool annotiert. In der unteren Schicht befinden sich die verschiedenen Annotierungsklassen. Für jede Annotierungsklasse existiert ein Button. Im Mittelteil wird das Dokument in einer JTextArea angezeigt. Um eine Entität zu annotieren, muss diese mit dem Cursor ausgewählt werden. Im Anschluss muss nur mehr den Button mit der gewünschten Annotierungsklasse angeklickt werden. Die ausgewählte Passage wird sofort mit den jeweiligen Tags gekapselt. Hat man das Ende des Dokuments erreicht, können die Änderungen in der Ausgangsdatei gespeichert werden. Dieser Vorgang wird durch eine Meldung bestätigt. Die Annotierungsanwendung wurde mit den gängigen JAVA Swing Komponenten erstellt. Die verschiedenen Annotierungsklassen werden durch eine einfache textuelle Konfigurationsdatei im Hintergrund in die Anwendung geladen. Für jede Klasse existiert ein eigener Zeileneintrag in dieser Datei. Diese werden im Programm alphabetisch sortiert dargestellt. Wurde die gewünschte Datei mit den jeweiligen Metainformationen versehen, so kann der Anwender seine Änderungen speichern. Diese modifizierte Datei kann auch später für die

Evaluierung verwendet werden. Die verfügbaren Annotierungsklassen werden einfach in einen spezifischen Parser geladen. Jener durchläuft die manuell annotierten Dokumente und extrahiert sämtliche relevante Informationen. Diese werden strikt nach Klasse getrennt. Die Menge an relevanten Informationen kann nun mit den tatsächlichen Resultaten des Extraktionsprozesses verglichen und die entsprechenden Maßzahlen errechnet werden. Dies gilt allerdings nur für die Evaluierung des Basissystems. In den Erweiterungen wurde aus den extrahierten Informationen ein semantisches Netz erstellt, welches Artists mit Bandmitgliedern und Medien verknüpft. Um die Qualität dieser Komponente zu evaluieren benötigt man eine andere Herangehensweise. Im Grunde genommen könnte man eine eine zweite Ausprägung des semantischen Netzwerks manuell erstellen. Jenes beinhaltet die Gesamtmenge aller richtigen Relationen zwischen den einzelnen Entitäten. Allerdings müsste jemand dieses Netz manuell mit Informationen füllen. Zur Evaluierung des zugehörigen LineUps einer Band wird der Testcorpus aus [48] verwendet. Dieser beinhaltet ein Set von 51 Bands mit jeweils 100 spezifischen Seiten. Die gewonnen Informationen werden im Anschluss des Extraktionsprozess mit der beiliegenden Ground Truth evaluiert. Für die verbleibenden Entitäten wurden zwei Hilfsklassen erstellt, welche die erforderliche Daten über einen WebService beziehen. Sehr viele Musikplattformen verwalten einzelne Seiten mit allen relevanten Informationen, welche man für die Evaluierung benötigt. Manche davon bieten zusätzlich noch WebServices an, um diese Daten in einem proprietären Format zu beziehen. Für die Seiten last.fm (ein Internet Radio, welches zusätzlich auch benutzergenerierte Biographien bzw. Diskographien auf seiner Webpräsenz anbietet) und MusicBrainz (eine ausführliche Musikdatenbank) wurden zwei Module realisiert, welche die relevanten Verknüpfungen beziehen. Diese Module konzentrieren sich auf zwei Aufgaben:

1. Überprüfung, ob der extrahierte Artistname auch tatsächlich existiert.

2. Bereitstellen der Medientitel der jeweiligen Band

Somit kann man auch die Medienrelation evaluieren, sowie die Trefferquote aller extrahierten Bandnamen errechnen.

7.2 Vorgang der Evaluierung

Nachdem man nun über eine Ground Truth verfügt, können nun die einzelnen Maßzahlen für die Evaluierung berechnet werden. Für diese Aufgabe werden die üblichen Metriken verwendet. Diese stammen ursprünglich aus dem Bereich des Information Retrieval [44] und wurden für diesen Zweck adaptiert. Um diese beiden Werte zu kombinieren, wurde ein zusätzliches Maß eingeführt. Das F-Measure in seiner Standardform

($\beta = 1$) repräsentiert das harmonische Mittel zwischen Precision und Recall. Die exakte Definition dieser drei Metriken findet man in den Formeln 3.1, 3.2 und 3.3.

Für die vorliegende Evaluierung wurden diese drei Maßzahlen verwendet. Für den F-Measure Werte wurden Precision und Recall gleichgewichtet. Somit brauchen nur mehr die Annotierungen des entwickelten Systems mit denen der vorliegenden Ground Truth verglichen werden. Für die Evaluierung von einzelnen Dokumenten werden diese Maßzahlen auf zwei unterschiedlichen Ebenen berechnet:

1. **Level 1** überprüft ob eine gefunden Entität tatsächlich existiert. Wie oft diese im Dokument vorhanden ist, spielt hier keine Rolle

2. **Level 2** bezieht die Anzahl einer Entität im Dokument in die Berechnung mit ein.

Für die Evaluierung der Relationskomponente kann jedoch nur Level 1 verwendet werden, da keine vollständig vorannotierte Ground Truth vorliegt, welche die Anazhl der relevanten Ausprägungen einer Entität im Corpus verwaltet. Sie gibt lediglich darüber Auskunft, ob die vorliegende Entität relevant ist oder nicht.

7.2.1 Berechnung der Maßzahlen

Im Zuge der Evaluierung wurde Precision (Formel 3.1) und Recall (Formel 3.2) für **Artist-Member** und **Artist-Media** Relationen berechnet. Die erforderlichen Daten bzw. alle relevanten Relationen werden dabei in einer einfachen Textdatei verwaltet. Es existiert sowohl eine Textdatei für **Artist-Member**, als auch eine Textdatei für **Artist-Media** Relationen. Aus den Textdateien wird zuerst eine Liste von relevanten Artists erstellt. Jeder konkrete Artist in dieser Liste verfügt über eine Referenz zu einer Menge an relevanten Relationen. Für den konkreten Fall "Metallica" werden die aktuellen Mitgliedernamen folgendermaßen verwaltet:

$$Metallica \rightarrow \{JamesHetfield, LarsUlrich, KirkHammett, RobertTrujillo\}$$

Für jeden einzelnen Artist in dieser Liste werden Precision und Recall für die vorhandenen Relationen berechnet. Die Ergebnisse jeder Band werden aufsummiert und durch die Anzahl der vorhandenen Bandnamen in der Liste dividiert. Der exakte Vorgang der Evaluierung läuft folgendermaßen ab:

Einlesen der relevanten Daten rel
$relArt =$ Auslesen aller Artists aus rel

Einlesen der extrahierten Daten
while weitere Artists in $relArt$ vorhanden **do**
  $relSet$ = Auslesen aller relevanten Relationen für $artist_i$
  $extSet$ = Auslesen aller extrahierten Relationen für $artist_i$
  if | extSet | > 0 **then**
    $totalPre+ = |extSet \cap relSet|/|extSet|$
  else
    $totalPre+ = 1$
  end if
  if | relSet | > 0 **then**
    $totalRec+ = |extSet \cap relSet|/|relSet|$
  else
    $totalRec+ = 0$
  end if
end while
$totalPre/ = |relArt|$
$totalRec/ = |relArt|$
$totalF = (2 \cdot totalPre \cdot totalRec)/(totalPre + totalRec)$

Dieser Algorithmus wird sowohl für die Evaluierung von **Artist-Member**, als auch für **Artist-Media** Relationen verwendet.

7.3 Ergebnisse

Die Evaluierung des Systems wurde anhand von verschiedenen Aspekten durchgeführt. Der Erfolg des extrahierten Beziehungsnetzwerks beruht dabei auf folgenden wesentlichen Punkten:

1. Die Qualität der extrahierten Entitäten
2. Der Grad an Korrektheit bei der Artist Klassifikation
3. Precision und Recall bei den Relationen Artist - Member und Artist - Media
4. Vergleich anhand einer geeigneten Baseline

Daher konzentriert sich die Evaluierung auf diese vier grundlegenden Punkte. Mit Hilfe der vorgestellten Evaluierungsmechanismen werden die jeweiligen Maßzahlen berechnet und anschließend interpretiert. Als Testcorpus wurde jener aus [48] verwendet. Dieser

besteht aus einem Set an 51 Bands vorwiegend aus dem Metal Genre. Die extrahierten Informationen wurden ausschließlich aus diesem Corpus bezogen.

7.3.1 Korrektheit der Bandnamen

Stellvertretend für die Korrektheit der Entitäten werden alle extrahierten Bandnamen bzw. Interpreten in der Wissensbasis evaluiert. Dies betrifft jetzt zwar nur $\frac{1}{3}$ der extrahierten Entitäten, allerdings kann man mit den Evaluierungen der Relationen Artist - Member und Artist - Media in den folgenden Kapiteln Rückschlüsse auf deren Qualität ziehen. Da die Entitäten ohnehin in einer einfachen Textdatei verwaltet werden, muss man jene nur mehr traversieren und deren Korrektheit überprüfen. Für diesen Zweck wurde die Methode **isArtist** des MusicBrainzRetriever verwendet. Wird ein Artist als korrekt klassifiziert, so wird die Anzahl der korrekten Interpreten um eins inkrementiert. Der Grad an Korrektheit ergibt sich durch das Verhältnis der korrekten Artists *correct* zur Gesamtheit aller Artists *total* in der Wissensbasis. Um Probleme mit unterschiedlichen Schreibweisen zu minimieren wurde die Strings auf eine Stammform (ä → a, ö → o, usw.) normalisiert.

Im Zuge dieser Evaluierung wurden 831 Interpreten extrahiert. Davon wurden 675 Entitäten als korrekt eingestuft. Das ergibt ein Ergebnis von 81.23 Prozent.

Allerdings befinden sich in der Wissensbasis ab und an unterschiedliche Ausprägungen (Anthrax - ANTHRAX, usw), welche die Qualität tendenziell überschätzen. Allerdings befinden sich auch sehr viele Bandnamen in der Liste, welchen ein ”The” voran geht (The Offspring, The Scorpions), aber häufig ohne dieses ”The” gelistet werden. Somit dürfte sich das Ergebnis hier wieder relativieren.

Bei subjektiver Betrachtung der extrahierten Entitäten fällt folgendes auf: Die Anzahl von nicht relevanten Entitäten ist sehr gering. Die meisten extrahierten Entitäten haben einen Musikbezug und wurden auch fehlerfrei extrahiert, jedoch handelt es sich in einzelnen Fällen um einen anderen Annotierungstyp. So wird zum Beispiel ”Bad Religion” Mitglied ”Brett Gurewitz” als Artist gelistet. Dies liegt an sehr vagen Regeln, welche viel Handlungsspielraum lassen. Ein Besipiel wäre folgendes Muster:

<Artist> recorded

In seltenen Fällen steht hier ein Mitglied einer Band statt des Bandnamens.Manchmal findet sich auch ein Medientitel in dieser Liste, wobei es hier schwer fällt zu unterscheiden, da sehr häufig ein Debut Album den Titel der Band trägt. Leider hat das GATE

Framework einige Schwächen mit Umlauten. So wird die Band ”Motörhead ” auch als ”MotÃ¶rhead” geführt.

7.3.2 Klassifikation von Webseiten

In Kapitel 6.2.1 wurden zwei einfache Strategien vorgestellt. Im Zuge dieser Evaluierung wurden die Ergebnisse beider Ansätze ausgewertet.

Beide Strategien wurden mit relevanten Bandbiographien ausgewertet. Dabei erreichte die erste Metrik eine Erfolgsrate von ca. 70 Prozent. Die Erfolgsquoten bei der zweiten Strategie lag knapp über 75 %.Eine Erfolgsrate von 75 % klingt bei erster Betrachtung nicht schlecht, aber diese bedeutet, dass $\frac{1}{4}$ des Corpus nicht richtig klassifiziert wurde.

Vor allem klassifizierte der erste Ansatz jene Seiten richtig, bei welchen der zweite Ansatz versagte. Dies war auch umgekehrt der Fall. Deswegen ist es naheliegend die beiden Strategien miteindander zu kombinieren. Vor allem besteht ein sehr hoher Unsicherheitsfaktor, wenn die Seite nur anhand des ersten auftretenden Namens klassifiziert wird.

7.3.3 Line Up Evaluierung

Für die Evaluierung der Relation **Artist - Member** wird die Ground Truth aus [48] verwendet. Dabei existieren zwei Versionen der Ground Truth: Eine beinhaltet sowohl aktuelle, als auch ehemalige Mitglieder, die Andere besteht nur aus dem aktuellen Line Up. Für die Auswertung der Ergebnisse wurden die aktuellen, sowie die ehemaligen Mitglieder verwendet. Allerdings lassen sich die Werte aus [48] nur bedingt vergleichen, da sich diese auf **member-instrument** Relationen konzentrieren.

