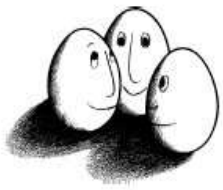
Diplomarbeit

Eine Umgebung zur Informationsextraktion aus Geschäftsbriefen

Benjamin Helbig



Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

02. August 2005

Betreuer:

Prof. Dr. Katharina Morik
Prof. Dr. Wolfgang Hoeppner
M.A. Marc Rössler

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der heutigen Geschäftswelt ist es im produzierenden Gewerbe essentiell, schnell auf Anfragen seiner Kunden zu reagieren. Außer im unwahrscheinlichen Fall, dass man eine Monopolstellung genießt, wird es eine Vielzahl von Mitbewerbern geben, die ebenfalls um einen Kostenvoranschlag gebeten werden. Derjenige, dessen Antwort am wenigsten auf sich warten lässt, hat unbestritten einen gewissen Vorteil. Das Auffinden von relevanten Eckdaten in der Kundenanfrage ist dabei ein erster, wichtiger Schritt zur Erstellung eines Angebots. Für einen Geschäftsbrief mit einem Umfang von vier Seiten, ist das sicherlich eine schnell zu bewerkstelligende Arbeit, schließlich kann man den Inhalt fast auf einen Blick erfassen. Schwieriger und vor allem langwieriger gestaltet sich die Aufgabe jedoch bei einer Seitenzahl von hundert oder noch mehr Seiten. Hier gerät die Suche zu einem fortwährenden Prozess des Herausfilterns der zuhauf vorhandenen Daten, die für diesen Arbeitsgang überflüssig sind.
Diese Situation lässt sich beispielsweise mit der des Internets vergleichen. Mit dessen rasanten Verbreitung stehen immer mehr Texte on-line zur Verfügung, womit es immer schwieriger wird, relevante Informationen zu finden, zu extrahieren und in kompakter Form zu repräsentieren.
Um die Informationsüberflutung adäquat meistern zu können, wird bereits fieberhaft nach neuen Technologien für zukünftige intelligente Informationsmanagementsysteme geforscht. Eine sich neu etablierte Forschungsrichtung ist die Erforschung und Realisierung von Systemen zur *Informationsextraktion* (IE). Das Ziel der IE ist die Konstruktion von Systemen, die gezielt domänenspezifische Informationen aus freien Texten aufspüren und strukturieren können, bei gleichzeitigem "Überlesen" irrelevanter Information. IE-Systeme versuchen keine umfassende Analyse des gesamten Inhalts aller Textdokumente, sondern sollen nur die Textpassagen analysieren bzw. "verstehen", die relevante Information beinhalten. Was als relevant gilt, wird

dabei durch vordefinierte domänenspezifische Lexikoneinträge bzw. Regeln dem System fest vorgegeben oder aber anhand von Beispielen gelernt. Dieses Wissen muss dabei so detailiert und genau wie möglich festlegen, welche Typen von Information von einem IE-System extrahiert werden soll, damit eine umfangreiche und zugleich präzise Extraktion möglich wird.
Typischerweise modelliert die vorgegebene Information komplexe, zusammenhängende Antwortmuster bezüglich *wer, was, wem, wann, wo, warum* und *wieviel*. Sie werden in Form von *Templates* spezifiziert, also verbundartigen Strukturen in Form von Merkmal/Wert-Paaren, z.B. Firmen- und Produktinformationen, Umsatzmeldungen, Personalwechsel, Stellenausschreibungen. Die Kernfunktionalität eines IE-Systems lässt sich dann kurz wie folgt charakterisieren:

- Eingabe: Spezifikation des Typs der relevanten Information in Form von Templates (Menge von Merkmalen) und eine Menge von freien Textdokumenten (Pressemitteilungen, Internet-Dokumente etc.)
- Ausgabe: eine Menge von instanziierten Templates (Werte für Merkmale), die mit den als relevant identifizierten und normalisierten Textfragmenten gefüllt sind.

(aus [1])

Diese Diplomarbeit widmet sich dem anfangs beschriebenen Problem der Informationsextraktion aus Geschäftsbriefen.
Angestoßen von einem Partner aus der Industrie sollte ein System entwickelt werden, das das Auswerten einer Kundenanfrage unterstützt und es dadurch schneller und effizienter macht. Dieser Vorgang wurde bis dahin ohne oder nur teilweise mit der Hilfe eines Computers ausgeführt. Der Arbeitsprozess sieht also so aus, dass ein Mitarbeiter sich den Geschäftsbrief durchliest und, sobald er dabei auf eine seinem Empfinden nach relevante Information stößt, diese, mit einem entsprechenden Bezeichner versehen, notiert. So findet er beispielsweise folgende Textstelle:

Betriebsbedingungen

Arbeitsdruck p_e	bar	-1 / 0,1
Arbeitstemperatur	°C	250

Daraus resultieren die Einträge:
`Temperatur: 250 °C`,

`Druck: -1 / 0,1 bar`,
die die Fachkraft entweder auf einem Blatt Papier oder in einer ggf. formatierten Textdatei festhält. Weiter unten auf derselben Seite könnte sich weiterhin ein Absatz befinden, der aussagt, dass eine Dokumentation in fünf Sprachen ausgeführt werden soll. Diesen Wunsch überliest er jedoch, da dies nicht in sein Aufgabengebiet gehört; die Information könnte aber für einen anderen Mitarbeiter außerordentlich von Belang sein. Danach fährt er mit der Suche fort und bringt schlussendlich all seine Funde in eine saubere Form zum Zwecke der Weiterverarbeitung.
Um diese Vorgänge komplett zu automatisieren ließe sich ein oder gleich mehrere Lerner einsetzen, die mit Hilfe von Beispielen, d.h. von getroffenen Entscheidungen eines Mitarbeiters, trainiert werden. Das Problem hieran ist, dass (noch) kein Lerner für sich beanspruchen kann, zu hundert Prozent korrekte Resultate zu liefern, was bei der Wichtigkeit dieser Daten aber vonnöten wäre. Außerdem müsste bestenfalls eine Unmenge an Beispielen zur Verfügung gestellt werden, da Lerner zumeist umso besser arbeiten, je mehr Trainingsinstanzen sie erhalten.
Der in dieser Diplomarbeit verfolgte Ansatz geht deshalb einen anderen Weg. Der Benutzer soll ständig am Prozess der Informationsextraktion beteiligt sein, da er schließlich wertvolles Expertenwissen liefert.
Um dies umzusetzen, soll das System ihm zunächst eine Oberfläche bieten, in der er sich das Dokument leicht durchsehen und entscheidende Daten schnell in strukturierter Form erfassen kann. Das bedeutet, dass ein Transfer seiner Tätigkeit auf die Computerebene vollzogen wird. Dafür wird der Geschäftsbrief durch OCR zuvor in ein HTML-Dokument umgewandelt; diese Umwandlung liegt außerhalb dieser Diplomarbeit und wird mit einem beliebigen Programm vollzogen, es wird aber davon ausgegangen, dass sie stattgefunden hat.
Durch die Arbeit des Anwenders soll das System nun lernen, wo sich wichtige Werte im Dokument befinden und welchem Attribut sie zuzuordnen sind. Da es sich hierbei weitgehend um technische Daten handelt, liegt der Schluss nahe, dass sie größtenteils in einer Tabelle oder zumindest in einer tabellenähnlichen Struktur statt in Fließtext zu finden sind. Diese Annahme wird auch von den vom Partner zur Verfügung gestellten Beispieldokumenten gestützt. Deswegen soll sich das Lernen auf derartige Strukturen konzentrieren.
Als Ergebnis sollen dem Benutzer Vorschläge unterbreitet werden, wo weitere relevante Daten stehen. Diese Voschläge soll er leicht annehmen oder auch verwerfen können. Dadurch hat der Anwender stets die komplette Kontrolle, da er ja schließlich auch bei seinem Arbeitgeber in Verantwortung steht, soll dabei jedoch eine breite Unterstützung erfahren.
Die Anforderungen an dieses System, das PROMETHEUS getauft wurde, sind

somit die folgenden:

- Es soll ein Template geben, dessen Attribute der Benutzer selbst bestimmen kann. So soll gewährleistet werden, dass jeder Mitarbeiter PROMETHEUS exakt auf sein Arbeitsressort ausrichten kann.
- Dieses Template soll in Form eines Formulars grafisch präsentiert werden und über diese Benutzerobefläche modifiziert werden können.
- Das (HTML-)Dokument soll ebenfalls angezeigt werden. In dieser Anzeige soll der Benutzer die Möglichkeit haben, relevante Informationen zu markieren und auf einfache Weise in das Formular zu übernehmen.
- Der Benutzer soll durch eine semi-automatische Auszeichnungsmethode unterstützt werden. Diese soll durch das Verhalten des Anwenders inkrementell lernen und aufgrund dessen Vorschläge unterbreiten.
- Um dem Benutzer Zeit bei der Suche zu sparen, sollen diese Vorschläge eher erschöpfend als genau sein.
- Das Verfahren soll die Art und Weise, wie Werte zu ihren Attributen stehen, erfassen.
- Die (Zwischen-)Ergebnisse der Informationsextraktion sollen nicht nur abgespeichert werden können, sondern auch für eine Weiterverarbeitung außerhalb von PROMETHEUS verfügbar gemacht werden.

Diese Punkte sind wünschenswert:

- Ein weiteres Auszeichnungsverfahren sollte zur Verfügung stehen, das lediglich auf Zeichenketten beruht. Auf diese Weise sollen dem Anwender ungeachtet der Struktur noch mehr Vorschläge unterbreitet werden.
- Eine robuste Suchfunktion soll das Blättern zu bedeutenden Textstellen erleichtern.
- Damit Fehler der OCR nicht allzu schwer ins Gewicht fallen, soll die Anzeige des Dokuments vor OCR direkt in PROMETHEUS möglich sein.