Bei den vorliegenden Ergebnissen handelt es sich um Totalwerte. Diese bilden den Mittelwert aus den Resultaten von Precision und Recall für jede einzelne Band. Die aufsummierten Werte werden durch die Anzahl der Interpreten im Testcorpus dividiert. Diese Form der Evaluierung wurde sowohl für die LineUp, als auch die Medien Beziehungen angewendet.

Um die Ergebnisse zu relativieren, wurde zusätzlich der höchstmögliche Recall Wert des vorligenden Testcorpus ermittelt. Dieser liegt bei 79.52 % für sowohl aktuelle, als auch ehemalige Mitglieder. Ein kleiner Anteil wird in der Ground Truth etwas anders gelistet wird als im Corpus, womit der tatsächliche Anteil noch etwas nach oben korrigiert werden muss.

Da im Graphen nicht nur eine konkrete Relation gespeichert wird, sondern auch deren Häufigkeit *count*, ist es möglich einen Verlauf zwischen Precision und Recall, wie in Abbildung 7.2 dargestellt, zu visualisieren. Dabei werden Precision und Recall abhängig von der auftretenden Häufigkeit ($count > 1, 2, \ldots, 12, 15, 20, 25, 30$) einer Relation berechnet.

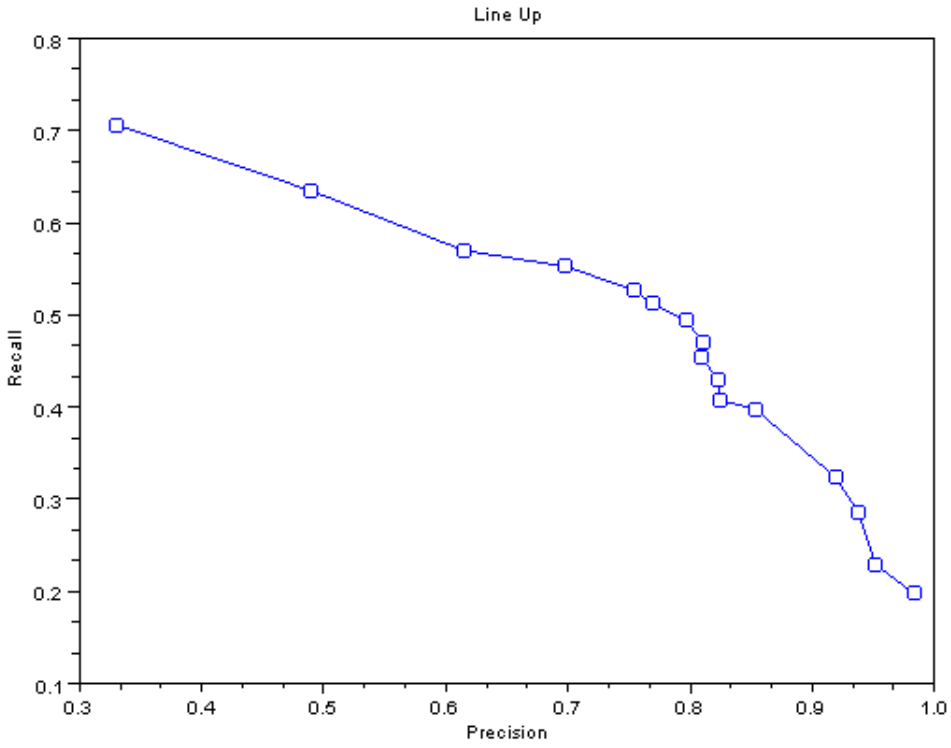

Abbildung 7.2: Precision-Recall Plot für das ehemalige und aktuelle Line Up (Stand 2007) für verschiedene Ausprägungen von count

Das beste Ergebniss wird mit einem $count > 4$ erreicht. Dabei wird ein F-Measure Wert von 62.04 % erzielt. Bei einem $count > 0$ kommt der Recall Wert nahe an die zuvor errechnete obere Grenze des Corpus heran. Allerdings wurde bei der Berechnung der oberen Grenze der Faktor der Verknüpfung zwischen Artist und Member nicht miteinbezogen. Bei einer Evaluierung des aktuellen Lineups ist der Recall Wert noch etwas höher, wie man in Abbuildung 7.3 erkennen kann. Auffallend ist, dass der Precision Wert bei dieser Evaluierung ein wenig geringer ist.

Zusätzlich werden Bandmitglieder im Graphen mit Spitznamen geführt, welche aber in der Ground Truth unter ihrem vollständigen Namen gelsitet sind. Zusätzlich existieren

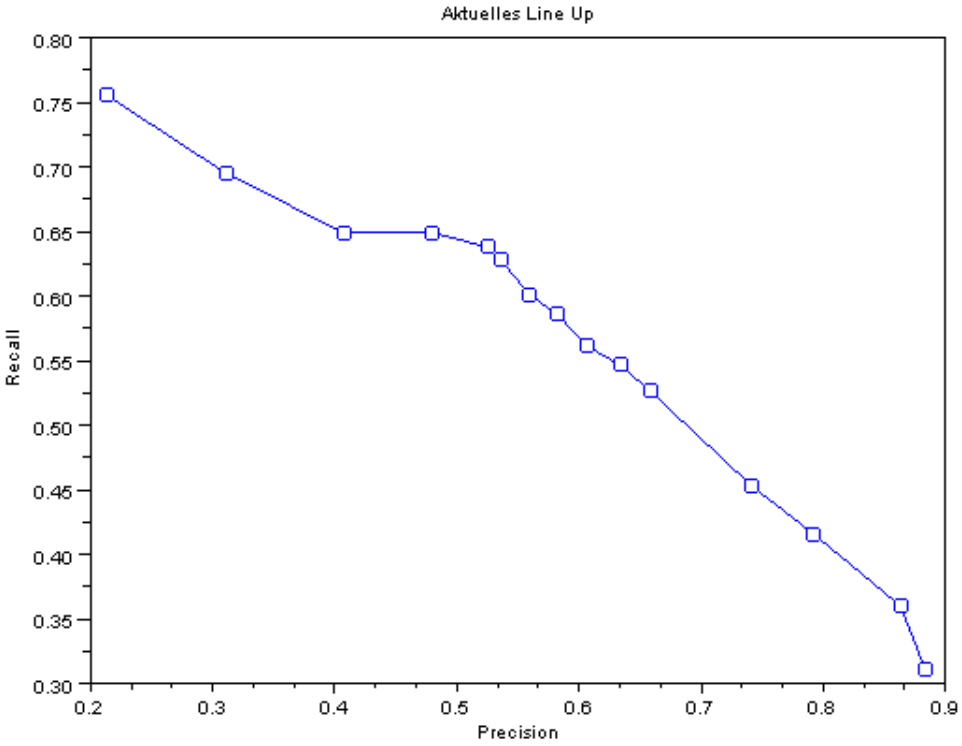

Abbildung 7.3: Precision-Recall Plot für das aktuelle Line Up (Stand 2007)

Ausprägungen mit Rechstschreibfehlern und Konstrukte wie **Brian "Head" Welch**, welche ebenfalls die Precision mindern.

7.3.4 Media Evaluierung

Da für die Evaluierung der **Artist - Media** Beziehungen keine Ground Truth vorliegt, muss diese erst erstellt werden. Dafür werden die Klassen LastFMRetriever bzw. MusicBrainzRetriever verwendet. Beide verfügen über eine Funktionalität, welche eine Liste von Medientiteln eines bestimmten Artists erstellt. Bei subjektiver Analyse der erhaltenen Daten existiert ein eindeutiger Qualitätsunterschied zwischen Last.FM und MusicBrainz. Da Last.FM sehr stark user-orientiert ist, finden sich sehr häufig falsche Medientitel in der Liste. Aus diesem Grund wird die Evaluierung mit MusicBrainz Daten durchgeführt. Dafür wurde ebenfalls ein Verlauf erstellt, welcher die Ergebnisse abhängig von der Häufigkeit der Relationen darstellt. Jener ist in Abbildung 7.4 zu finden.

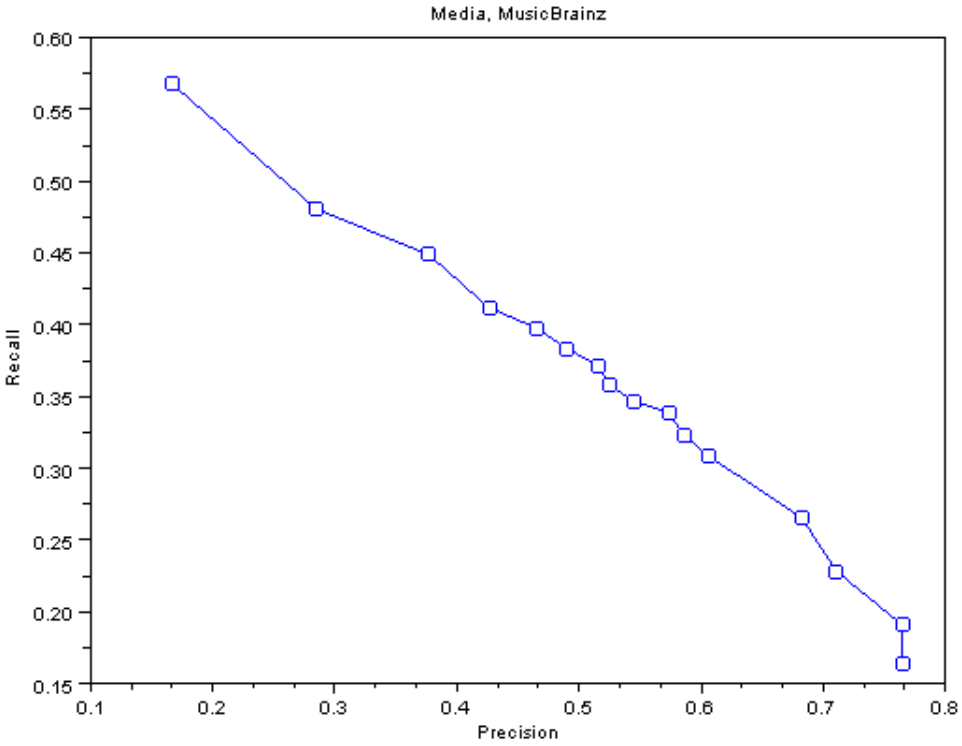

Abbildung 7.4: Precision-Recall Plot für Band Releases

Am Besten schneidet das System mit einem *count* > 6 ab, dabei wird ein F-Measure Wert von 43.18 % erreicht. Einschränkend muss beachtet werden, dass der Testcorpus 2007 erstellt wurde. Die meisten darin enthaltene Künstler sind heute noch aktiv und haben in der Zwischenzeit neue Alben veröffentlicht. Bei Berücksichtigung dessen, steigt dieser Wert noch etwas an, wie in Abbildung 7.5 zu sehen ist.

Die Precision Werte relativieren den ersten Eindruck ein wenig. Dies hat vor allem sehr viele Gründe: Die Titel von Medien variiern sehr stark. Gleiche Titel werden bei Last.FM und Musicbrainz teilweise unterschiedlich geführt. Bei diesen Fällen kann auch eine Normalisierung der Strings auf eine Stammform wenig ausrichten. Bei subjetiver Betrachtung der Relationen finden sich eigentlich sehr viele valide Titel, welche aber etwas von der Ground Truth abweichen. Außerdem annotieren die vorgestellten Regeln auch DVDs, EPs und Compilations. Von MusicBrainz hingegen werden aber nur Alben und Single Titel bezogen. Somit werden oft korrekte Entitäten als falsch gewertet. Diese Erfahrungen können in einer zukünftigen Weiterentwicklung von Nutzen sein.

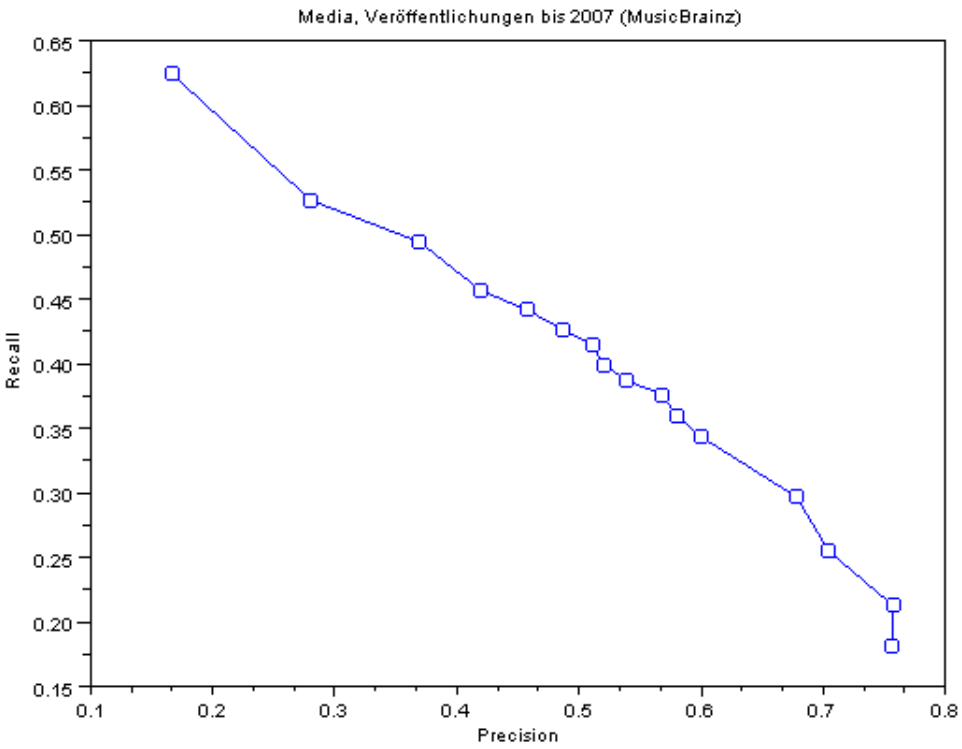

Abbildung 7.5: Precision-Recall Plot für Band Releases bis 2007

7.3.5 Vergleich anhand einer Baseline

Für die Extraktion von Entitäten werden generell Eigennamen verwendet. Da diese Strategie aber nur bedingt bei musikspezifischen Entitäten anwendbar ist, wurden in Kapitel 5.3 neue Strukturen konzipiert. Jene werden im Zuge dieses Kapitels mit dem gängigen Standardverfahren verglichen um deren Vorteile zu verdeutlichen. Für diesen Zweck wurde ein zweiter Relationsgraph erstellt, welcher die Baseline darstellt. Jener besteht gänzlich aus Entitäten, welche durch das Eigennamen-Verfahren (NNP Abfolgen) extrahiert wurden. Berechnet man nun ebenfalls Precision und Recall für diesen Graphen, ist es möglich beide Strategien gegenüber zu stellten. In Abbildung 7.6 wird ein Vergleich anhand der **Artist-Member** Relationen in Form eines Verlaufes dargestellt.

Obwohl die vorgestellte Stratgie einen etwas höheren F-Measure Maximalwert erzielt, sind beide Graphen fast ident. Allerdings existiert zwischen den beiden Member-Annotierungsmethoden nicht so ein gravierender Unterschied wie bei den anderen Entitäten. Dies wird vor allem bei den **Artist-Media** Verknüpfungen deutlich, welche in Abbildung 7.7 verglichen werden.

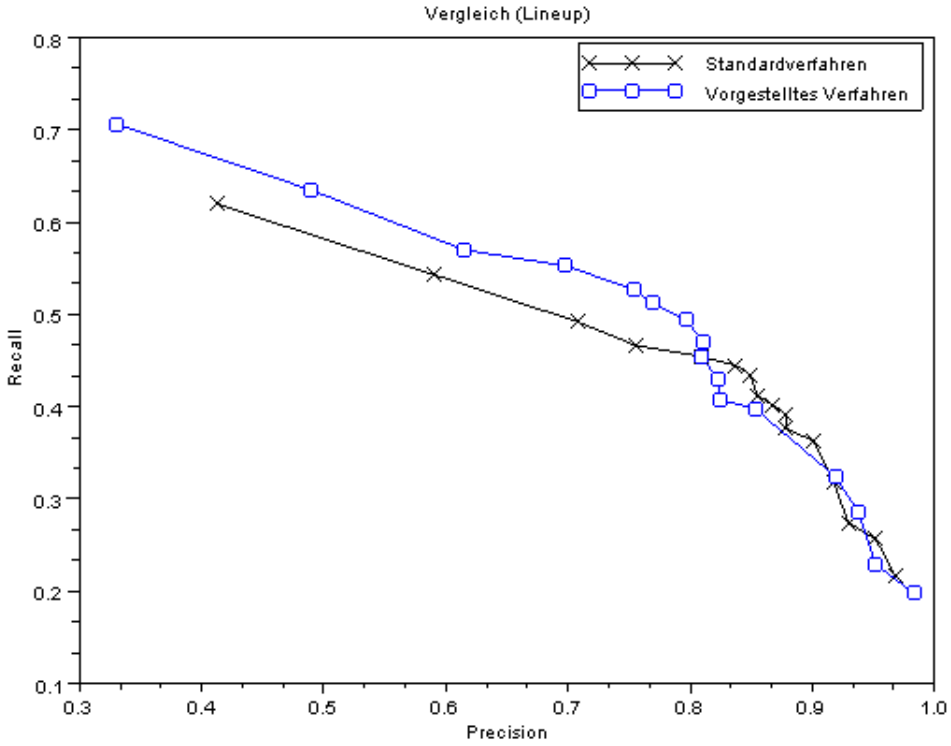

Abbildung 7.6: Vergleich beider Verfahren anhand Artist-Member Relationen

Hier besteht ein erheblich größerer Unterschied zwischen den beiden Verfahren. Ab einer gewissen Relationshäufigkeit (ab etwa $count > 30$) können einige Artists gar keine Entitäten mehr bereitstellen. Eine Ausnahme stellen dabei bekannte Gruppen wie ”Metallica” dar. In diesem konkreten Fall wird deren Sänger ”James Hetfield” ca. 140 Mal mit der Band in Relation gestellt.

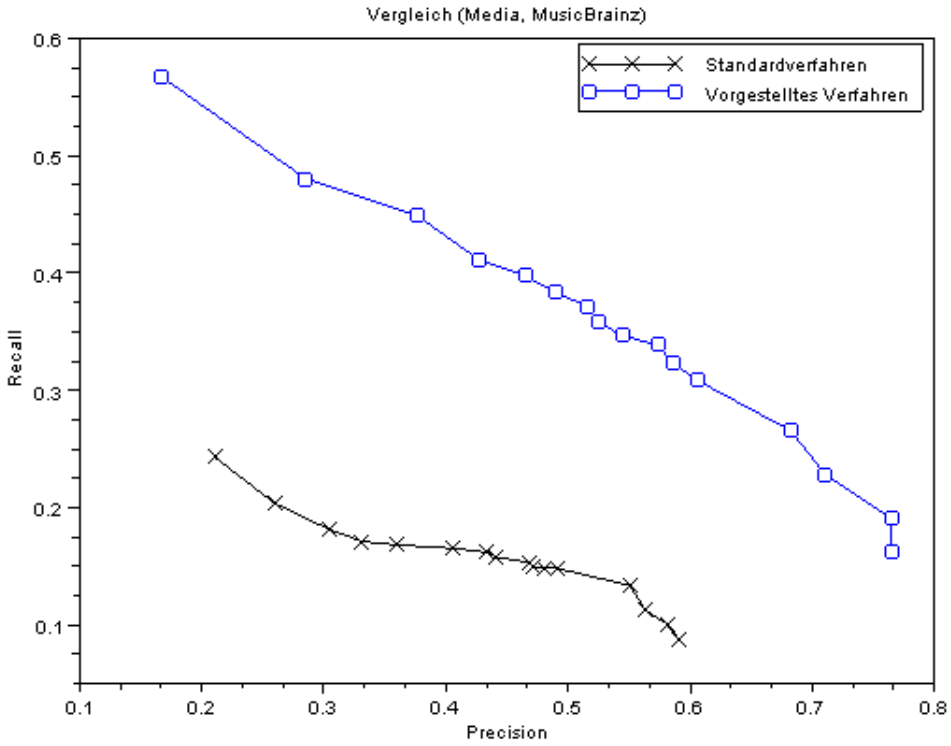

Abbildung 7.7: Vergleich beider Verfahren anhand Artist-Media Relationen

Kapitel 8

Kritische Analyse, Schlußfolgerungen und mögliche Weiterentwicklungen

In dieser Arbeit wurde ein System für die automatische Extraktion von musikspezifischen Informationen vorgestellt. Die dabei konzipierte Basisarchitektur liefert die Grundlage für zahlreiche Anwendungen. Konkret wurde eine Anwendung zur automatischen Extraktion von neuen Regeln, sowie eine Komponente zur Klassifikation von Seiten und eine Implementierung eines Beziehungsnetzwerks erarbeitet. Vor allem die letzten Komponenten wurden in dieser Arbeit einer Evaluierung unterzogen. Für diesen Zweck wurde extra ein Set an Evaluierungstools konzipiert. Diese umfasst einen eigenständigen Editor zur einfachen und schnellen Annotierung von Texten, einen Parser um die markierten Passagen aus dem Text zu beziehen, einfach modifizierbare Evaluierungsmetriken und Hilfsmodule für den einfachen Bezug von relevanten Daten aus dem Internet. Zudem wurde der verwendete Trainingscorpus vollständig händisch annotiert und kann somit auch für das Training von statistischen Verfahren vewendet werden.

Allerdings zeigt sich, dass bei den aufgezählten Punkten immer noch Verbesserungen möglich sind. Vor allem die Regelbasis kann von einer Adaption profitieren. Im Prozess der Evaluierung wurden einige Erkenntnisse gewonnen, um die Perfromance der Regeln noch zu verbessern. Besonders das große Potential der Wissensbasis wird in der aktuellen Regelbasis nicht voll und ganz ausgeschöpft. Speziell bei Strukturen wie Aufzählungen, Tabellen, etc. kann anhand bereits vorhandenem Wissen neues annotiert werden. Im Zuge der Evaluierung fiel besonder folgendes Muster auf:

<Artist> ”-” <Media>

Dafür wird lediglich eine Artist oder Medien Ausprägung benötigt. Außerdem könnte die bestehende Regelbasis durch neu gewonnen Regeln aus der automatischen Regel-Extraktionskomponente erweitert werden. Außerdem besteht die Möglichkeit die Annotierungklassen zu präzisieren (Member → singer, guitarist, usw.). Somit könnten auch exaktere Relationen im Netzwerk verwaltet werden.

Zusätzlich wäre eine Art Blacklist für die Gazetteer Listen denkbar, da ab und an Indikatoren wie CD, DVD, usw. als Band getaggt wurden. Dies würde vor allem den Precision Wert verbessern. Eine rudimentäre Form dieser Blacklist existiert bereits. Wird fälschlicherweise eine Genrebezeichnung als Band annotiert wird diese ignoriert. Eine andere Erweiterung betrifft die Implementierung der Wissensbasis. Im Testcorpus finden sich zahlreiche Ausprägungen der gleichen Band. Die Gruppe Anthrax wurde zum Beispiel in folgenden Variationen annotiert: ANTHRAX, A.N.T.H.R.A.X., etc. Generell wäre hier die Generierung von zwei Klassen denkbar. Ausprägungen welche folgende Eigenschaften besitzen: **UPPERCASE** und **ALLCAPS**. Wird eine davon in die Wissenbasis aufgenommen, so werden alle anderen Variationen erzeugt und ebenfalls in die Basis integriert.

Zudem existieren ein paar GATE interne Probleme: Zum einem die Darstellung von Umlauten als auch die Erkennung von Line Feeds. Seltsamerweise werden keine **Split** Typen mehr annotiert, wenn eine Webseite in das 8859-1 Format konvertiert wird. Außerdem werden gleiche String Ausprägungen bei meheren Sizungen doppelt bzw. mehrfach in der Gazetteer Liste bzw. Wissensbasis gespeichert.

Eventuell ist auch eine kleine Adaptierung bei der Annotierung von Namen notwendig. Vor allem bei der Annotierung von langen Bandnamen bzw. Medientitel werden ab und an umliegende nicht-relevante Wörter miteinbezogen. Zuätzlich hat die Evaluierung gezeigt, dass ein kleiner Teil der Regeln zu offen angelegt wurde. Dies führte teilweise zur Annotierung von nicht korrekten Entitiäten. Allerdings sind die Ergebnisse durchaus positiv zu bewerten. Ein Blick in die Wissenbasis zeigt, dass sowohl einfache Ausprägungen wie "Cult", aber auch komplexere Strukturen wie "Plays Metallica by Four Cellos" gefunden wurden. Schwierig wird es aber erst bei Interpreten wie "Rage" und "Rage Against The Machine", da sich beide Ausprägungen überlappen und eventuell sowohl "Rage" als auch "Rage Against The Machine" getaggt werden. Hinzu kommt noch, dass "Rage Against The Machine" von Fans oft einfach als "Rage" bezeichnet wird, was zusätzlich Probleme schafft. Man sieht also, dass umgangsprachliche Namen wie bekannt problematisch sind, aber Bandnanmen, welche substrings anderer Interpreten sind, die Algorithmen in ihrer vorliegenden Form vollends überfordern. Allerdings existieren solche Fälle in diesem Kontext und verlangen nach einem geeigneten Verfahren, um diese Probleme zu beseitigen. Daher werden die verwendeten Konzepte wohl nie an die Interpretationsfähigkeiten eines Menschen herankommen, sie stellen aber die

gewünschten Informationen aus Tausenden von Seiten für eine weiterführende Analyse (durch Menschenhand) bereit.