Insgesamt soll also eine Umgebung zur interaktiven Informationsextraktion geschaffen werden.

Aufbau

Zum Abschluss dieses einleitenden Abschnitts soll kurz die Gliederung dieser Diplomarbeit vorgestellt werden.
Kapitel 2 befasst sich mit zu PROMETHEUS verwandten Ansätzen und arbeitet Gemeinsamkeiten und Unterschiede heraus.
Das darauffolgende Kapitel 3 führt einige Grundbegriffe ein und vermittelt Hintergrundwissen, das für das Verständnis der anschließenden Ausführungen benötigt wird.
Den Kern dieser Arbeit bildet das vierte Kapitel. Es beschreibt detailliert die Komponenten des Systems sowie ihre Funktionsweise.
Der anschließende beispielhafte Arbeitsvorgang zeigt den praktischen Einsatz von PROMETHEUS anhand eines realen Dokumentes.
Eine Evaluierung der Auszeichnungsverfahren aus Kapitel 4.7 und 4.8 wird in Kapitel 6 durchgeführt.
Abschließend fasst Kapitel 7 die Ergebnisse dieser Diplomarbeit zusammen und stellt Ideen für Weiterentwicklungen vor.
Im Anhang befindet sich eine Art Werdegang, den PROMETHEUS hinter sich hat.

Kapitel 2

Abgrenzung zu bestehenden Ansätzen

Dieser Abschnitt stellt einige existierende Ansätze vor und vergleicht diese mit Komponenten aus PROMETHEUS, oder klärt ggf., warum diese nicht in PROMETHEUS verwendet wurden.

2.1 Informationsextraktion nach MUC

Die *Message Understanding Conferences* (MUC) wurden von der DARPA initiiert und finanziert mit dem Ziel die Entwicklung neuer und besserer Methoden der Informationsextraktion zu fördern. Dazu werden Aufgaben, wie das Auffinden von Daten zu bestimmten Themengebieten wie Flugzeugabstürze (MUC-7) oder terroristische Aktivitäten (MUC-3, MUC-4), gestellt.[2] Da festgestellt wurde, dass es die IE nach vorn gebracht hat, in einem ersten Schritt z.B. Koreferenzen aufzulösen, sowie wichtige Entitäten zu erkennen und ihnen semantische Kategorien zuzuordnen, bevor die eigentlichen Relationen wie "Bombe" für ein "Waffen"-Attribut ermittelt werden, werden diese vorverarbeitenden Maßnahmen ab MUC-6 sogar als isolierte Aufgaben angeboten und als integrale Bestandteile eines IE-Systems verstanden; zumindest sind die Anforderungen der Informationsextraktionsaufgaben derart ausgelegt[3, 4].
Besonders die *Named Entity Recognition*, kurz *NER*, ist ein Bereich, in dem ein großer Teil der Forschung innerhalb der Computerlinguistik betrieben wird. Es geht dabei um die Erkennung und Klassifizierung von Eigennamen. Welche Elemente als Eigennamen zählen und welche Faktoren diese Kategorien definieren, ist nicht immer klar und hängt von der Anwendung ab. Zum Beispiel unterteilt die *MUC-7 Named Entity Task* das Unterfan-

gen in drei Teilaufgaben, nämlich Eigennamen, temporäre und zahlenbezogene Ausdrücke. Dabei sollen die Ausdrücke, die Eigennamen darstellen, mit den eindeutigen Bezeichnern `Organisation`, `Person` oder `Ort` gekennzeichnet werden. Bei den zeitlichen Ausdrücken sollen es `Datum` oder `Zeit` und bei den numerischen entweder `Geld-Wert` oder `Prozentzahl` sein [5].
Die Ergebnisse können, wie gesagt, bei der Informationsextraktion, aber auch beim Information Retrieval, der Bestimmung des Themas, Zusammenfassungen und anderen sprachbasierten Anwendungen eingesetzt werden.
Die Ansätze bei der NER sind vielfältig und reichen von regelbasierten Systemen über Wortlisten bis hin zu verschiedenen statistischen und maschinellen Lernverfahren. Diese sind meist sehr domänenspezifisch und lassen sich somit erst nach einem Adaptierungsprozess auf andere anwenden. Die Verfahren arbeiten hauptsächlich mit sprachbezogenen und kontextsensitiven Merkmalen. So benutzen (jeweils u.a.) *Chen, Ding, Tsai* und *Bian* Hinweise aus dem Kontext wie Anreden, Speech-Act Worte, Positionen etc.[6]; Guo-Dong verwendet in [7] Attribute, die auf morphologischen Informationen und Part-of-Speech(POS) Tagger beruhen; *Mohit* und *Hwa* schlagen syntaktische Merkmale vor[8].
Rössler beschreibt in [9] einen wissensarmen Ansatz für die Named Entity Recognition im Deutschen. Um leicht auf andere Domänen übertragbar zu sein, wird hier vollkommen auf sprachbezogene Werkzeuge wie die morphologische Analyse, POS Speech Tagger oder syntaktische Chunker verzichtet. Stattdessen werden u.a. n-Gramme der umgebenen Wörter als Features für eine SVM verwendet. Dabei wird ein Fenster von 6 Worten gewählt, so dass der Kontext ausreichend berücksichtigt wird.
Gerade ihre Berücksichtigung des textuellen Kontexts und ihre teilweise Verbundenheit zu natürlichsprachlichen Äußerungen, d.h. Freitext, machen viele Ansätze der NER für PROMETHEUS unattraktiv. Die Daten und Bezeichner, die für PROMETHEUS relevant sind, sind zu speziell und werden wohl nur in geringer Zahl in Fließtexten, d.h. in Satzkonstruktionen auftauchen, sondern eher in Tabellen oder Listen, bei denen der Kontext oder so etwas wie morphologische Struktur eher zweitrangig ist. Auch sind die Ansätze zumeist voll von dömanen- und sprachspezifischen Wissen, was die Möglichkeiten einer Übertragung sehr einengt.
Ohnehin wäre eine aufwändige Named Entity Recognition nur interessant gewesen, falls PROMETHEUS die komplette Auszeichnung eines Dokuments ohne Beteiligung des Benutzers zugefallen wäre. Dann hätte die NER, als Vorstufe der Informationsextraktion, ggf. mögliche Merkmalsbezeichner ausfindig machen können. So wird in der derzeitigen Version von PROMETHEUS darauf verzichtet, mehrstufig vorzugehen, stattdessen wird direkt auf der Relationsebene gearbeitet.

Wrapper	*Einsatzgebiet*
`LR`	der einfachste Wrapper für tabellarische Ressourcen, besteht aus Beschränkungen, die links und rechts vom Attribut stehen
`HLRT`	s.u.
`OCLR`	eine Wrapper-Klasse die sowohl aus Beschränkungen, die links und rechts vom Attribut stehen, als auch aus Beschränkungen, die das Tupel öffnen und schließen, besteht
`HOCLRT`	eine Wrapper-Klasse, die die Funktionalitäten von `HLRT` und `OCLR` vereint.
`N-LR`	die einfachste Wrapper-Klasse für verschachtelte Ressourcen bestehend aus Beschränkungen, die links und rechts vom Attribut stehen
`N-HRLT`	die Erweiterung von `HLRT`

Tabelle 2.1: Wrapper Klassen [12]

Dennoch wurden Teile der Auszeichnungskomponenten von PROMETHEUS von Arbeiten im Bereich der NER inspiriert. So gibt es neben Wortlisten bestehend aus *Eintragsbezeichnern* und *Synonymen* (s.u.) auch Konstrukte, die Ideen wie den 'Lexical Features' aus [10] nicht unähnlich sind. Bei diesen wird beispielsweise eine Person dadurch gefunden, dass ihr das Wort 'Mr' vorausgeht.

2.2 Wrapper Induction

Im Internet kursieren viele Quellen, die relationale Daten liefern. So gibt es z.B. Anfrage-Systeme, die bei einem übergebenen Namen, Paare der Form $\langle Name, Email \rangle$ liefern. Da diese Antwort zunächst für das Auge des Menschen angenehm in HTML aufbereitet wird, sind diese Daten erstmal nicht ohne weiteres in einem Programm zu verwenden. Um an sie zu gelangen, muss mühsam ein so genannter Wrapper geschrieben werden, der die relevanten Informationen herausfiltert. Da handgeschriebene Wrapper zeitaufwändig und gemeinhin fehlerbehaftet sind, schlagen *Kushmerick, Weld* und *Doorenbos* in [11] vor, diese automatisch lernen zu lassen. Dabei konzentriert sich dieser Ansatz hauptsächlich auf HTML-Seiten, die von einem Anfrage-System generiert worden sind und die besagte Relationen in Tabellenform darstellen. Um den verschiedenen möglichen Realisierungen von Tabellen Herr zu werden, werden in [12] verschiedene Klassen von Wrappern kurz vorgestellt, die in Tabelle 2.1 festgehalten sind. Es wird


```
<HTML><TITLE>Some Country Codes</TITLE>
<BODY><B>Some Country Codes</B><P>
<B>Congo</B> <I>242</I><BR>
<B>Egypt</B> <I>20</I><BR>
<B>Belize</B> <I>501</I><BR>
<B>Spain</B> <I>34</I><BR>
<HR><B>End</B></BODY></HTML>
```

Abbildung 2.1: Fiktive Antwort-Seite einer Anfrage

sich jedoch hauptsächlich auf `HLRT` (head-left-right-tail) konzentriert. Diese Wrapper durchsuchen ihre Eingabe nach Teilstrings, die die Informationen begrenzen, die extrahiert werden sollen. Diese sollen als Tupel gefunden werden wie eben z.B. $\langle Name, Email \rangle$. Formell wird ein `HLRT` Wrapper für eine Domäne mit K Attributen pro Tupel als Vektor mit $2K + 2$ Strings gespeichert:
$\langle h, t, l_1, r_1, \ldots, l_K, r_K \rangle$
Eine Zeichenkette (h) markiert das Ende des Headers der Seite, eine andere (t) markiert den Start des Abschlusses und zwei Strings (l_K) und (r_K) begrenzen jeweils eins der (K) Attribute.
Zu beachten ist, dass dieser Ansatz gar nicht auf HTML angewiesen ist, da er ausschließlich auf Zeichenketten arbeitet.
Für das Beispiel in Abbildung 2.1 hätte der Vektor also die Form ⟨`<P>`, `<HR>`, `<B>`, `</B>`, `<I>`, `</I>` ⟩. Um die Daten nun zu extrahieren wird bei dieser Seite zunächst bis zum ersten Auftreten von <P> gesprungen. Daraufhin werden solange jeweils die Werte, die erst zwischen <B>

und </B> und dann zwischen <I>, </I> liegen, extrahiert und als Tupel abgelegt, bis <HR> erscheint. Als Resultat erhält man {⟨Congo,242⟩, ⟨Egypt, 20⟩, ⟨Belize, 501⟩, ⟨Spain, 34⟩}.