Literaturverzeichnis

[1] *Proceedings of the 3rd Conference on Message Understanding, MUC 1991, San Diego, California, USA, May 21-23, 1991*. ACL, 1991.

[2] *Proceedings of the 4th Conference on Message Understanding, MUC 1992, McLean, Virginia, USA, June 16-18, 1992*, 1992.

[3] *Proceedings of the 5th Conference on Message Understanding, MUC 1993, Baltimore, Maryland, USA, August 25-27, 1993*, 1993.

[4] *Proceedings of the 6th Conference on Message Understanding, MUC 1995, Columbia, Maryland, USA, November 6-8, 1995*, 1995.

[5] Douglas E. Appelt, Jerry R. Hobbs, John Bear, David J. Israel, and Mabry Tyson. Fastus: A finite-state processor for information extraction from real-world text. pages 1172–1178, 1993.

[6] Douglas E. Appelt and David Israel. Introduction to information extraction technology. IJCAI-99 Tutorial, August 2 1999.

[7] Douglas E. Appelt and David Martin. Named entity recognition in speech: Approach and results using the TextPro system. In *Proc. DARPA Broadcast News Workshop*, pages 51–54, 1999.

[8] David Brackeen, Bret Barker, and Laurence Vanhelsuwe. *Developing Games in Java*. New Riders Publishing, 2003.

[9] T. Brants. TnT: A Statistical Part-of-Speech Tagger. In *Proceedings of the 6th Conference on Applied Natural Language Processing*, pages 224–231, Seattle, Washington, 2000. ACL.

[10] Eric Brill. A simple rule-based part of speech tagger. In *In Proceedings of the Third Conference on Applied Natural Language Processing*, 1992.

[11] C. Cardie. Empirical methods in information extraction. *AI Journal*, 18(4):65–79, 1997.

[12] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[13] G. Chroust. *Vorlesungsunterlagen: Software Engineering 1.* 2006. Johannes Kepler Universität Linz, Austria.

[14] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. Towards the self-annotating web. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 462–471, New York, NY, USA, 2004. ACM Press.

[15] Philipp Cimiano and Steffen Staab. Learning by googling. *SIGKDD Explor. Newsl.*, 6(2):24–33, 2004.

[16] Fabio Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 2001. Seattle.

[17] E. Codd. A Relational Model of Data for Large Shared DataBases. *Communication of the ACM*, June 1970.

[18] E. Codd. Further normalisation of the database relational models. In R. Rustin, editor, *Database Systems*, pages 33–74. Prentice-Hall, 1972.

[19] Jim Cowie and Yorick Wilks. *Information Extraction.* A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text. Marcel Dekker Inc., New York, USA, 2000.

[20] P. Coxhead. Natural language processing & applications, 2001. University of Birmingham.

[21] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, N. Aswani, I. Roberts, Y. Li, and A. Shafirin. Developing language processing components with gate version 5 (a user guide). *University of Sheffield*, 2005.

[22] H. Cunningham, Y. Wilks, and R. Gaizauskas. GATE – a General Architecture for Text Engineering. In *Proceedings of the 16th Confe-*

rence on Computational Linguistics (COLING-96), Copenhagen, August 1996. `ftp://ftp.dcs.shef.ac.uk/home/hamish/auto_papers/Cun96b.ps`.

[23] Hamish Cunningham. JAPE: a Java Annotation Patterns Engine. Research Memorandum CS – 99 – 06, Department of Computer Science, University of Sheffield, May 1999.

[24] Hamish Cunningham. *Software Architecture for Language Engineering*. PhD thesis, University of Sheffield, 2000. `http://gate.ac.uk/sale/thesis/`.

[25] Hamish Cunningham, Kevin Humphreys, Robert Gaizauskas, and Yorick Wilks. GATE - a TIPSTER-based general architecture for text engineering. In *Proc. of the TIPSTER Text Program Phase III*. Morgan Kaufmann, 1997.

[26] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA*, 2002.

[27] D. Downey, O. Etzioni, S. Soderlandand, and D. S. Weld. Learning Text Patterns for Web Information Extraction and Assessment. In *AAAI'04 Workshop on Adaptive Text Extraction and Mining - San Joseand California*, July 26 2004.

[28] Line Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.

[29] Oren Etzioni, Michael J. Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Methods for domain-independent information extraction from the web: An experimental comparison. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 391–398. AAAI Press / The MIT Press, 2004.

[30] A. Ferscha. *Vorlesungsunterlagen: Algorithmen und Datenstrukturen 2*. 2005. Johannes Kepler Universität Linz, Austria.

[31] Michael Fleischman, Eduard Hovy, and Abdessamad Echihabi. Offline strategies for online question answering: answering questions before they are asked. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 1–7, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[32] Dianne Freitag and A. McCallum. Information extraction with hmms and shrinkage. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[33] R. Weinreich G. Blaschek. *Vorlesungsunterlagen: Software-Architekturen.* 2008. Johannes Kepler Universität Linz, Austria.

[34] Ralph Grishman. Information extraction techniques and challenges. In Maria Teresa Pazienza, editor, *Information Extraction: a multidisciplinary approach to an emerging technology*, 1997.

[35] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*, pages 539–545, Nantes, France, July 1992.

[36] Mark Hepple. Independence and commitment assumptions for rapid training and execution of rule-based part-of-speech taggers. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, 2000. Hong Kong.

[37] Kevin Humphreys, Rob Gaizauskas, Hamish Cunningham, and Saliha Azzam. GATE: VIE technical specifications. Technical report, ILASH, University of Sheffield, 1996. Included in the documentation of GATE 1.0.0.

[38] W. Wöß J. Küng, B. Pröll. *Vorlesungsunterlagen: Knowledge Centered Systems.* 2008. Johannes Kepler Universität Linz, Austria.

[39] Alfons Kemper and André Eickler. *Datenbanksysteme.* Oldenbourg, München [u.a.], 6., aktualisierte und erw. aufl edition, 2006.

[40] Kalina Bontcheva Yaoyong Li and Hamish Cunningham. SVM based learning system for information extraction. In Joab Winkler, Neil Lawrence, and Mahesan Niranjan, editors, *Deterministic and Statistical Methods in Machine Learning: First International Workshop, Sheffield, UK, September 7–10, 2004. Revised Lectures*, volume 3635 of *Lecture Notes in Computer Science*, pages 319–339, Berlin, 2005. Springer.

[41] H. Mössenböck. *Vorlesungsunterlagen: Übersetzerbau.* 2006. Johannes Kepler Universität Linz, Austria.

[42] MUC7. *Proceedings of the 7th Message Understanding Conference (MUC7)*. NIST, 1998.

[43] J. Peckham and F. Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(3):153–189, September 1988.

[44] B. Pröll. *Vorlesungsunterlagen: Information Retrieval*. 2007. Johannes Kepler Universität Linz, Austria.

[45] Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a question answering system. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 41–47, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[46] Sunita Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[47] Markus Schedl, Peter Knees, Klaus Seyerlehner, and Tim Pohle. The CoMIR-VA Toolkit for Visualizing Music-Related Data. In *Proceedings of the 9th Eurographics/IEEE VGTC Symposium on Visualization (EuroVis'07)*, Norrköping, Sweden, May 2007.

[48] Markus Schedl, Gerhard Widmer, Tim Pohle, and Klaus Seyerlehner. Web-based Detection of Music Band Members and Line-Up. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR'07)*, Vienna, Austria, September 2007.

[49] Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI'99 Workshop on Machine Learning for Information Extraction*, 1999.

[50] S. Shapiro. Path-based and node-based inference in semantic networks. In *Proceedings of the Workshop on Theoretical Issues in Natural Language Processing*, 1978.

[51] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1297–1304. MIT Press, Cambridge, MA, 2005.

[52] Beth M. Sundheim and Nancy A. Chinchor. Survey of the message understanding conferences. In *HLT '93: Proceedings of the workshop on Human Language Technology*, pages 56–60, Morristown, NJ, USA, 1993. Association for Computational Linguistics.

[53] Valentin Tablan, Diana Maynard, Kalina Bontcheva, and Hamish Cunningham. GATE - an application developer's guide. Technical report, Department of Computer Science, University of Sheffield, 2004.

[54] Unicode Consortium. *The Unicode Standard: Version 4.0.* Addison Wesley, Reading, Massachusetts, August 2003.

[55] Roland Wagner. *Vorlesungsunterlagen: Informationssystem 1.* 2005. Johannes Kepler Universität Linz, Austria.

[56] F. Winkler. *Vorlesungsunterlagen: Formale Grundlagen 2.* 2005. Johannes Kepler Universität Linz, Austria.

[57] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 2279 (Draft Standard), January 1998. Obsoleted by RFC 3629.

Anhang A

Spezifische Wortlisten

A.1 genre.lst

acapella
Acapella
acoustic rock
Acoustic rock
Acoustic Rock
acid
Acid
acid jazz
Acid jazz
acid rock
Acid rock
alternative
Alternative
alternative dance
Alternative dance
Alternative Dance
alternative metal
Alternative metal
Alternative Metal
alternative rock
Alternative rock
Alternative Rock
ambient
Ambient
art rock
Art rock
Experimental Rock
folk
Folk
folk rock
Folk rock
Folk Rrock
folk-rock
Folk-rock
funk
Funk
funk metal
Funk metal
Funk Metal
funk rock
Funk rock
Funk Rock
garage revival
Garage revival
Garage Revival
garage rock
Garage rock
Garage Rock
glam metal
Glam metal
Glam Metal
glam rock
nu metal
Nu metal
Nu Metal
nu-metal
Nu-metal
Metal
metal
pop
Pop
pop punk
Pop punk
post-punk
Post-punk
post-punk revival
Post-punk revival
Post-punk Revival
pop rock
Pop rock
Pop Rock
post-grunge
Post-grunge
power metal
Power metal
Power Metal
power pop
Power pop

Art Rock
big beat
Big beat
Big Beat
Black Metal
Black metal
black metal
blues
Blues
blues-rock
Blues-rock
blues rock
Blues rock
Blues Rock
breakbeat
Breakbeat
britpop
Britpop
chamber music
Chamber music
Chamber Music
college rock
College rock
College Rock
comedy rock
Comedy rock
Comedy Rock
dance
Dance
death metal
Death metal
Death Metal
dance punk
Dance punk
Dance Punk
disco
Disco
disco house
Disco house
Disco House
electro
Electro
Glam rock
Glam Rock
groove
Groove
groove metal
Groove metal
Groove Metal
grunge
Grunge
hard rock
Hard rock
Hard Rock
hardcore
Hardcore
hardcore punk
Hardcore punk
Hardcore Punk
hardcore techno
Hardcore techno
Hardcore Techno
heartland rock
Heartland rock
Heartland Rock
heavy metal
Heavy metal
Heavy Metal
hip-hop
Hip-hop
hip hop
Hip hop
Hip Hop
house
House
indie rock
Indie rock
industrial rock
Industrial rock
Industrial Rock
industrial metal
Industrial metal
Industrial Metal
jangle pop
Power Pop
punk rock
Punk rock
Punk Rock
progressive
Progressive
Progressive Metal
progressive metal
Progressive metal
progressive rock
Progressive rock
Progressive Rock
psychedelic rock
Psychedelic rock
Psychedelic Rock
rapcore
Rapcore
rap metal
Rap metal
Rap Metal
rap rock
Rap rock
Rap Rock
rave
Rave
rock
Rock
rock and roll
Rock and roll
rhythm and blues
Rhythm and blues
shock rock
Shock rock
ska
Ska
ska punk
Ska punk
ska-punk
Ska-punk
skate punk
Skate punk
soul

electro house
Electro house
Electro House
electronic
Electronic
electronic rock
Electronic rock
Electronic Rock
electronica
Electronica
europop
Europop
experimental
Experimental
experimental rock
Experimental rock
Jangle pop
jazz
Jazz
jazz fusion
Jazz fusion
melodic hardcore
Melodic hardcore
Melodic Hardcore
neo-psychedelia
Neo-psychedelia
new wave
New wave
New Wave
noise rock
Noise rock
Noise Rock
Soul
space rock
Sspace rock
speed metal
Speed metal
symphonic metal
Symphonic metal
synthpop
Synthpop
techno
Techno
thrash metal
Thrash metal
trip hop
Trip hop

A.2 instrument.lst

accordion
Accordion
acoustic guitar
Acoustic guitar
backing vocals
Backing vocals
bass
Bass
bass guitar
Bass guitar
cello
Cello
didgeridoo
Didgeridoo
drums
Drums
guitar
Guitar
guitars
Guitars
harmonica
Harmonica
keys
Keys
keyboard
Keyboard
keyboards
Keyboards
lead guitar
Lead guitar
lead vocals
Lead vocals
mandolin
Mandolin
organ
Organ
percussion
Percussion
piano
Piano
rhythm guitar
Rhythm guitar
saxophone
Saxophone
sitar
Sitar
songwriting
Songwriting
synth
Synth
synthesizer
Synthesizer
trumpet
Trumpet
tuba
Tuba
turntables
Turntables
violin
Violin
virtuoso guitar
Virtuoso guitar
vocals
Vocals

A.3 role.lst

bass player
Bass player
bassist
Bassist
cellist
Cellist
drummer
Drummer
frontman
Frontman
guitarist
Guitarist
keyboarder
Keyboarder
keyboardist
Keyboardist
keyboard player
Keyboard player
organist
Organist
percussionist
Percussionist
percoussionist
Percoussionist
roadie
Roadie
rhythm guitarist
Rhythm guitarist
saxophonist
Saxophonist
singer
Singer
songwriter
Songwriter
vocalist
Vocalist
turntablist
Turntablist

Anhang B

JAPE Regeln

B.1 SecureMediaRules.jape

```
1  Phase: SecureMediaRules
2  Input: Token Lookup Split
3  Options: control = appelt
4
5  Macro: Apostroph
6  (
7    ({Token.orth == apostrophe} |
8    {Token.category == POS, Token.kind == punctuation, Token.length == 1}) |
9    ({Token.kind == punctuation, Token.length == 1, Token.string == "'"}
10   ({Token.length == 1, Token.string == "s"})?)
11 )
12
13 Macro: Medium
14 (
15   ({Token.string != "\""})[0,16]
16 )
17
18 Rule: SecureMediaComma
19 (
20   ({Token.string == "album"} | {Token.string == "demo"} |
21   {Token.string == "debut"} | {Token.string == "effort"} |
22   {Token.string == "single"} | {Token.string == "record"} |
23   {Token.string == "release"} | {Token.string == "track"} |
24   {Token.string == "song"} |{Token.string == "hit"} |
25   {Token.string == "follow-up"} | {Token.string == "LP"} |
26   {Token.string == "EP"} | {Token.string == "CD"} |
27   {Token.string == "DVD"} |{Token.string == "full-length"} |
28   {Token.string == "video"} | {Token.string == "collection"})
29   ({Token.string == ","})?
```

```
30   {Token.string == "\""}
31   (
32     (Medium)
33   ):Media
34   {Token.string == "\""}
35 )--> :Media.Media = {kind = "Media", rule = "SecureMediaComma"}
36
37 Rule: SecureWordOf
38 (
39   ({Token.string == "release"} | {Token.string == "riff"} |
40   {Token.string == "performance"} | {Token.string == "recording"} |
41   {Token.string == "remix"} | {Token.string == "chorus"} |
42   {Token.string == "lyrics"} | {Token.string == "sound"} |
43   {Token.string == "version"} | {Token.string == "cover"} |
44   {Token.string == "success"} | {Token.string == "medley"})
45   {Token.string == "of"}
46   {Token.string == "\""}
47   (
48     (Medium)
49   ):Media
50   {Token.string == "\""}
51 )--> :Media.Media = {kind = "Media", rule = "SecureWordOf"}
52
53 Rule: SecureMediaTitled
54 (
55   ({Token.string == "album"} | {Token.string == "demo"} |
56   {Token.string == "debut"} | {Token.string == "effort"} |
57   {Token.string == "single"} | {Token.string == "record"} |
58   {Token.string == "release"} | {Token.string == "track"} |
59   {Token.string == "song"} |{Token.string == "hit"} |
60   {Token.string == "follow-up"} | {Token.string == "LP"} |
61   {Token.string == "EP"} | {Token.string == "CD"} |
62   {Token.string == "DVD"} |{Token.string == "full-length"} |
63   {Token.string == "video"} |    {Token.string == "collection"})
64   ({Token.string == ","})?
65   ({Token.string == "titled"} | {Token.string == "entitled"} |
66   {Token.string == "named"} | {Token.string == "called"})
67   {Token.string == "\""}
68   (
69     (Medium)
70   ):Media
71   {Token.string == "\""}
72 )--> :Media.Media = {kind = "Media", rule = "SecureMediaTitled"}
73
74 Rule: SecureYearsAlbum
75 (
76   {Lookup.majorType == year}
```

```
77    (Apostroph)
78    {Token.string == "\""}
79    (
80      (Medium)
81    ):Media
82    {Token.string == "\""}
83  )--> :Media.Media = {kind = "Media", rule = "SecureYearsAlbum"}
84
85  Rule: SecureVideo
86  (
87    ({Token.string == "video"} | {Token.string == "videoclip"})
88    {Token.string == "for"}
89    {Token.string == "\""}
90    (
91      (Medium)
92    ):Media
93    {Token.string == "\""}
94  )--> :Media.Media = {kind = "Media", rule = "SecureVideo"}
95
96  Rule: SecureMediaList
97  (
98  ({Token.string == "albums"} | {Token.string == "songs"} |
99  {Token.string == "tracks"} |{Token.string == "singles"} |
100 {Token.string == "hits"} | {Token.string == "ballads"})
101 ({Token.string == ","} | {Token.string == ":"})?
102 ((
103 {Token.string == "\""}(Medium){Token.string == "\""}
104 ({Token.string == ","}
105 {Token.string == "\""}(Medium){Token.string == "\""})[0,7]
106 ({Token.string == ","})?
107 {Token.string == "and"}
108 {Token.string == "\""}(Medium){Token.string == "\""}
109 )):mention
110 )
111 -->
112 {
113 List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
114 Collections.sort(annList, new OffsetComparator());
115 Node start = null;
116 Node end = null;
117 boolean isStart = true;
118 char quote = 34;
119 for(int i = 0; i < annList.size(); i++){
120   Annotation anAnn = (Annotation)annList.get(i);
121   String word = anAnn.getFeatures().get("string").toString();
122   if(word.charAt(0) == quote){
123     if(isStart){
```

```
124       isStart = false;
125     }else{
126       FeatureMap features = Factory.newFeatureMap();
127             features.put("rule", "SecureMediaList");
128             annotations.add(start , end, "Media", features);
129       isStart = true;
130       start = null;
131     }
132   }else{
133     if(start == null && !word.equals(",") && !word.equals("and")){
134             start = anAnn.getStartNode();
135     }
136     end = anAnn.getEndNode();
137   }
138 }
139 }
140
141 Rule: SecureHearstMedia1
142 (
143 ({Token.string == "albums"} | {Token.string == "songs"} |
144 {Token.string == "tracks"} | {Token.string == "singles"} |
145 {Token.string == "hits"} | {Token.string == "ballads"})
146 ({Token.string == ","})?
147 ({Token.string == "including"} | {Token.string == "like"} |
148 ({Token.string == "such"}{Token.string == "as"}))
149 ((
150 {Token.string == "\""}(Medium){Token.string == "\""}
151 ({Token.string == ","}
152 {Token.string == "\""}(Medium){Token.string == "\""})[0,7]
153 ({Token.string == ","})?{Token.string == "and"}
154 {Token.string == "\""}(Medium){Token.string == "\""}
155 )):mention
156 )
157 -->
158 {
159 List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
160 Collections.sort(annList, new OffsetComparator());
161 Node start = null;
162 Node end = null;
163 boolean isStart = true;
164 char quote = 34;
165 for(int i = 0; i < annList.size(); i++){
166   Annotation anAnn = (Annotation)annList.get(i);
167   String word = anAnn.getFeatures().get("string").toString();
168   if(word.charAt(0) == quote){
169     if(isStart){
170       isStart = false;
```

```
171     }else{
172       FeatureMap features = Factory.newFeatureMap();
173             features.put("rule", "SecureHearstMedia1");
174             annotations.add(start , end, "Media", features);
175       isStart = true;
176       start = null;
177     }
178   }else{
179     if(start == null && !word.equals(",") && !word.equals("and")){
180             start = anAnn.getStartNode();
181     }
182     end = anAnn.getEndNode();
183   }
184    }
185
186 }
```