Um einen solchen Wrapper zu generieren, wird eine Methode, `BuildHLRT`, solange mit neuen ausgezeichneten Beispielseiten aufgerufen, bis eine PAC Analyse besagt, dass genug Beispiele gesehen wurden, dass ein zufriedenstellender Wrapper gelernt worden ist. In der Methode `BuildHLRT` wird über alle Möglichkeiten für die $2K + 2$ Begrenzer iteriert, bis ein mit *allen* gesehenen Beispielseiten konsistenter Wrapper gefunden worden ist.
Die Beispielseiten und deren Auszeichnungen sollen dabei idealerweise auch automatisch erzeugt werden. Bei den Beispielseiten ist dies kein großes Problem: man sendet einfach Anfragen an die Informationsquelle, deren Antwortseiten man untersuchen will.
Die automatischen Auszeichnungen sollen mit dem so genannten *Corroboration* Algorithmus vorgenommen werden. Dieser arbeitet mit *Recognizers*, die jeweils Instanzen eines bestimmten Attributs finden sollen. So gibt es bei einer Seite, die Informationen über ein Land, dessen Ländercode und dessen Hauptstadt beinhaltet, einen Recognizer für Länder, einen für Ländercodes und einen für Hauptstädte. Die Funde werden dann zu Tupeln angeordnet, dessen Attribute natürlich immer in der gleichen Reihenfolge auftreten. Da die Recognizer ggf. nicht perfekt sind, kann es bei den Fundstellen zu Überschneidungen oder mehreren Möglichkeiten für z.B. eine Hauptstadt in einer Zeile[1] kommen. Um dann eine einheitliche Reihenfolge zu gewährleisten, wird festgelegt, dass ein Recognizer perfekt sein muss. An diesem wird sich dann orientiert. Auch müssen für das Gelingen des Algorithmus die "Schwächen"[2] der anderen Recognizer bekannt sein.
In [12] wird etwas blauäugig behauptet, dass *Named Entity Recognition* und *Reguläre Ausdrücke* wohl ausreichende Recognizer seien. Allerdings wird dann auch zugegeben, dass nur der Mensch als perfekter Recognizer (derzeit) in Frage kommt und dass alle Auszeichnungen für Experimente von Hand vorgenommen wurden.
Dem Argument, dass im Grunde ja eigentlich die gesamte Informationsextraktion mit Hilfe der Recognizer vonstatten gehen kann, wird entgegnet, dass Wrapper viel schneller sind.

[1] Es wird davon ausgegangen, sich in einer Tabelle zu befinden.
[2] perfect, incomplete, unsound, unreliable

Vergleich mit Prometheus

Die beiden Ansätze haben gemein, dass beide eine Informationsextraktion aus Tabellen anstreben. Kushmericks Wrapper Induction ist aber eher auf generierte Seiten ausgerichtet und arbeitet auf Stringebene. PROMETHEUS hingegen benutzt das *Document Model* von Java zum Auffinden von relevanten Daten. Dadurch wird gewährleistet, dass nur auf Indizes zurückgegriffen wird, die auch für die grafische Präsentation des Dokuments Relevanz haben, so dass der Benutzer interaktiv am Extraktionsprozess beteiligt werden kann.

2.3 Hierarchische Wrapper Induction

Die Motivation für diesen Ansatz ist dieselbe wie in 2.2, man möchte Daten mit Hilfe von Wrappern aus semi-strukturierten Websites gewinnen. Er hat dabei den Anspruch, Informationen nicht nur aus Tabellen, sondern auch aus hierarchischen Strukturen gewinnen zu wollen.

Um diese zu beschreiben, wird der 'embedded catalog' ($\mathcal{EC}$)-Formalismus eingeführt. Die ($\mathcal{EC}$)-Beschreibung einer Seite ist eine baumartige Struktur, in der die Blätter die für den Anwender relevanten Daten darstellen. Die internen Knoten repräsentieren k-Tupel, wobei jeder Posten in diesen Tupeln entweder ein Blatt oder eine weitere Liste darstellt (siehe Abbildung 2.2 für ein Beispiel). Ein Dokument wird nun als eine Sequenz S von Token (z.B. Wörter, Zahlen, HTML-Tags,...) gesehen. Der Inhalt der Wurzel der ($\mathcal{EC}$)-Beschreibung ist dann die gesamte Sequenz S. Der Inhalt der Kinder der Wurzel sind Teilsequenzen. Folglich ist im Allgemeinen der Inhalt eines Knotens x eine Teilsequenz seines Elters p.

Die Extraktion von Daten findet nun mit Hilfe eines Wrappers statt, der die ($\mathcal{EC}$)-Beschreibung des Dokuments und eine Menge von Regeln benutzt. Der Wrapper benötigt jeweils eine Regel, um einen bestimmten Knoten eines Elters zu extrahieren, und zusätzlich Regeln für Listen-Knoten, die besagen, wie die Liste in individuelle Knoten zerlegt werden kann. Diese Regeln basieren auf so genannten 'Landmarks', die eine Gruppe von Token darstellen. So kann man im Beispiel aus Abbildung 2.3 mit der Regel
`R1` = *SkipTo*(<b>)
den Beginn des Restaurantnamens finden. Sie hat die Bedeutung, dass ausgehend vom Beginn des Dokuments alles übersprungen werden soll, bis das <b> Landmark gefunden worden ist. Formell bedeutet das, dass die Regel `R1` auf den Inhalt des Elters des Knoten, in diesem Fall die Wurzel, angewandt wird, wobei das Prefix des Elters bis zum Restaurantnamen absorbiert wird. `R1` ist als *Start-Regel* zu bezeichnen, es gibt aber auch *End-Regeln*, die am

Figure 1. **LA-Weekly and Zagat's Restaurant Descriptions**

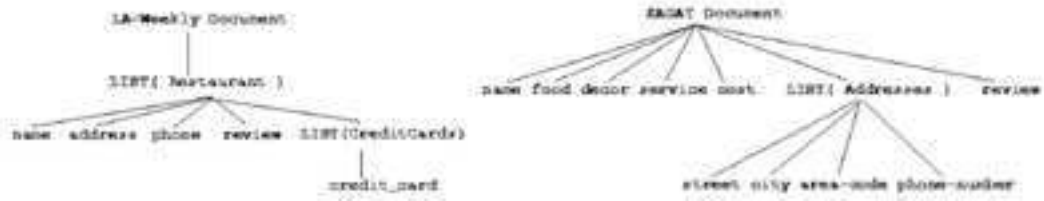


Abbildung 2.2: ($\mathcal{EC}$)-Beschreibung von LA-Weekly und ZAGAT Seiten[13]

Ende des Dokuments beginnen und sich zu dessen Anfang vorarbeiten.
R2 = *SkipTo*()
wäre beispielsweise eine Regel, die das Ende eines Restaurantnamens im Beispiel findet.
Es ist möglich Regeln zu kombinieren, wie R3 = *SkipTo*(Name) *SkipTo*(), die besagt, dass erst alles bis zum Landmark Name und dann alles bis ignoriert werden soll.
Um den verschiedenen Formaten von Dokumenten gerecht zu werden, sind Disjunktionen wie
either *SkipTo*() or *SkipTo*(<i>)
erlaubt.
Um Listen zu bearbeiten, muss der Wrapper zunächst die Liste extrahieren und diese dann in ihre einzelnen Tupel zerlegen. Beim Beispiel aus Abbildung 2.3 wäre das zunächst die Start-Regel *SkipTo*(<p><i>) und die End-Regel

```
1:    <p> Name: <b> Yala </b><p> Cuisine: Thai <p><i>
2:    4000 Colfax, Phoenix, AZ 85258 (602) 508-1570
3:    </i> <br> <i>
4:    523 Vernon, Las Vegas, NV 89104 (702) 578-2293
5:    </i> <br> <i>
6:    403 Pico, LA, CA 90007 (213) 798-0008
7:    </i>
```

Abbildung 2.3: Eine einfache Version des ZAGAT Dokuments[13]

SkipTo() für die Liste, woraufhin die einzelnen Tupel mit *SkipTo*() für den Anfang und *SkipTo*() für das jeweilige Ende zum Einsatz kommen.
Die Vorteile dieser Art der Extraktion liegen auf der Hand:

- Die hierarchische Extraktion erlaubt eine beliebige Tiefe von Verschachtelungen.
- Jeder Knoten wird unabhängig von seinen Geschwistern extrahiert, so dass keine feste Ordnung nötig ist, und fehlende Einträge nichts ausmachen.
- Andere Ansätze werden komplexer, je tiefer der ($\mathcal{EC}$)-Baum wird; hier werden verschiedene einfache Regeln kombiniert.

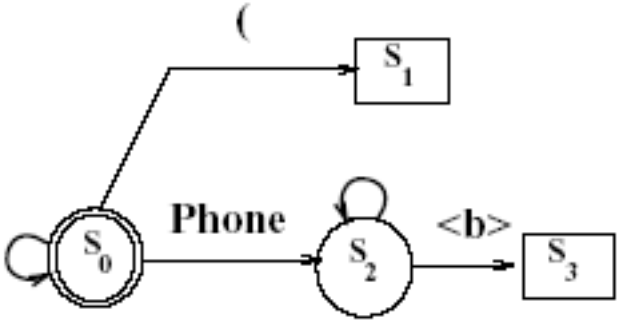


Abbildung 2.4: Ein beispielhafter $\mathcal{SLG}$[13]

Die Extraktionsregeln werden formell als nichtdeterministische finite Automaten definiert, die *Landmark automata* ($\mathcal{LA}$). Jeder Übergang von einem Zustand S_i in einen anderen S_j wird durch einen linearen Landmark $l_{i,j}$ markiert. Ein linearer Landmark ist eine Sequenz von Token und Wildcards wie

HtmlTag, Number, Sign,.... Ein Übergang $S_i \overset{l_{i,j}}{\rightarrow} S_j$ findet nur dann statt, wenn es sich im Zustand S_i bei der Eingabe um einen String s handelt, der vom Landmark $l_{i,j}$ akzeptiert wird.
Simple Landmark Grammars ($\mathcal{SLG}$s) sind die Klasse von $\mathcal{LA}$s, die den disjunktiven Regeln entsprechen. Sie haben folgende Eigenschaften:

- Vom Startzustand S_0 können k Kanten abgehen.
- Es gibt genau k akzeptierende Zustände (einer pro Kante).
- Alle k Kanten, die von S_0 ausgehen sind *sequentielle* $\mathcal{LA}$s, d.h. von jeden Zustand S_i gibt es exakt zwei Übergänge: eine Schleife zu sich selbst und einen Übergang in den nächsten Zustand.
- *Lineare Landmarks* kennzeichnen jeden Übergang, der keine Schleife bezeichnet.
- Alle Schleifen haben die Bedeutung "absorbiere alle Tokens bis auf den linearen Landmark gestoßen wird, der zum nächsten Zustand führt".

Ein beispielhafter $\mathcal{SLG}$ ist in Abbildung 2.4 zu sehen.
Extraktionsregeln werden gelernt, indem ein Algorithmus, *Stalker*, $\mathcal{SLG}$ Regeln generiert, die den Start und das Ende eines Eintrags x innerhalb seines Elters p identifiziert. Dazu werden anhand von ausgezeichneten Beispielen zunächst ein linearer Landmark Automat generiert, der so viele Beispiele wie möglich abdeckt. Für die restlichen Beispiele wird dann wiederum ein Automat mit maximaler Abdeckung gesucht, usw. Wenn alle Beispiele abgedeckt sind, wird als Ergebnis die Disjunktion aller $\mathcal{LA}$s geliefert.
Für eine genaue Beschreibung von *Stalker* siehe [13].

Vergleich mit Prometheus

Der Vergleich fällt ähnlich aus wie in Kapitel 2.2. Beide Ansätze haben gemein, dass sie eine Informationsextraktion aus Tabellen anstreben, wobei der hier vorgestellte mit seinem Hierarchie-Ansatz sogar noch tiefer geht. Die hierarchische Wrapper Induction arbeitet aber auf Stringebene in Verbindung mit einigen Wildcards wie z.B. *htmlTag, Symbol, etc.*. PROMETHEUS hingegen benutzt das *Document Model* von Java zum Auffinden von relevanten Daten. Dadurch wird gewährleistet, dass nur auf Indizes zurückgegriffen wird, die auch für die grafische Präsentation des Dokuments Relevanz haben, so dass der Benutzer interaktiv am Extraktionsprozess beteiligt werden kann.

2.4 Interactive Information Extraction with Constrained Conditional Random Fields

Statistisch gesehen verwenden 59%, d.h. 70 Millionen, der arbeitenden Bevölkerung der USA einen großen Teil ihrer Zeit darauf, Formulare auszufüllen, wenn sie Informationen sammeln und verwalten. Dabei liegen in vielen Fällen die Daten, mit denen die Formularfelder gefüllt werden, schon in einer für den Computer lesbaren Form vor. Um den Arbeiter hierbei bestmöglich zu entlasten, stellen *Kristjannson, Culotta, Viola* und *McCallum* in [14] ihr Modell eines interaktiven Informationsextraktions-Systems vor. Dabei geht es ihnen nicht allein darum, Formulare automatisch mit Informationen aus unstrukturierten Text unter Verwendung des Ansatzes der *Conditional Random Fields* (s.u.) zu füllen, sondern dem Benutzer ein Gefühl der Integrität der Daten zu geben. Dies soll anhand einer interaktiven Oberfläche geschehen, mit der er schnell vorgeschlagene Daten verifizieren und korrigieren kann. Da Formularfelder mitunter in Beziehung zueinander stehen (z.B. *Name* und *Vorname*), sollen außerdem Anwenderkorrekturen auch Auswirkungen auf andere Felder haben, was bedeutet, dass diese ggf. automatisch korrigiert werden. Zudem soll dem Benutzer angezeigt werden, wenn Felder mit Vorschlägen mit einer niedrigen Konfidenz gefüllt worden sind.

In [14] werden diese Ideen in einem Tool zum Erfassen von Kontaktdaten beispielhaft umgesetzt (siehe Abbildung 2.5). Dieses liefert auf der linken Seite ein starres Formular für Daten wie *Name* und *Vorname* und zur Rechten eine Ansicht des (unstrukturierten) Textes, dem die Informationen entnommen werden sollen. Hier kommt nun zunächst die Informationsextraktion mit Hilfe von Conditional Random Fields (CRFs) zum Einsatz. Diese sind eine Verallgemeinerung des *Maximum Entropy* und des *Hidden Markov Models* (HMM); es wird, ähnlich dem HMM, eine Sequenz von Labeln für eine bestimmte Eingabesequenz ermittelt, der Ansatz erlaubt jedoch die Einführung von beliebigen nicht-lokalen Features, die die Abhängigkeit zwischen Labeln (und zwischen Labeln und der Eingabe wie z.B. einer Sequenz der Wörter eines Dokuments) einfangen können. Darunter fallen beispielsweise Eigenschaften von Worten innerhalb eines festgelegten Fensters wie n-Gramme, ob sie groß geschrieben werden, Zahlen beinhalten etc. Für eine genaue Beschreibung von CRFs siehe [15].

Die extrahierten Daten werden danach in den entsprechenden Feldern angezeigt und im Text mit einer zum Feld zugehörigen Farbe unterlegt. Da CRFs es ermöglichen, die Konfidenz in einen Vorschlag zu schätzen, können bei mehreren Vorschlägen für einen Formulareintrag diese geordnet und die Kandidaten mittels eines Drop-down-Menüs angezeigt werden. Vorschläge

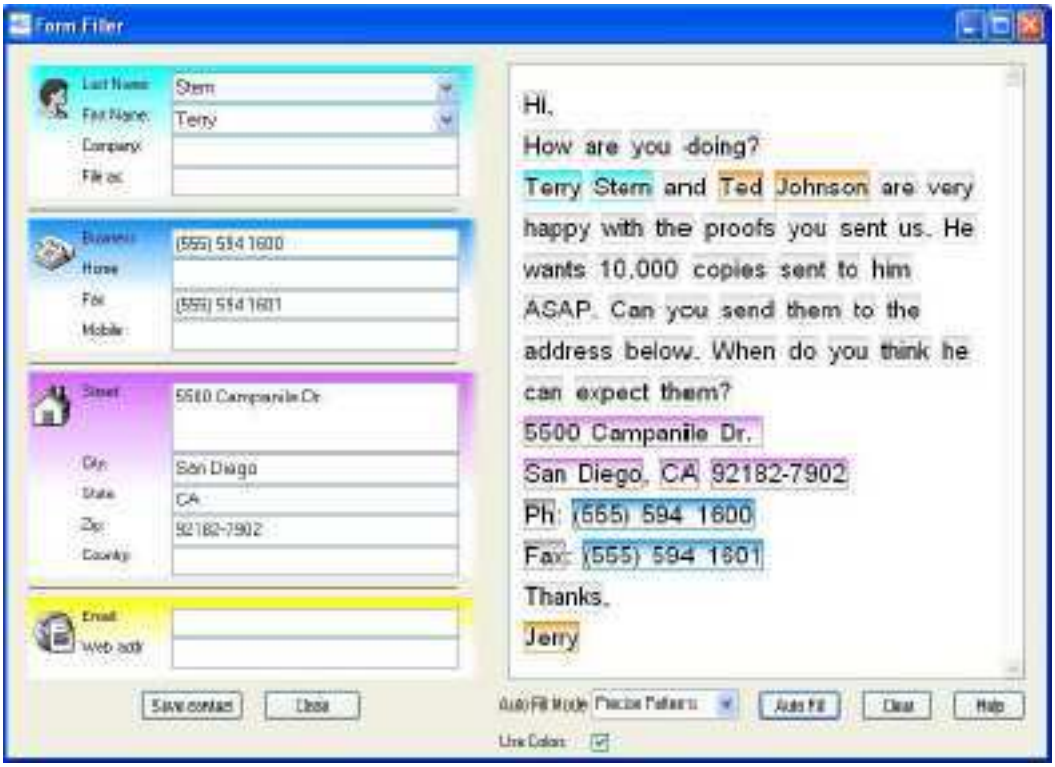


Abbildung 2.5: Eine Benutzeroberfläche für Kontaktinformationen[14]

mit einem niedrigen Konfidenz-Wert werden mit einer Signalfarbe wie orange gekennzeichnet. Die Schätzung findet mittels des *Constrained Forward-Backward* Algorithmus statt (für Details siehe [16]).
Falls der Benutzer eine Korrektur vornehmen möchte, kann er einfach Text markieren und diesen in das Formular ziehen. Dies kann zur Folge haben, dass sich automatisch andere Felder verbessern, was *Correction Propagation* genannt wird. Dies funkioniert mit einem *Constrained Viterbi* Algorithmus. Da bei CRFs Viterbi eingesetzt wird, um den Pfad der wahrscheinlichsten Zustandssequenz (also die wahrscheinlichsten Label) für eine beobachtete Sequenz (Sequenz der Worte eines Dokuments) zu finden, wird bei einer Benutzerkorrektur der Algorithmus dahingehend beschränkt, dass er den besten Pfad, der mit dem korrigierten Feld konform ist, sucht. Dies kann selbstverständlich zur Folge haben, dass sich andere Label und somit Einträge im

Formular ändern können. Für eine genaue Einführung in dieses Thema sei hiermit auf [14] verwiesen.