B.2 SimpleBandRules.jape

```
1
2 Phase: Band
3 Input: Token Lookup Split
4 Options: control = appelt
5
6 Macro: Apostroph
7 (
8   ({Token.orth == apostrophe} |
9   {Token.category == POS, Token.kind == punctuation, Token.length == 1}) |
10  ({Token.kind == punctuation, Token.length == 1, Token.string == "'"}
11  ({Token.length == 1, Token.string == "s"})?)
12 )
13
14 Macro: Shortcut
15 (
16  {Token.orth == upperInitial, Token.length <= 2}
17  {Token.string == "."}
18 )
19
20 Macro: Start
21 (
22  ({Token.orth == upperInitial} | Shortcut |
23  {Token.orth == allCaps} | {Token.orth == mixedCaps} |
24  {Token.kind == symbol})
25 )
26
27 Macro: End
```

```
28 (
29   ({Token.orth == upperInitial} | Shortcut |
30   {Token.orth == allCaps} | {Token.orth == mixedCaps} |
31   {Token.category == CD, Token.kind == number, Token.length != 4} |
32   {Token.string == "!"})
33 )
34
35 Macro: StartSingle
36 (
37   ({Token.orth == upperInitial, Token.category != IN,
38   Token.category != DT, Token.category != PRP, Token.category != JJ,
39   Token.category != "PRP$", Token.category != JJS, Token.category != RBR,
40   Token.category != RBS,Token.category != JJR} |
41   {Token.orth == allCaps, Token.category != IN, Token.category != DT,
42   Token.category != PRP, Token.category != JJ, Token.category != "PRP$",
43   Token.category != JJS, Token.category != RBR, Token.category != RBS,
44   Token.category != JJR} | {Token.orth == mixedCaps, Token.category != IN,
45   Token.category != DT, Token.category != PRP, Token.category != JJ,
46   Token.category != "PRP$", Token.category != JJS, Token.category != RBR,
47   Token.category != RBS, Token.category != JJR})
48 )
49
50
51 Macro: Follow
52 (
53   ({Token.category == MD, Token.orth == upperInitial} |
54   {Token.category == VB, Token.orth == upperInitial} |
55   {Token.category == VBP, Token.orth == upperInitial} |
56   {Token.category == VBG, Token.orth == upperInitial} |
57   {Token.category == VBZ, Token.orth == upperInitial} |
58   {Token.category == TO} | {Token.category == DT} |
59   {Token.category == POS} | {Token.category == CD} |
60   {Token.category == IN} | {Token.category == RBR} |
61   {Token.category == JJR, Token.orth == upperInitial} |
62   {Token.category == NN, Token.position != endpunct, Token.length == 1} |
63   {Token.category == JJS, Token.orth == upperInitial} |
64   {Token.category == JJ, Token.orth == upperInitial} |
65   {Token.category == NN, Token.orth == upperInitial} |
66   {Token.category == NNP, Token.orth == upperInitial} |
67   {Token.category == NNPS, Token.orth == upperInitial} |
68   {Token.category == PRP} | {Token.category == "PRP$"} |
69   {Token.string == "&"} | Shortcut)
70 )
71
72
73 Macro: BandN
74 (
```

```
75   (Start (Follow)+ End)  | (Start End) | (StartSingle)
76 )
77
78 Rule: consistsOf
79 (
80 (
81   (BandN)
82 ):BandName
83   {Token.string  == "consists"}
84   {Token.string  == "of"}
85 )--> :BandName.bandname = {kind = "Band", rule = "consistsOf"}
86
87 Rule: soundsOf
88 (
89   ({Token.string == "disband"} | {Token.string == "breakup"}
90   | {Token.string == "reunion"} | {Token.string == "cover"} |
91   {Token.string == "remix"} | {Token.string == "sound"} |
92   {Token.string == "sounds"} | {Token.string == "roots"} |
93   {Token.string == "covers"} | {Token.string == "music"} |
94   {Token.string == "formerly"} | {Token.string == "performance"})
95   {Token.string == "of"}
96 (
97   (BandN)
98 ):BandName
99   {Token.string != "in"}
100 )--> :BandName.bandname = {kind = "Band", rule = "soundsOf"}
101
102 Rule: disbandReunion
103 (
104 (
105   (BandN)
106 ):BandName
107   ({Token.string == "disbanded"} | {Token.string == "reunited"})
108 )--> :BandName.bandname = {kind = "Band", rule = "disbandReunion"}
109
110 Rule: isA
111 (
112 (
113   (BandN)
114 ):BandName
115   ({Token.string == "is"} | {Token.string == "are"} |
116   {Token.string == "was"} | {Token.string == "were"})
117   ({Token.string == "a"} | {Token.string == "an"})
118   ({Lookup.majorType == country_adj})?
119   ({Lookup.majorType == genre})?
120   ({Token.string == "band"} | {Token.string == "group"})
121 )--> :BandName.bandname = {kind = "Band", rule = "isA"}
```

```
122
123 Rule: BandAposAlbum
124 (
125 (
126   (BandN)
127 ):BandName
128   (Apostroph)
129   (
130     {Token.category == JJ, Token.orth == lowercase} |
131     {Token.string == "self-titled"} |
132     {Token.string == "full-length"} |
133     {Token.string == "breakthrough"} |
134     ({Token.string == "most"}{Token.string == "successful"}) |
135     ({Token.string == "least"}{Token.string == "successful"})
136   )?
137   ({Token.string == "album"} | {Token.string == "record"} |
138   {Token.string == "tour"} | {Token.string == "song"} |
139   {Token.string == "single"} | {Token.string == "albums"} |
140   {Token.string == "records"} | {Token.string == "singles"} |
141   {Token.string == "performance"})
142 )--> :BandName.bandname = {kind = "Band", rule = "BandAposAlbum"}
143
144 Rule: Covered
145 (
146 (
147   (BandN)
148 ):BandName
149   ({Token.string == "has"} | {Token.string == "have"} |
150   {Token.string == "had"})?
151   ({Token.string == "also"})?
152   ({Token.string == "covered"})
153 )--> :BandName.bandname = {kind = "Band", rule = "Covered"}
154
155 Rule: Released
156 (
157 (
158   (BandN)
159 ):BandName
160   ({Token.string == "has"} | {Token.string == "have"} |
161   {Token.string == "had"})?
162   ({Token.string == "also"})?
163   ({Token.string == "released"} | {Token.string == "re-released"} |
164   {Token.string == "self-released"} | {Token.string == "issued"})
165   ({Token.string != "in", Token.string != "by"})
166 )--> :BandName.bandname = {kind = "Band", rule = "Released"}
167
168 Rule: Toured
```

```
(
(
  (BandN)
):BandName
  ({Token.string == "has"} | {Token.string == "have"} |
  {Token.string == "had"})?
  ({Token.string == "toured"} | {Token.string == "signed"} |
  {Token.string == "supported"})
)--> :BandName.bandname = {kind = "Band", rule = "Toured"}

Rule: PastGerund
(
(
  (BandN)
):BandName
  {Token.string == "was"}
  ({Token.string == "touring"} | {Token.string == "playing"} |
  {Token.string == "recording"})
)--> :BandName.bandname = {kind = "Band", rule = "PastGerund"}

Rule: Headlined
(
(
  (BandN)
):BandName
  ({Token.string == "has"} | {Token.string == "have"} |
  {Token.string == "had"})?
  ({Token.string == "headlined"} | {Token.string == "co-headlined"})
)--> :BandName.bandname = {kind = "Band", rule = "Headlined"}

Rule: Formed
(
(
  (BandN)
):BandName
  ({Token.string == "was"} | {Token.string == "were"})
  ({Token.string == "formed"} | {Token.string == "supported"} |
  {Token.string == "founded"})
)--> :BandName.bandname = {kind = "Band", rule = "Formed"}

Rule: Recorded
(
(
  (BandN)
):BandName
  ({Token.string == "has"} | {Token.string == "have"} |
  {Token.string == "had"})?
```

```
216   ({Token.string == "recorded"} | {Token.string == "re-recorded"})
217 )--> :BandName.bandname = {kind = "Band", rule = "Recorded"}
218
219 Rule: Formed
220 (
221 (
222   (BandN)
223 ):BandName
224   {Token.string == "formed"}
225   {Token.string == "in"}
226 )--> :BandName.bandname = {kind = "Band", rule = "Formed"}
227
228 Rule: Performed
229 (
230 (
231   (BandN)
232 ):BandName
233   ({Token.string == "has"} | {Token.string == "have"} |
234   {Token.string == "had"})?
235   ({Token.string == "debuted"} | {Token.string == "opened"} |
236   {Token.string == "played"})
237   {Lookup.majorType != instrument}
238 )--> :BandName.bandname = {kind = "Band", rule = "Performed"}
239
240 Rule: LineUp
241 (
242 (
243   (BandN)
244 ):BandName
245   (Apostroph)
246   ({Token.category == JJ, Token.orth == lowercase})?
247   ({Token.string == "lineup"} | {Token.string == "line-up"} |
248   ({Token.string == "line"}{Token.string == "up"}) |
249   {Lookup.majorType == role})
250 )--> :BandName.bandname = {kind = "Band", rule = "LineUp"}
251
252 Rule: BandFollow
253 (
254 (
255   (BandN)
256 ):BandName
257   (Apostroph)
258   ({Token.category == JJ, Token.orth == lowercase} |
259   {Lookup.majorType == year})?
260   ({Token.string == "follow-up"} | {Token.string == "music"} |
261   {Token.string == "sound"} | {Token.string == "record"} |
262   {Token.string == "release"} | {Token.string == "album"} |
```

```
263   {Token.string == "song"} | {Token.string == "single"} |
264   {Token.string == "debut"})
265 )--> :BandName.bandname = {kind = "Band", rule = "BandFollow"}
266
267 Rule: openedFor
268 (
269   ({Token.string == "opened"} |
270   {Token.string == "openening"} |
271   {Token.string == "support"})
272   {Token.string == "for"}
273 (
274   (BandN)
275 ):BandName
276 )--> :BandName.bandname = {kind = "Band", rule = "openedFor"}
277
278 Rule: BandFollow2
279 (
280 (
281   (BandN)
282 ):BandName
283   ({Token.string == "albums"} | {Token.string == "records"} |
284   {Token.string == "singles"} | {Token.string == "tracks"} |
285   {Token.string == "songs"} | {Token.string == "concerts"})
286 )--> :BandName.bandname = {kind = "Band", rule = "BandFollow2"}
287
288 Rule: Audience
289 (
290 (
291   (BandN)
292 ):BandName
293   (Apostroph)
294   ({Token.string == "fanbase"} | {Token.string == "audience"})
295 )--> :BandName.bandname = {kind = "Band", rule = "Audience"}
296
297 Rule: Fan
298 (
299 (
300   (BandN)
301 ):BandName
302   ({Token.string == "fan"} | {Token.string == "fans"} |
303   {Token.string == "maniac"}  | {Token.string == "maniacs"})
304 )--> :BandName.bandname = {kind = "Band", rule = "Fan"}
305
306 Rule: leftJoined
307 (
308   ({Token.string == "joined"} | {Token.string == "left"})
309 (
```

```
310   (BandN)
311 ):BandName
312 )--> :BandName.bandname = {kind = "Band", rule = "leftJoined"}
313
314 Rule: nameFollow
315 (
316   ({Token.string == "band"} | {Token.string == "supergroup"}
317   {Token.string == "group"} | {Token.string == "act"} |
318   {Token.string == "artist"} | {Token.string == "duo"} |
319   {Token.string == "trio"} | {Token.string == "quartet"} |
320   {Token.string == "quintet"})
321   ({Token.string == ","})?
322 (
323   (BandN)
324 ):BandName
325 )--> :BandName.bandname = {kind = "Band", rule = "nameFollow"}
326
327 Rule: Called
328 (
329   ({Token.string == "band"} | {Token.string == "supergroup"} |
330   {Token.string == "group"} | {Token.string == "act"} |
331   {Token.string == "artist"} | {Token.string == "project"})
332   ({Token.string == "called"} | {Token.string == "named"})
333 (
334   (BandN)
335 ):BandName
336 )--> :BandName.bandname = {kind = "Band", rule = "Called"}
337
338 Rule: toVerb
339 (
340   {Token.string == "to"}
341   ({Token.string == "form"} | {Token.string == "join"} |
342   {Token.string == "leave"} | {Token.string == "found"})
343 (
344   (BandN)
345 ):BandName
346 )--> :BandName.bandname = {kind = "Band", rule = "toVerb"}
347
348 Rule: VerbFollow
349 (
350   ({Token.string == "formed"} | {Token.string == "founded"} |
351   {Token.string == "co-founded"})
352 (
353   (BandN)
354 ):BandName
355 )--> :BandName.bandname = {kind = "Band", rule = "VerbFollow"}
356
```

```
Rule: AdjBand
(
  ({Token.category == JJ, Token.orth == lowercase} |
  {Token.category == JJS, Token.orth == lowercase})
(
  (BandN)
):BandName
  ({Token.string == "album"} | {Token.string == "record"} |
  {Token.string == "song"} | {Token.string == "track"} |
  {Token.string == "single"} | {Token.string == "CD"} |
  {Token.string == "EP"} | {Token.string == "LP"})
)--> :BandName.bandname = {kind = "Band", rule = "AdjBand"}

Rule: FavouriteBand
(
(
  (BandN)
):BandName
  {Token.string == "is"}
  ({Token.string == "one"}{Token.string == "of"})?
  {Token.string == "my"}
  ({Token.string == "personal"})?
  {Token.string == "favourite"}
  ({Token.string == "band"} | {Token.string == "bands"})
)--> :BandName.bandname = {kind = "Band", rule = "FavouriteBand"}

Rule: Compare

(

  {Token.string == "compare"}

(
  (BandN)

):BandName
  {Token.string == "to"}
  ({Token.string == "any"})?
  ({Token.string == "other"})?
  ({Token.string == "band"} | {Token.string == "bands"})
)--> :BandName.bandname = {kind = "Band", rule = "Compare"}

Rule: MediaBy
(
  ({Token.string == "album"} | {Token.string == "albums"} |
  {Token.string == "record"} | {Token.string == "records"} |
  {Token.string == "CD"} | {Token.string == "LP"} |
```

```
404   {Token.string == "song"} | {Token.string == "songs"} |
405   {Token.string == "single"} | {Token.string == "track"} |
406   {Token.string == "covered"})
407   {Token.string == "by"}
408 (
409   (BandN)
410 ):BandName
411 )--> :BandName.bandname = {kind = "Band", rule = "MediaBy"}
412
413 Rule: BandProfession
414 (
415 (
416   (BandN)
417 ):BandName
418   ({Lookup.majorType == role} | {Token.string == "member"})
419 )--> :BandName.bandname = {kind = "Band", rule = "BandProfession"}
420
421 Rule: OfBand
422 (
423   ({Token.string == "members"} | {Token.string == "member"} |
424   {Token.string == "fan"}| {Token.string == "fans"} |
425   {Lookup.majorType == role})
426   {Token.string == "of"}
427 (
428   (BandN)
429 ):BandName
430 )--> :BandName.bandname = {kind = "Band", rule = "OfBand"}
431
432 Rule: BandReunion
433 (
434 (
435   (BandN)
436 ):BandName
437   (({Token.string == "cover"}{Token.string == "version"}) |
438   {Token.string == "reunion"})
439 )--> :BandName.bandname = {kind = "Band", rule = "BandReunion"}
440
441 Rule: Ex
442 (
443     ({Token.string == "ex"} | {Token.string == "Ex"})
444     {Token.string == "-"}
445 (
446   (BandN)
447 ):BandName
448 )--> :BandName.bandname = {kind = "Band", rule = "Ex"}
449
450 Rule: Ex2
```

```
(
    ({Token.string == "ex-"} | {Token.string == "Ex-"})
(
  (BandN)
):BandName
)--> :BandName.bandname = {kind = "Band", rule = "Ex2"}

Rule: TouredWith
(
  (({Token.string == "tour"} | {Token.string == "toured"} |
  {Token.string == "along"}) {Token.string == "with"})
(
  (BandN)
):BandName
  ({Token.string == ","} | {Token.string == "."} |
  {Token.string == "in"} | {Token.string == "on"})
)--> :BandName.bandname = {kind = "Band", rule = "TouredWith"}
```