Vergleich mit Prometheus

Ungefähr zur selben Zeit entwickelt, weisen beide Systeme zumindest an der (Benutzer-)Oberfläche eine Vielzahl von Ähnlichkeiten auf. So gibt es sowohl im System aus [14] als auch bei PROMETHEUS eine Formular- und eine Dokumentansicht, aus der der Benutzer schnell Daten in Felder übertragen kann. Es werden Vorschläge gemacht, die der Benutzer verifizieren oder ablehnen kann, und diese werden bei beiden sogar im Text farblich unterlegt. Die in [14] beschriebene Oberfläche scheint ob der in [14] aufgezeigten Resultate im Bereich *Precision* und *Recall* PROMETHEUS überlegen (siehe [14] und Kapitel 6). Außerdem gibt es hier ein Ordnen von Vorschlägen oder die Propagierung von Korrekturen. Allerdings wird dies durch einen hohen Trainingsaufwand[3] erkauft. Auch lernt das Programm nicht wirklich aus Benutzereingaben. Es gibt zwar das oben erwähnte Propagieren, was sicherlich das Arbeiten erleichtert, aber der Klassifizierer wird einen gemachten Fehler immer wieder und wieder begehen. Neben der Tatsache, dass PROMETHEUS schon mit im Vergleich sehr, sehr wenigen Trainingsbeispielen funktioniert, hat der in dieser Diplomarbeit vorgestellte Ansatz eindeutig den Vorteil, dass das Formular flexibel ist und somit der Anwender sogar zur Laufzeit bestimmen kann, welche Daten extrahiert werden sollen.

[3] Für die Experimente mussten 27560 Wörter von Hand markiert werden[14].

Kapitel 3

Grundbegriffe und Hintergründe

Im Folgenden werden wichtige Grundbegriffe erläutert, die in nachfolgenden Kapiteln als bekannt vorausgesetzt werden.

3.1 XML

XML steht für "Extensible Markup Language". Im Grunde handelt es sich dabei um eine abgespeckte Version von SGML (Standard Generalized Markup Language, ISO 8879:1985, siehe auch www.w3.org/Markup/SGML/), dem internationalen Standard, um Beschreibungen einer Struktur für unterschiedliche Typen von elektronischen Dokumenten zu definieren. XML behält dabei genügend Funktionalität, ist aber von übermäßiger Komplexität befreit. Es handelt sich dabei um eine Auszeichnungssprache, quasi eine Metasprache, um die logische Struktur eines beliebigen Dokuments festzulegen. Man bestimmt also die Art der Daten sowie Abhängigkeiten und Hierarchien in einer Baum-Struktur. Dies alles geschieht mittels Tags, die wie beim verwandten HTML (siehe auch www.w3.org/Markup/) die jeweiligen Informationen umschließen. Tags bestehen aus einem von spitzen Klammern <> umschlossenen Bezeichner. Sie treten stets paarweise auf, ein einleitendes sowie ein abschließendes Tag, das mit einem "/" gekennzeichnet ist. Hier ein Beispiel:

```
<Mitarbeiter>
    <Name>Paschunke</Name>
    <Vorname>Kuno</Vorname>
</Mitarbeiter>
```

Außerdem ist es möglich, Daten auch in so genannten Attributen zu halten, die direkt in einem Tag auftauchen, zum Beispiel:

```
<Mitarbeiter Zweigstelle="69a">
```

Gerade die Flexibilität und die Einfachheit, Hierarchien zu realisieren, machen XML zu einem wertvollen Werkzeug.

3.2 HTML

HTML bedeutet "HyperText Markup Language". Es handelt sich dabei um eine Sprache, die mit Hilfe von SGML (Standard Generalized Markup Language, ISO 8879:1985, siehe auch www.w3.org/Markup/SGML/) definiert wurde. Es stellt praktisch den Standard zur Erstellung von WWW-Seiten dar und erfreut sich aufgrund seiner Einfachheit großer Beliebtheit. HTML hat dabei die Aufgabe, die logischen Bestandteile eines textorientierten Dokuments zu beschreiben, wobei aber auch die Möglichkeit besteht, Grafiken und multimediale Inhalte in Form einer Referenz einzubinden und in den Text zu integrieren. Dem Text wird durch Auszeichnung von Textteilen mit in der Regel paarweisen (öffnenden und schließenden) Tags eine Struktur verliehen. Die jeweils zusammengehörenden Tags bilden zusammen mit dem dazwischen liegenden Text („Inhalt“) ein Element; diese Elemente lassen sich auch nach bestimmten Regeln verschachteln. Dadurch wird ermöglicht, typische Bestandteile eines textorientierten Dokuments, wie Überschriften, Textabsätze, Listen, Tabellen oder Grafikreferenzen, als solche auszuzeichnen. [17, 18]
Hier ein Beispiel:

```
<TABLE FRAME="BOX" RULES="ALL" BORDER="1">
    <TR>
        <TD>Name</TD><TD>Paschunke</TD>
    </TR>
    <TR>
        <TD>Vorname</TD><TD>Kuno</TD>
    </TR>
</TABLE>
```

Dieses Beispiel ergibt (eingebettet in ein HTML-Dokument) in einem üblichen Browser eine Anzeige wie in Abbildung 3.1.
Tags, die bei der Verarbeitung einer HTML-Datei in PROMETHEUS eine entscheidende Rolle spielen, sind:

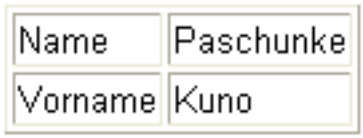


Abbildung 3.1: Tabelle mit HTML

<hr> Horizontale Linie
<table> Tabelle
<td> Tabellenzelle
<tr> Tabellenzeile
Außerdem wird auf die Attribute *FACE* und *SIZE* des <font> Tags zurückgegriffen. Dieses Tag existiert nicht in der Dokumentrepräsentation in Java, es werden eben nur jene Attribute gespeichert (siehe Kapitel 3.5.1).

3.3 TIFF

TIFF (engl. "Tagged Image File Format") ist ein Dateiformat zur Speicherung von Bilddaten. Das TIFF-Format wurde ursprünglich von Aldus (1994 von Adobe übernommen) und Microsoft für gescannte Rastergrafiken für die Farbseparation entwickelt. Zusammen mit Encapsulated Postscript ist es das wichtigste Format zum Austausch von Daten in der Druckvorstufe.
Die Kodierung von Zahlen (Byte order) kann entweder Big Endian oder Little Endian sein. In einer Datei können mehrere Bilder abgelegt werden (Multipage-TIFF). Das können, müssen aber nicht, verschiedene Versionen desselben Bildes sein, z. B. ein Vorschaubild (Thumbnail) und das Originalbild. TIFF kennt verschiedene Farbräume und Algorithmen zur Datenkompression. Dabei unterstützt es sowohl verlustlose (z.B. LZW, Lauflängenkodierung) als auch verlustbehaftete Kompressionsverfahren (z.B. eines der JPEG-Verfahren). Es ist auch möglich, IPTC-Metadaten in der TIFF-Datei unterzubringen.
[19]

3.4 Pdf

Adobe PDF ("Portable Document Format") ist ein von Adobe Systems entwickeltes offenes Dateiformat, das von Standardisierungsgremien weltweit für den sicheren und zuverlässigen Austausch von elektronischen Dokumenten

verwendet wird. PDF-Dateien erhalten dabei die Darstellungsqualität der Originaldokumente. Sie behalten Schriftarten, Bilder, Grafiken und Layout jedes Ausgangsdokuments bei, unabhängig davon, welche Anwendung und Plattform bei der Erstellung eingesetzt wurden. Zum Anzeigen und Ausdrucken wird lediglich der kostenlose Adobe Reader benötigt. Dieser bietet u.a. leistungsstarke Suchfunktionen, mit denen Adobe PDF-Dateien nach Wörtern, Lesezeichen und Datenfeldern durchsucht werden können. [20]

3.5 Datenhaltung in Java Swing

PROMETHEUS benutzt für die Anzeige von HTML-Dateien die javax.swing.JEditorPane Klasse, eine Erweiterung von javax.swing.text.JTextComponent, das die Superklasse für alle Swing Komponenten, die sich mit Text befassen, darstellt und all deren Funktionalitäten definiert. Swing Komponenten basieren auf dem *Model-View-Controller* (MVC) Design (s. Abbildung 3.2). Der *'Model'*-Teil ist dafür zuständig, jeden möglichen Status zu verwalten, wie z.B. den Zustand eines Buttons, d.h. gedrückt oder nicht; der *'View'*-Teil bestimmt die visuelle Repräsentation des *'Models'*; der *'Controller'* sagt, ob die Komponente auf eine Eingabe von Maus oder Tastatur reagieren soll. Beim Markieren einer Check-Box, zum