B.3 SimpleListRules.jape

```
Phase: ListRules
Input: Token Lookup
Options: control = appelt

Macro: Entity
(
  {Token.orth == upperInitial}({Token.string != ",",
  Token.string != "and"})[0, 8]
)

Rule: BandHearst1
(
({Token.string == "bands"} | {Token.string == "acts"} |
{Token.string == "artists"} | {Token.string == "groups"})
{Token.string == "such"}
{Token.string == "as"}
((
(Entity)
({Token.string == ","}(Entity))[0,5]
{Token.string == ","}{Token.string == "and"}
)):mention
)
-->
{
List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
Collections.sort(annList, new OffsetComparator());
```

```
Node start = null;
Node end = null;
for(int i = 0; i < annList.size(); i++){
        Annotation anAnn = (Annotation)annList.get(i);
  if(anAnn.getFeatures().get("string").equals(",")){
    FeatureMap features = Factory.newFeatureMap();
          features.put("rule", "BandHearst1");
          annotations.add(start , end, "bandname", features);
    start = null;
  }else{
    if(start == null){
            start = anAnn.getStartNode();
    }
    end = anAnn.getEndNode();
  }
}
}

Rule: BandHearst2
(
({Token.string == "bands"} | {Token.string == "acts"} |
{Token.string == "groups"} | {Token.string == "artists"})
{Token.string == "like"}
((
(Entity)
({Token.string == ","}(Entity))[0,5]
{Token.string == ","}{Token.string == "and"}
)):mention
)
-->
{
List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
Collections.sort(annList, new OffsetComparator());
Node start = null;
Node end = null;
for(int i = 0; i < annList.size(); i++){
        Annotation anAnn = (Annotation)annList.get(i);
  if(anAnn.getFeatures().get("string").equals(",")){
    FeatureMap features = Factory.newFeatureMap();
          features.put("rule", "BandHearst2");
          annotations.add(start , end, "bandname", features);
    start = null;
  }else{
    if(start == null){
            start = anAnn.getStartNode();
    }
    end = anAnn.getEndNode();
```

```
74   }
75 }
76 }
77
78 Rule: BandHearst3
79 (
80 ({Token.string == "bands"} | {Token.string == "acts"} |
81 {Token.string == "groups"} | {Token.string == "artists"})
82 ({Token.string == ","})?
83 {Token.string == "including"}
84 ((
85 (Entity)
86 ({Token.string == ","}(Entity))[0,5]
87 {Token.string == ","}{Token.string == "and"}
88 )):mention
89 )
90 -->
91 {
92 List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
93 Collections.sort(annList, new OffsetComparator());
94 Node start = null;
95 Node end = null;
96 for(int i = 0; i < annList.size(); i++){
97       Annotation anAnn = (Annotation)annList.get(i);
98   if(anAnn.getFeatures().get("string").equals(",")){
99     FeatureMap features = Factory.newFeatureMap();
100          features.put("rule", "BandHearst3");
101          annotations.add(start , end, "bandname", features);
102    start = null;
103  }else{
104    if(start == null){
105          start = anAnn.getStartNode();
106    }
107    end = anAnn.getEndNode();
108  }
109 }
110 }
111
112 Rule: BandHearst4
113 (
114 {Token.string == "such"}
115 ({Token.string == "bands"} | {Token.string == "acts"} |
116 {Token.string == "groups"} | {Token.string == "artists"})
117 {Token.string == "as"}
118 ((
119 (Entity)
120 ({Token.string == ","}(Entity))[0,5]
```

```
121 {Token.string == ","}{Token.string == "and"}
122 )):mention
123 )
124 -->
125 {
126 List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
127 Collections.sort(annList, new OffsetComparator());
128 Node start = null;
129 Node end = null;
130 for(int i = 0; i < annList.size(); i++){
131       Annotation anAnn = (Annotation)annList.get(i);
132   if(anAnn.getFeatures().get("string").equals(",")){
133     FeatureMap features = Factory.newFeatureMap();
134           features.put("rule", "BandHearst4");
135           annotations.add(start , end, "bandname", features);
136     start = null;
137   }else{
138     if(start == null){
139           start = anAnn.getStartNode();
140     }
141     end = anAnn.getEndNode();
142   }
143 }
144 }
145
146 Rule: BandsFollow
147 (
148 ({Token.string == "bands"} | {Token.string == "acts"} |
149 {Token.string == "groups"} | {Token.string == "artists"})
150 ({Token.string == ","})?
151 ((
152 (Entity)
153 ({Token.string == ","}(Entity))[0,5]
154 {Token.string == ","}{Token.string == "and"}
155 )):mention
156 )
157 -->
158 {
159 List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
160 Collections.sort(annList, new OffsetComparator());
161 Node start = null;
162 Node end = null;
163 for(int i = 0; i < annList.size(); i++){
164       Annotation anAnn = (Annotation)annList.get(i);
165   if(anAnn.getFeatures().get("string").equals(",")){
166     FeatureMap features = Factory.newFeatureMap();
167           features.put("rule", "BandsFollow");
```

```
168             annotations.add(start , end, "bandname", features);
169       start = null;
170     }else{
171       if(start == null){
172               start = anAnn.getStartNode();
173       }
174       end = anAnn.getEndNode();
175     }
176 }
177 }
178
179 Rule: MediaList
180 (
181 ({Token.string == "albums"} | {Token.string == "songs"} |
182 {Token.string == "tracks"} | {Token.string == "singles"} |
183 {Token.string == "hits"} | {Token.string == "ballads"})
184 {Token.string == ","}
185 ((
186 (Entity)
187 ({Token.string == ","}(Entity))[0,5]
188 {Token.string == ","}{Token.string == "and"}
189 )):mention
190 )
191 -->
192 {
193 List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
194 Collections.sort(annList, new OffsetComparator());
195 Node start = null;
196 Node end = null;
197 for(int i = 0; i < annList.size(); i++){
198         Annotation anAnn = (Annotation)annList.get(i);
199   if(anAnn.getFeatures().get("string").equals(",")){
200     FeatureMap features = Factory.newFeatureMap();
201           features.put("rule", "MediaList");
202           annotations.add(start , end, "Media", features);
203     start = null;
204   }else{
205     if(start == null){
206           start = anAnn.getStartNode();
207     }
208     end = anAnn.getEndNode();
209   }
210 }
211 }
212
213 Rule: MediaHearst1
214 (
```

```
215 ({Token.string == "albums"} | {Token.string == "songs"} |
216 {Token.string == "tracks"} | {Token.string == "singles"} |
217 {Token.string == "hits"} | {Token.string == "ballads"})
218 ({Token.string == ","})?
219 ({Token.string == "like"} | {Token.string == "including"})
220 ((
221 (Entity)
222 ({Token.string == ","}(Entity))[0,5]
223 {Token.string == ","}{Token.string == "and"}
224 )):mention
225 )
226 -->
227 {
228 List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
229 Collections.sort(annList, new OffsetComparator());
230 Node start = null;
231 Node end = null;
232 for(int i = 0; i < annList.size(); i++){
233         Annotation anAnn = (Annotation)annList.get(i);
234   if(anAnn.getFeatures().get("string").equals(",")){
235     FeatureMap features = Factory.newFeatureMap();
236           features.put("rule", "MediaHearst1");
237           annotations.add(start , end, "Media", features);
238     start = null;
239   }else{
240     if(start == null){
241           start = anAnn.getStartNode();
242     }
243     end = anAnn.getEndNode();
244   }
245 }
246 }
247
248 Rule: MediaHearst2
249 (
250 ({Token.string == "albums"} | {Token.string == "songs"} |
251 {Token.string == "tracks"} | {Token.string == "singles"} |
252 {Token.string == "hits"} | {Token.string == "ballads"})
253 {Token.string == "such"}{Token.string == "as"}
254 ((
255 (Entity)
256 ({Token.string == ","}(Entity))[0,5]
257 {Token.string == ","}{Token.string == "and"}
258 )):mention
259 )
260 -->
261 {
```

```
List annList = new ArrayList((AnnotationSet)bindings.get("mention"));
Collections.sort(annList, new OffsetComparator());
Node start = null;
Node end = null;
for(int i = 0; i < annList.size(); i++){
        Annotation anAnn = (Annotation)annList.get(i);
  if(anAnn.getFeatures().get("string").equals(",")){
    FeatureMap features = Factory.newFeatureMap();
          features.put("rule", "MediaHearst2");
          annotations.add(start , end, "Media", features);
    start = null;
  }else{
    if(start == null){
            start = anAnn.getStartNode();
    }
    end = anAnn.getEndNode();
  }
}
}
```