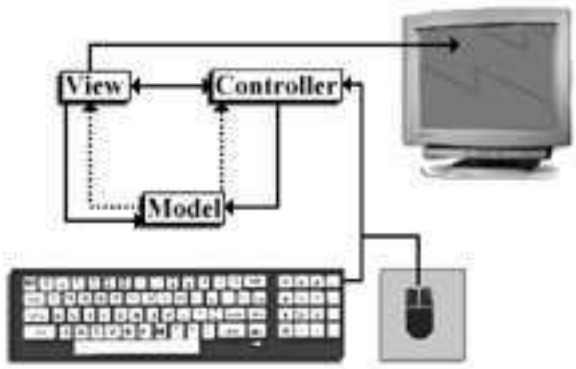


Abbildung 3.2: MVC Design

Beispiel, bestimmt der *'Controller'*, dass ein Mausklick stattgefunden hat und sendet eine Nachricht an die Ansicht (*'View'*) der Komponente. Diese stellt fest, dass auf die Check-Box geklickt wurde und sendet eine Nachricht an das *'Model'*. Dieses aktualisiert sich und übermittelt eine Nachricht, die vom *'View'*-Teil empfangen wird. Diese besagt, dass die Ansicht sich auf den neuen Status des *'Models'* anpassen soll. So bleibt das *'Model'* unabhängig

von den beiden anderen Teilen. [21]
Da PROMETHEUS ständig Daten aus der angezeigten HTML-Datei extrahieren muss, ist es unerlässlich zu verstehen, wie Java diese organisiert, d.h. wie das *'Model'* von JEditorPane aufgebaut ist.

3.5.1 Das *'Document Model'* in Java

Sobald man einer JEditorPane eine HTML-Seite als anzuzeigenden Inhalt zuweist, wechselt es zu dem passenden EditorKit (siehe [22] s. 1047 ff.), das im allgemeinen dafür zuständig ist, ggf. zu bestimmen, wie die Ansicht erstellt werden soll, mögliche Aktionen zu definieren und die richtige Datenstruktur, das Dokument, zur Verfügung zu stellen, in diesem Fall das javax.swing.text.html.HTMLDocument, das das Document Interface implementiert. Dieses beinhaltet nicht nur den eigentlichen Text, sondern beschreibt auch sein *'Model'*. Durch eine hierarchisch angeordnete Menge von Elementen wird die Struktur des Dokuments bzw. dessen Inhalt in Form einer Baumstruktur beschrieben. Dadurch hat auch jedes Dokument (mindestens) ein Wurzelelement. Wenn man den Baum von der Wurzel ausgehend durchläuft, stößt man auf Knoten- und Blatt-Elemente. Jedes Blatt, das auch unter dem Namen CharacterElement läuft, wird mit einem Stück Text assoziiert, dessen Position man durch die Methoden getStartOffset() und getEndOffset in Erfahrung bringen kann. Knoten-Elemente werden bei StyledDocuments (siehe http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/text/StyledDocument.html) auch ParagraphElements genannt, da durch sie der Text in verschiedene Paragraphen eingeteilt wird.[21] Im Falle eines HTMLDocuments stellen diese jedoch eher Teile des Markup dar, nämlich diejenigen, die eine bestimmte Strukturierung des Texts, wie das Zentrieren oder Tabellen, erzwingen. Da Elemente jeweils eine Bezeichnung besitzen, die man mit getName() erfahren kann, weiß man auch stets für welches HTML-Tag es steht.
Mit den Elementen sind unter Umständen auch eine Menge von Attributen verknüpft, die in Farb-, Font-, Zeichen- und Paragraph-Attribute aufgeteilt werden; die Liste kann jedoch beliebig erweitert werden. Attribute sind Schlüssel/Wert Paare, die durch die Ansicht(*'View'*) interpretiert werden. Sofern sie nicht überschrieben werden, werden sie an abhängige Elemente weitergegeben. Außerdem werden durch sie HTML Tags wie beispielsweise <B> oder <I> ersetzt.
Durch die Elemente der Baumstruktur lässt sich nun mit dem javax.swing.text.ElementIterator navigieren. Dieser startet entweder beim Wurzel- oder einem beliebigen angegebenen Element und durchläuft den darunter liegenden Baum in depth-first Manier.

Da es nun mit PROMETHEUS zu jeder Zeit möglich ist, Textstellen zu markieren wie auch zu bestimmten Passagen zu springen, oder das Programm selbst diese Schritte ausführt, ist es notwendig, jegliche internen Operationen auf der Element-Ebene auszuführen, auch wenn es auf den ersten Blick sinnvoller scheint, auf der reinen Textebene zu arbeiten; diese liefert jedoch keine sinnvollen Offsets, d.h. Positionen, an denen der Text tatsächlich für den Benutzer sichtbar ist. Außerdem ist dieses Modell im Bereich des automatischen Taggings sowohl sinnvoll als auch nützlich, da das Markup sofort zur Verfügung steht.

3.6 OCR

Die für diese Diplomarbeit zur Verfügung gestellten Geschäftsbriefe existierten zunächst nur in Papierform. Nachdem diese gescannt worden waren, lagen sie entweder in dem Format TIFF (siehe Kapitel 3.3) oder Pdf (siehe Kapitel 3.4) vor. Dies waren die Ausgangsdaten für dieses Projekt. Dabei ist zu beachten, dass auch diese TIFF- und Pdf-Dateien den Brief nur in Bildform beinhalten. Da der Computer im Gegensatz zum Menschen aus Bildern per se keine textuellen Informationen gewinnen kann, ist es nötig, ein OCR-Programm einzusetzen.
Als *OCR* (Optical Character Recognition) wird der Prozess bezeichnet, die digitale Pixelrepräsentation eines Textdokuments in eine Buchstabenfolge zu überführen. Er kann grob in vier Phasen unterteilt werden, nämlich Vorverarbeitung, Merkmalsextraktion, Klassifikation und eine Postkorrektur, meistens mit Unterstützung eines Lexikons (s.u.). Hierfür stehen einige (kommerzielle) Lösungen wie beispielsweise *Abbeys FineReader* bereit.
Leider ist dieses Verfahren trotz aller wissenschaftlicher Anstrengungen nicht ganz fehlerfrei; so kommt es ob der Ähnlichkeit vieler Zeichen zu Rechtschreibfehlern. Beispielsweise kommen folgende Verwechslungen oft vor:
$ii \leftrightarrow ü$, $l \leftrightarrow 1$, $m \leftrightarrow n$, usw.
Da derartige Anwendungen so gut wie immer auch mit der Unterstützung von Wörterbüchern arbeiten, sind sie natürlich abhängig von deren Güte; außerdem stellen Eigennamen eine Hürde dar. Auch für handschriftlichen Text gibt es (bis jetzt) keine zufriedenstellenden Lösungen.
Programme wie *FineReader* wandeln nicht nur Bild- in Textdaten um, sondern versuchen auch, die Struktur nachzuahmen, d.h. Paragraphen, Tabellen und Positionen von Abbildungen beizubehalten.

3.7 Levenshtein-Abstand

Der *Levenshtein*-Abstand zweier Wörter V und W ist der minimale Editierabstand, der nötig ist, um V in W zu überführen. Der Editierabstand ergibt sich hier (es gibt durchaus eine Vielzahl von Variationen, die auf diesen Ansatz basieren) aus der Anzahl der Anwendung folgender Operationen: *Einfügung* eines Symbols, die *Löschung* eines Symbols, sowie die *Substitution* eines Symbols durch ein anderes.

3.7.1 Berechnung des Levenshtein-Abstands nach Wagner und Fisher

(aus [23])
Zur Berechnung des Editierabstands gibt es ein Standardverfahren, das auf dynamischer Programmierung beruht und von Wagner und Fisher eingeführt wurde. Die Überlegung dahinter ist, dass man den Levenshtein-Abstand zweier Wörter in systematischer Weise aus den Levenshtein-Abständen ihrer Buchstabenpräfixe erhält. Ist V ein Wort der Länge $x \geq 0$, so bezeichnen wir mit V_i das i-te Symbol von V $(1 \leq i \leq n)$. Es steht weiter $V_{1,i}$ für das Präfix mit den Symbolen $V_1, ..., V_i$ von V $(0 \leq i \leq n)$. Für $i = 0$ ist also $V_{1,i}$ gerade das leere Wort ϵ . Bei der nachfolgenden Darstellung des Berechnungsverfahrens verwenden wir eine Notation, die sich sofort verallgemeinern lässt. Wir setzen

$c(a, \epsilon) = 1$ die Kosten für das Löschen des Symbols a
$c(\epsilon, a) = 1$ die Kosten für das Einfügen des Symbols a
$c(a, b) = 1$ die Kosten für das Substitieren von b für a $(a \neq b)$
$c(a, a) = 0.$

Es seien nun zwei zu vergleichende Wörter V und W der Länge m bzw. n gegeben. Wir definieren ein zweidimensionales Array $C[i, j]$ der Größe $(m + 1)$x$(n + 1)$, wo $0 \leq i \leq m$ und $0 \leq j \leq n$. Die Idee ist, dass im Feld $C[i, j]$ der Editierabstand von $V_{1,i}$ nach $W_{1,j}$ notiert wird. Um die Array-Einträge zu berechnen, verwenden wir die Idee der dynamischen Programmierung.

Zur *Initialisierung* berechnen wir zunächst die Einträge der ersten Spalte $C[\_, 0]$. Es ist $C[i, 0]$ der Editierabstand von $V_{1,i}$ und $W_{1,0} = \epsilon$. Es ergibt sich offenkundig durch die Summe der Kosten der Löschung der Symbole $V_1, ..., V_i$ $(1 \leq i \leq m)$. Wir erhalten $C[0, 0] := 0$ und $C[i, 0] = C[i - 1, 0] + c(V_i, \epsilon)$ für $1 \leq i \leq m$. Analog erhalten wir für die verbleibenden Einträge der ersten Zeile die Formel $C[0, j] = C[0, j - 1] + c(\epsilon, W_j) \quad (1 \leq j \leq n)$.