B.4 SimpleMediaRules.jape

```

Phase: MediaRules
Input: Token Lookup Split
Options: control = appelt

Macro: Apostroph
(
  ({Token.orth == apostrophe} |
  {Token.category == POS, Token.kind == punctuation, Token.length == 1}) |
  ({Token.kind == punctuation, Token.length == 1, Token.string == "'"}
  ({Token.length == 1, Token.string == "s"})?)
)

Macro: Shortcut
(
  {Token.orth == upperInitial, Token.length <= 2}{Token.string == "."}
)

Macro: Start
(
  ({Token.orth == upperInitial} | Shortcut | {Token.orth == allCaps} |
  {Token.orth == mixedCaps} | {Token.category == CD, Token.kind == number,
  Token.length != 4})
)
```

```
25
26 Macro: End
27 (
28   ({Token.orth == upperInitial, !Token contains {Lookup.majorType == date}} |
29   Shortcut | {Token.orth == allCaps} | {Token.orth == mixedCaps} |
30   {Token.category == CD, Token.kind == number, Token.length != 4} |
31   {Token.string == "!"} | {Token.string == "?"})
32 )
33
34 Macro: StartSingle
35 (
36   ({Token.orth == upperInitial, Token.category != IN,
37   Token.category != DT, Token.category != PRP,
38   Token.category != JJ, Token.category != "PRP$", Token.category != JJS,
39   Token.category != RBR, Token.category != RBS,Token.category != JJR} |
40   {Token.orth == allCaps, Token.category != IN, Token.category != DT,
41   Token.category != PRP, Token.category != JJ, Token.category != "PRP$",
42   Token.category != JJS, Token.category != RBR, Token.category != RBS,
43   Token.category != JJR} | {Token.orth == mixedCaps, Token.category != IN,
44   Token.category != DT, Token.category != PRP, Token.category != JJ,
45   Token.category != "PRP$", Token.category != JJS, Token.category != RBR,
46   Token.category != RBS, Token.category != JJR})
47 )
48
49 Macro: Follow
50 (
51   ( {Token.category == EX, Token.orth == upperInitial} |
52   {Token.category == VB, Token.orth == upperInitial} |
53   {Token.category == VBN, Token.orth == upperInitial} |
54   {Token.category == VBG, Token.orth == upperInitial} |
55   {Token.category == VBD, Token.orth == upperInitial} |
56   {Token.category == WRB, Token.orth == upperInitial} |
57   {Token.category == VBP, Token.orth == upperInitial} |
58   {Token.category == VBZ, Token.orth == upperInitial} |
59   {Token.category == WP, Token.orth == upperInitial} |
60   {Token.category == MD, Token.orth == upperInitial} |
61   {Token.category == TO} | {Token.category == DT} |
62   {Token.category == POS} | {Token.orth == apostrophe} |
63   {Token.category == CD} | {Token.category == RBR} |
64   {Token.category == RB}  | {Token.category == NNP,
65   Token.orth == upperInitial,
66   !Token contains {Lookup.majorType == date}} |
67   {Token.category == NNS, Token.orth == upperInitial} |
68   {Token.category == NNPS, Token.orth == upperInitial} |
69   {Token.category == IN} | {Token.category == NN,
70   Token.orth == upperInitial} | {Token.category == JJR,
71   Token.orth == upperInitial} | {Token.category == JJS,
```

```
72   Token.orth == upperInitial} | {Token.category == JJ} |
73   {Token.category == PRP} | {Token.category == "PRP$"} |
74   {Token.string == "&"} | {Token.string == "/"}  |
75   {Token.string == "And"} | {Token.string == "Or"} |
76   Shortcut | {Token.string == ":"})
77 )
78
79 Macro: Medium
80 (
81   (Start (Follow)+ End)  | (Start End) | (StartSingle)
82 )
83
84 Rule: MediaYear
85 (
86   ({Token.string == "\""})?
87   (
88     (Medium)
89   ):Media
90   ({Token.string == "\""})?
91   {Token.string == "("}
92   {Lookup.majorType == year}
93   {Token.string == ")"}
94 )--> :Media.Media = {kind = "Media", rule = "MediaYear"}
95
96 Rule: MediaComma
97 (
98   ({Token.string == "album"} | {Token.string == "demo"} |
99   {Token.string == "debut"} | {Token.string == "effort"} |
100  {Token.string == "single"} | {Token.string == "record"} |
101  {Token.string == "release"} | {Token.string == "track"} |
102  {Token.string == "song"} | {Token.string == "hit"} |
103  {Token.string == "follow-up"} | {Token.string == "LP"} |
104  {Token.string == "EP"} | {Token.string == "CD"} |
105  {Token.string == "DVD"} | {Token.string == "full-length"} |
106  {Token.string == "video"} | {Token.string == "collection"})
107  ({Token.string == ","} | {Token.string == ":"})
108  (
109    (Medium)
110  ):Media
111 )--> :Media.Media = {kind = "Media", rule = "MediaComma"}
112
113 Rule: Released
114 (
115  ({Token.string == "released"} | {Token.string == "recorded"} |
116  {Token.string == "issued"})
117  ({Token.string == "\""})?
118  (
```

```
119     (Medium)
120   ):Media
121   ({Token.string == "\""})?
122 )--> :Media.Media = {kind = "Media", rule = "Released"}
123
124 Rule: HitWith
125 (
126   ({Token.string == "hit"} | {Token.string == "hits"} |
127   {Token.string == "ballads"} | {Token.string == "ballad"} |
128   {Token.string == "single"})
129   {Token.string == "with"}
130   ({Token.string == "\""})?
131   (
132     (Medium)
133   ):Media
134   ({Token.string == "\""})?
135 )--> :Media.Media = {kind = "Media", rule = "HitWith"}
136
137 Rule: FollowedBy
138 (
139   ({Token.string == "followed"} | {Token.string == "preceded"})
140   {Token.string == "by"}
141   ({Token.string == "\""})?
142   (
143     (Medium)
144   ):Media
145   ({Token.string == "\""})?
146 )--> :Media.Media = {kind = "Media", rule = "FollowedBy"}
147
148 Rule: Followed
149 (
150   ({Token.string == "\""})?
151   (
152     (Medium)
153   ):Media
154   ({Token.string == "\""})?
155   {Token.string == "followed"}
156 )--> :Media.Media = {kind = "Media", rule = "Followed"}
157
158 Rule: MediaPassivReleased
159 (
160   ({Token.string == "\""})?
161   (
162     (Medium)
163   ):Media
164   ({Token.string == "\""})?
165   ({Token.string == "was"} | {Token.string == "was"} |
```

```
166   ({Token.string == "will"}{Token.string == "be"}))
167   ({Token.string == "released"} | {Token.string == "issued"} |
168   {Token.string == "produced"} | {Token.string == "recorded"} |
169   {Token.string == "played"} | {Token.string == "performed"} )
170 )--> :Media.Media = {kind = "Media", rule = "MediaPassivReleased"}
171
172 Rule: MediaFollow
173 (
174   ({Token.string == "album"} | {Token.string == "demo"} |
175   {Token.string == "full-length"} | {Token.string == "effort"} |
176   {Token.string == "single"} | {Token.string == "ballad"} |
177   {Token.string == "video"} |  {Token.string == "record"} |
178   {Token.string == "release"}  | {Token.string == "track"} |
179   {Token.string == "song"} | {Token.string == "hit"} |
180   {Token.string == "follow-up"} | {Token.string == "LP"} |
181   {Token.string == "EP"} | {Token.string == "CD"}  |
182   {Token.string == "DVD"})
183   (
184     (Medium)
185   ):Media
186 )--> :Media.Media = {kind = "Media", rule = "MediaFollow"}
187
188 Rule: verbMedia
189 (
190   ({Token.string == "covered"} | {Token.string == "performed"} |
191   {Token.string == "played"} | {Token.string == "singed"})
192   ({Token.string == "\""})?
193   (
194     (Medium)
195   ):Media
196   ({Token.string == "\""})?
197 )--> :Media.Media = {kind = "Media", rule = "verbMedia"}
198
199 Rule: YearsAlbum
200 (
201   {Lookup.majorType == year}
202   (Apostroph)
203   (
204     (Medium)
205   ):Media
206 )--> :Media.Media = {kind = "Media", rule = "YearsAlbum"}
207
208 Rule: AlbumTitled
209 (
210   ({Token.string == "album"} | {Token.string == "single"} |
211   {Token.string == "record"} | {Token.string == "track"} |
212   {Token.string == "song"} | {Token.string == "EP"} |
```

```
213   {Token.string == "CD"} | {Token.string == "DVD"} |
214   {Token.string == "compilation"} | {Token.string == "video"} |
215   {Token.string == "collection"})
216   ({Token.string == ","})?
217   ({Token.string == "titled"} | {Token.string == "entitled"} |
218   {Token.string == "named"} | {Token.string == "called"})
219   (
220     (Medium)
221   ):Media
222 )--> :Media.Media = {kind = "Media", rule = "AlbumTitled"}
223
224 Rule: ofMedia
225 (
226 ({Token.string == "release"} | {Token.string == "riff"} |
227 {Token.string == "performance"} | {Token.string == "recording"} |
228 {Token.string == "remix"} | {Token.string == "chorus"} |
229 {Token.string == "lyrics"} | {Token.string == "sound"} |
230 {Token.string == "version"} | {Token.string == "cover"} |
231 {Token.string == "success"} | {Token.string == "medley"})
232 {Token.string == "of"}
233 (
234   (Medium)
235 ):Media
236 ({Token.orth != apostrophe} | {Token.category != POS})
237 )--> :Media.Media = {kind = "Media", rule = "ofMedia"}
238
239 Rule: which
240 (
241   ({Token.string == "\""})?
242   (
243     (Medium)
244   ):Media
245   ({Token.string == "\""})?
246   ({Token.string == ","})?
247   ({Token.string == "which"})?
248   ({Token.string == "debuted"} | {Token.string == "sold"} |
249   {Token.string == "charted"} | {Token.string == "hit"} |
250   ({Token.string == "was"}{Token.string == "released"}))
251 )--> :Media.Media = {kind = "Media", rule = "which"}
252
253 Rule: followUpTo
254 (
255   {Token.string == "follow-up"}
256   {Token.string == "to"}
257   ({Token.string == "\""})?
258   (
259     (Medium)
```

```
260   ):Media
261   ({Token.string == "\""})?
262 )--> :Media.Media = {kind = "Media", rule = "followUpTo"}
263
264 Rule: Video
265 (
266   ({Token.string == "video"} | {Token.string == "videos"} |
267   {Token.string == "videoclip"})
268   {Token.string == "for"}
269   (
270     (Medium)
271   ):Media
272 )--> :Media.Media = {kind = "Media", rule = "Video"}
```

B.5 SimpleMemberRules.jape

```
1
2 Phase:  Time
3 Input: Token Lookup Split
4 Options: control = appelt
5
6 Macro: Apostroph
7 (
8   ({Token.orth == apostrophe} |
9   {Token.category == POS, Token.kind == punctuation, Token.length == 1}) |
10  ({Token.kind == punctuation, Token.length == 1, Token.string == "'"}
11  ({Token.length == 1, Token.string == "s"})?)
12 )
13
14 Macro: Shortcut
15 (
16  {Token.orth == upperInitial, Token.length <= 2}{Token.string == "."}
17 )
18
19 Macro: MemberName
20 (
21  ({Token.category == NNP,
22  !Token contains {Lookup.majorType == role}} | Shortcut)
23  (
24    ({Token.string == "\""}({Token}({Token})?
25    {Token.string == "\""}) |
26
27    ({Token.orth == upperInitial, Token.length <= 2}
28    (Apostroph)) |Shortcut |
29    ({Token.string == "von"} | {Token.string == "van"})
30
```

```
31    )?
32    ({Token.orth == upperInitial} | {Token.orth == mixedCaps} |
33    {Token.orth == allCaps} | Shortcut)[1,3]
34  )
35
36  Rule: Delim
37  (
38    (
39      (MemberName)
40    ):BandMember
41    ({Token.category == ":"} | {Token.category == NN, Token.length == 1})
42    ({Lookup.majorType == instrument} | {Lookup.majorType == role})
43  )--> :BandMember.Member = {kind = "BandMember", rule = "Delim"}
44
45  Rule: leftJoinedBand
46  (
47    (
48      (MemberName)
49    ):BandMember
50    ({Token.string == "had"} | {Token.string == "has"})?
51    ({Token.string == "left"} | {Token.string == "joined"} |
52    {Token.string == "rejoined"} | {Token.string == "replaced"})
53  )--> :BandMember.Member = {kind = "BandMember", rule = "leftJoinedBand"}
54
55  Rule: ProfessionMember
56  (
57    ({Lookup.majorType == role} | {Token.string == "member"} |
58    {Token.string == "members"})
59    ({Token.string  == ","})?
60    (
61      (MemberName)
62    ):BandMember
63  )--> :BandMember.Member = {kind = "BandMember", rule = "ProfessionMember"}
64
65  Rule: ProfessionBrackets
66  (
67    (
68      (MemberName)
69    ):BandMember
70    {Token.string == "("}
71    {Lookup.majorType == instrument}
72    ({Token.string == ","}{Lookup.majorType == instrument})?
73    ({Token.string == ","}{Lookup.majorType == instrument})?
74    (({Token.string == "and"} | {Token.string == "&"} |
75    {Token.string == "or"}){Lookup.majorType == instrument})?
76    {Token.string == ")"}
77  )--> :BandMember.Member = {kind = "BandMember", rule = "ProfessionBrackets"}
```

```
78
79 Rule: ProfessionSlash
80 (
81   (
82     (MemberName)
83   ):BandMember
84   {Token.string == "("}
85   {Lookup.majorType == instrument}
86   (
87     ({Token.string == "/"}{Lookup.majorType == instrument})
88     ({Token.string == "/"}{Lookup.majorType == instrument})?
89     ({Token.string == "/"}{Lookup.majorType == instrument})?
90   )
91   {Token.string == ")"}
92 )--> :BandMember.Member = {kind = "BandMember", rule = "ProfessionSlash"}
93
94 Rule: AposMember
95 (
96   (
97     (MemberName)
98   ):BandMember
99   (Apostroph)
100  ({Token.category == JJ} | {Token.category == NN} |
101  {Token.category == VB} | {Token.category == VBG})?
102  ({Token.string  == "voice"} |{Token.string  == "basslines"} |
103  {Token.string  == "drumming"} | {Token.string  == "chords"} |
104  {Lookup.majorType == instrument})
105 )--> :BandMember.Member = {kind = "BandMember", rule = "AposMember"}
106
107 Rule: MemberOn
108 (
109   (
110     (MemberName)
111   ):BandMember
112   {Token.string  == "on"}
113   {Lookup.majorType == instrument}
114 )--> :BandMember.Member = {kind = "BandMember", rule = "MemberOn"}
115
116 Rule: replacedBy
117 (
118   {Token.string  == "replaced"}
119   ({Token.string  == "by"} | {Token.string  == "with"})
120   (
121     (MemberName)
122   ):BandMember
123 )--> :BandMember.Member = {kind = "BandMember", rule = "replacedBy"}
124
```

```
125 Rule: consistedOf
126 (
127   ({Token.string == "consisted"} | {Token.string == "consists"} |
128   {Token.string == "comprised"} | {Token.string == "consisting"} |
129   {Token.string == "departure"})
130   {Token.string == "of"}
131   (
132     (MemberName)
133   ):BandMember
134 )--> :BandMember.Member = {kind = "BandMember", rule = "consistedOf"}
135
136 Rule: MemberAs
137 (
138   (
139     (MemberName)
140   ):BandMember
141   ({Token.string == "as"} | {Token.string == "became"})
142   {Lookup.majorType == role}
143 )--> :BandMember.Member = {kind = "BandMember", rule = "MemberAs"}
144
145 Rule: formedBy
146 (
147   ({Token.string == "formed"} | {Token.string == "founded"} |
148   {Token.string == "co-founded"})
149   {Token.string == "by"}
150   (
151     (MemberName)
152   ):BandMember
153 )--> :BandMember.Member = {kind = "BandMember", rule = "formedBy"}
154
155 Rule: CommaRole
156 (
157   (
158     (MemberName)
159   ):BandMember
160   {Token.string == ","}
161   ({Lookup.majorType == role} | {Token.string == "member"})
162 )--> :BandMember.Member = {kind = "BandMember", rule = "CommaRole"}
163
164 Rule: Verb
165 (
166   (
167     (MemberName)
168   ):BandMember
169   ({Token.string == "played"} | {Token.string == "performed"} |
170   {Token.string == "sang"})
171 )--> :BandMember.Member = {kind = "BandMember", rule = "Verb"}
```

```
172
173 Rule: Verb
174 (
175   (
176     (MemberName)
177   ):BandMember
178   ({Token.string == ","})?
179   {Token.string == "formerly"}
180   {Token.string == "of"}
181 )--> :BandMember.Member = {kind = "BandMember", rule = "Verb"}
182
183 Rule: including
184
185 (
186   ({Token.string == "performers"} | {Token.string == "musicians"})
187   {Token.string == "including"}
188   (
189     (MemberName)
190   ):BandMember
191 )--> :BandMember.Member = {kind = "BandMember", rule = "including"}
```