Bei der Berechnung der verbleibenden Einträge $C[i,j]$ setzen wir induktiv voraus, dass die Werte $C[i-1,j-1]$, $C[i-1,j]$ und $C[i,j-1]$ bereits vorliegen und im obigen Sinn korrekt sind. Wie oben dargestellt sind drei Möglichkeiten denkbar, wie $W_{1,j}$ aus $V_{1,i}$ hervorgeht.

> 1. indem wir zunächst $V_{1,i-1}$ in $W_{1,j-1}$ umeditieren und die Substitution von V_i durch W_j anschließen, was Gesamtkosten von $C[i-1,j-1]+c(V_i,W_j)$ ergibt,
>
> 2. indem wir zunächst $V_{1,i}$ in $W_{1,j-1}$ umeditieren und die Einfügung von W_j anschließen, was Gesamtkosten von $C[i,j-1]+c(\epsilon,W_j)$ ergibt, oder
>
> 3. indem wir zunächst $V_{1,i-1}$ in $W_{1,j}$ umeditieren und die Löschung von V_i ergänzen, was zu Gesamtkosten von $C[i-1,j]+c(V_i,\epsilon)$ führt.

Der Eintrag $C[i,j]$, also der Editierabstand von $V_{1,i}$ nach $W_{1,j}$ ergibt sich als Minimum dieser drei Alternativen. Hieraus erhält man die Beschreibung des Algorithmus von Wagner und Fisher:

Initialisierung
$C[0,0] := 0$
for $i = 1$ to m
$C[i,0] := C[i-1,0] + c(V_i,\epsilon)$
for $j = 1$ to n
$C[j,0] := C[0,j-1] + c(\epsilon,W_j)$

Berechnung der inneren Array-Werte
for $i = 0$ to $m-1$
for $j = 0$ to $n-1$
$C[i,j] = min\{C[i-1,j-1]+c(V_i,W_j),\quad C[i,j-1]+c(\epsilon,W_j),\quad C[i-1,j]+c(V_i,\epsilon)\}$

Beispiel 3.7.1 *Berechnung des Levenshtein-Abstands der Wörter "Kalle" und "Karlheinz".*

Jeder Sprung horizontal oder vertikal entspricht einer Editieroperation (Einfügen bzw. Löschen eines Zeichens). Der diagonale Sprung kostet 1, wenn die zwei Buchstaben in die Reihe und Spalte nicht übereinstimmen,

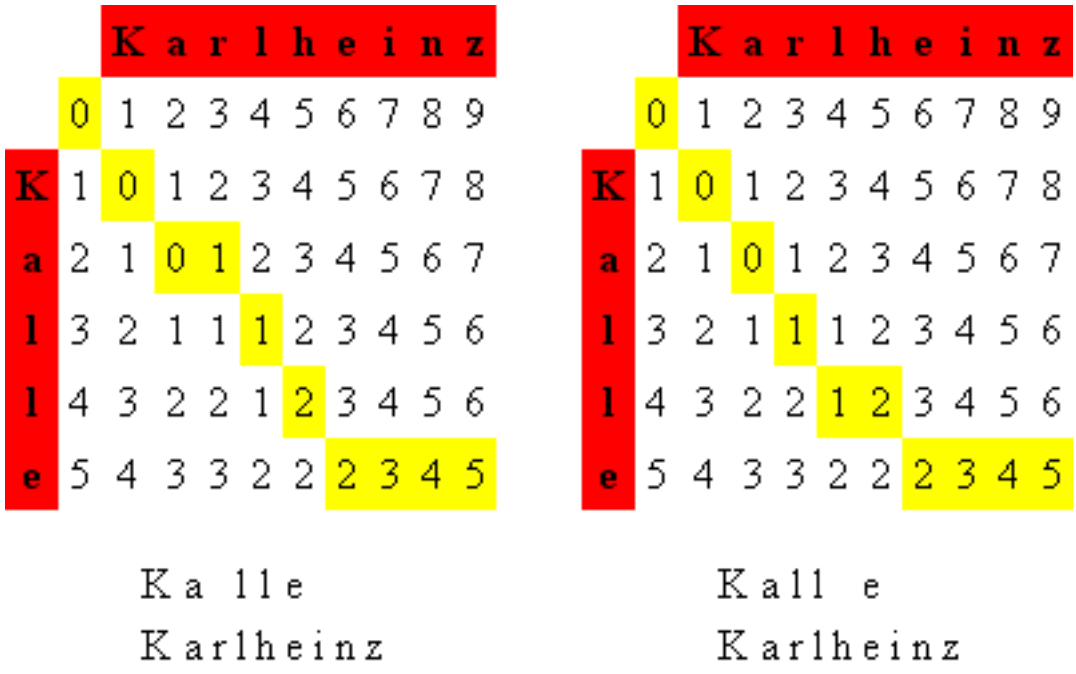


Abbildung 3.3: Zwei Wege, ein Ziel

oder im Falle einer Übereinstimmung 0. Jede Zelle minimiert jeweils die lokalen Kosten. Daher entspricht die Zahl in der untereren rechten Ecke dem Levenshtein-Abstand zwischen den beiden Wörtern.[24]
Wie man in Abbildung 3.3 sieht, gibt es zwei mögliche Wege durch das Tableau, die die geringsten Kosten produzieren.

3.8 Reguläre Ausdrücke

Reguläre Ausdrücke (Abk. RegExp oder Regex, engl. regular expression) dienen der Beschreibung einer Familie von formalen Sprachen, d. h. sie beschreiben (Unter-)Mengen von Zeichenketten. Sie gehören deswegen zur Theoretischen Informatik. Hier bilden sie die unterste und somit ausdrucksschwächste Stufe der Chomsky-Hierarchie (Typ-3). Es lässt sich zeigen, dass zu jedem regulären Ausdruck ein gleichwertiger endlicher Automat existiert und umgekehrt.
Der Mathematiker Stephen Kleene benutzte eine Notation, die er reguläre Mengen nannte. Ken Thompson nutzte diese Notation um *qed* (eine Vorgängerversion des Unix-Editors *ed*) zu bauen und später das Werkzeug

grep zu schreiben. Seither implementieren sehr viele Programme Funktionen, um reguläre Ausdrücke zum Suchen und Ersetzen von Zeichenketten zu nutzen. Einige Programme wie z. B. *Perl* unterstützen einige Erweiterungen der regulären Ausdrücke, z. B. Rückwärtsreferenzen. Hierbei handelt es sich nicht mehr um reguläre Ausdrücke im Sinne der theoretischen Informatik, denn die so erweiterten Ausdrücke gehören nicht mehr notwendigerweise zur untersten Stufe der Chomsky-Hierarchie.
Eine häufige Anwendung regulärer Ausdrücke besteht darin, spezielle Zeichenketten in einer Menge von Zeichenketten zu finden. Hierbei wird ein regulärer Ausdruck aus den Zeichen des zugrunde liegenden Alphabets in Kombination mit sogenannten Metazeichen wie ([,], (,), {, }, ?, +, *, ^, $, \, .) gebildet.[25] Alle übrigen Zeichen des Alphabets stehen für sich selbst. Mit Hilfe der Metazeichen lassen sich beispielsweise Zeichenklassen definieren, die es erlauben, mehr als ein Zeichen an einer bestimmten Position der Zeichenkette zuzulassen. Zudem gibt es unter ihnen Quantifizierer, die den vorherigen Ausdruck in verschiedener Vielfachheit in der Zeichenkette zulassen. Außerdem lassen sich mit runden Klammern Ausdrücke zusammenfassen. Es gibt beispielsweise Sonderzeichen für den Zeilenanfang, das -ende und eine Reihe weiterer hilfreicher Funktionen.
Folgende Features sind innerhalb dieser Diplomarbeit von Bedeutung[26]:

Metazeichen	**Passt auf**
.	Irgendein Zeichen
[...]	Eines der Zeichen aus der Liste
[^...]	Irgendein Zeichen *nicht* aus der Liste
[a-f]	Alle Buchstaben von *a* bis *f*
\t	Ein Tabulatorzeichen(Tab)
\n	Ein Newline
\r	Carriage-Return
\f	Ein Formfeed
\s	[\t\n\x0B\f\r], "weiße Zeichen", Whitespace (Leerzeichen, Tab, Newline, Formfeed, usw.)
\S	Jedes Zeichen, das nicht zu \s gehört
\d	[0-9], also eine Ziffer ("digit")
\b	Eine Wortgrenze
|	Alternation. Passt auf mindestens eine der Alternativen
(...)	Beschränkt den Geltungsbereich von Alternationen; gruppiert Dinge für nachfolgende Quantifier
$X?$	Quantifier. X ist einmal erlaubt, aber optional
$X*$	Quantifier. Jede Anzahl von X ist erlaubt, auch Null
$X+$	Quantifier. Mindestens ein X, mehr erlaubt.

Kapitel 4

Prometheus

Obwohl zunächst als eine reine Lernumgebung konzipiert, sollte PROMETHEUS nicht nur ein Programm werden, das sich ausschließlich auf seine Lernkomponente verlässt, sondern ein tatsächlich einsetzbares Tool, das auch in der realen Welt, d.h. außerhalb der universitären Forschung, bestehen kann. Deswegen ist es unerlässlich, dem Benutzer unterstützende Komfortelemente zur Verfügung zu stellen. Zu ihnen gehört zunächst einmal eine flexible grafische Oberfläche, Excel-Anbindung, robuste Suchwerkzeuge und Zugriff auf lediglich eingescannte Originaldokumente. Die beiden letztgenannten Funktionen sind notwendig, da davon ausgeganden wird, dass die zu bearbeitenden Daten durch einen OCR-Schritt zustande gekommen sind. Dieses Kapitel soll die Grundlagen zum Arbeiten mit PROMETHEUS und deren Funktionsweisen näherbringen.

4.1 Definitionen

Zum eingehenden Verständnis müssen an dieser Stelle einige PROMETHEUS eigene Begriffe geklärt werden, nämlich *Passage, Eintrag, Vorkommnis*. Diese sind in dieser Diplomarbeit mehrdeutig belegt. Die Mehrdeutigkeit entwuchs einem evolutionären Prozess während der Erstellung von PROMETHEUS. Am Anfang gab es tatsächlich eine strikte Trennung der Bedeutungen innerhalb des Systems, die jedoch einen höheren verwaltungstechnischen Aufwand sowie mehr Speicherverbrauch mit sich brachte, was in keinem Verhältnis zum Nutzen stand. Nach und nach wurden dann die semantisch ähnlichen Konzepte zusammengeführt.
Aus einem Dokument extrahierte Informationen werden in PROMETHEUS in einer hierarchischen Struktur abgespeichert, die an realen Auswertungen von

Geschäftsbriefen angelehnt ist. Die obersten Elemente dieser Struktur nennen sich hier *Passagen*. Sie besitzen einen Bezeichner, der als eine Art Überschrift für seine untergeordneten Elemente gesehen werden kann. Diese heißen *Einträge*, besitzen auch einen Bezeichner und sind quasi als die Attribute zu sehen, die die Informationen aufnehmen. Somit bildet die Summe aller Passagen und Einträge nach Kapitel 1 das *Template* der Informationsextraktion. Die Werte, die dieses aufnimmt, werden in dieser Diplomarbeit *Vorkommnisse* genannt. Einem *Eintrag* können mehrere *Vorkommnisse* zugewiesen werden. Mit *Vorkommnis* ist entweder die Textstelle im Dokument, der an dieser Stelle gewonnene String oder dessen Anzeige in der GUI gemeint. Abgesehen von der konzeptionellen Seite kann der Begriff *Passage* auch für seine grafische Repräsentation sowie *Eintrag* für die Ansammlung von Buttons und Textfeld in einer Zeile stehen, die durch einen Button, der den jeweiligen Bezeichner als Namen trägt, angeführt wird (siehe Abbildung 4.2).

4.2 Grafische Benutzeroberfläche

PROMETHEUS präsentiert sich nach dem Start als leerer Desktop. Jetzt ist es möglich, im 'File'-Menü mit 'Open' einen Geschäftsbrief zu öffnen, dabei ist es nur erlaubt, HTML-Dateien[1] zu laden. Nach der Auswahl öffnet sich ein Fenster, das mit dem Dokument assoziiert wird. Es können mehrere Dokumente gleichzeitig geöffnet sein, die jeweils in einem eigenen Fenster zur Verfügung stehen. Diese zeigen auf der linken Seite das Dokument und auf der rechten ein flexibles Formular, welches die logische Untergliederung mit *Passagen* und dazugehörigen *Einträgen* illustriert. Jede Zeile verfügt zudem über ein Textfeld, um Informationen, *Vorkommnisse*, aufzunehmen sowie anzuzeigen. Das Formular bietet somit ein *Template* (siehe Kapitel 1) zur Informationsextraktion. Vorkommnisse können auf verschiedene Arten registriert werden, siehe dazu Kapitel 4.9.

Zentrales Element einer jeden Zeile ist aber natürlich der Eintrags-Button. Dieser trägt nicht nur den Bezeichner des jeweiligen Eintrags, sondern zeigt daneben anhand zweier durch ein "/" getrennte Zahlen auch an, welches das aktuelle Vorkommnis ist und wieviele Vorkommnisse diesem Eintrag zugeordnet worden sind. Zudem spielt er auch eine Rolle bei der Zuweisung (siehe Kapitel 4.9). Falls mehrere Vorkommnisse verzeichnet sind, kann man sie mit diesem Button durchlaufen. Nach einem Linksklick darauf wird der entsprechende Text im Textfeld angezeigt und, außer bei einem Text-Vorkommnis

[1] Es wird davon ausgegangen, dass diese Dateien zuvor mit einem OCR-Programm erstellt worden sind, siehe Kapitel 3.6

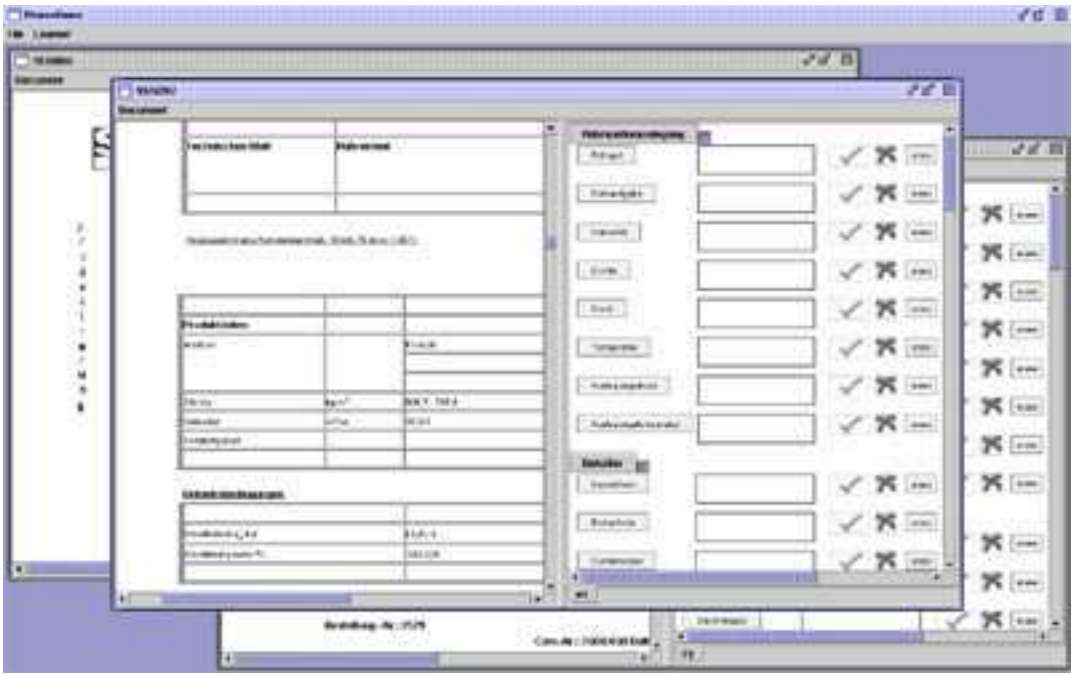


Abbildung 4.1: Dokument und Formular

(siehe Kapitel 4.9), zu der relevanten Stelle in der Dokumentansicht gesprungen. Dieser Sprung geschieht auch bei nur einem einzigen zugewiesenen Vorkommnis.
Neben dem oben genannten Textfeld gibt es zwei Buttons, der 'Positives Beispiel'- und der 'Negatives Beispiel'-Button, die eng mit dem Lerner verknüpft sind. Zur näheren Erläuterung ihrer Auswirkungen verweise ich auf Kapitel 4.9. Darauf folgt der Farb-Button, der anzeigt, in welcher Farbe ein Vorkommnis im Dokument markiert wird, die sich außerdem über einen Auswahldialog wechseln lässt, der nach Betätigung des Knopfes erscheint. Natürlich erfahren alle entsprechenden Einfärbungen der geöffneten Dokumenten ein Update.
Dem Erase-Button kommen zwei Funktionen zu. Wenn dem entsprechenden Eintrag mindestens ein Vorkommnis zugewiesen worden ist, wird das aktuelle gelöscht und das vorhergehende angezeigt; Näheres zum Löschen wird detailierter in Kapitel 4.9 beschrieben. Darüberhinaus ist es über diesen Knopf möglich, den gesamten Eintrag zu löschen, was bei anderen geöffneten Dokumenten dazu führt, dass der Eintrag genauso wie die zugeordneten Vorkommnisse gelöscht wird.
Per Synonyms-Button kann ein Dialog zur Verwaltung der Synonyme des

Abbildung 4.2: Formular

Eintragsbezeichners geöffnet werden (Abbildung 4.3). Diese Synonyme spielen bei den Auszeichnungs-Methoden von PROMETHEUS (siehe Kapitel 4.7 und 4.8) eine Rolle. In diesem Dialog können Synonyme in das Textfeld eingetragen und durch den 'add'-Button bzw. der 'Enter'-Taste übernommen werden. Verzeichnete Synonyme werden in der Liste auf der rechten Seite aufgeführt. Durch Markieren und Betätigen des 'remove'-Buttons können diese wieder entfernt werden, was ggf. Konsequenzen auf die Vorkommnisse dieses Eintrags in allen Dokumenten haben kann (siehe Kapitel 4.9).
Die Funktion des Type-Buttons wird in Kapitel in Kapitel 4.8) erläutert.
Ein neuer Eintrag kann erstellt werden, indem man auf den Button derjenigen Passage drückt, der der Eintrag untergeordnet werden soll. Daraufhin öffnet sich ein Dialog, in dem man den Namen des Eintrags und dessen Markierungsfarbe festlegen kann (Abbildung 4.4).
Eine neue Passage wiederum lässt sich über den Menüpunkt 'New Passage' im 'Document'-Menü des Dokuments einrichten (Abbildung 4.5).
Gelöscht werden kann eine Passage durch Betätigen des kleinen Buttons, der sich rechts neben dem betroffenen Passage-Buttons befindet. Das hat nicht nur zur Folge, dass die Passage entfernt wird, sondern auch all ihr untergeordneten Einträge samt verzeichneter Vorkommnisse, was zu Konsequenzen führt, die in Kapitel 4.9 beschrieben werden.
Für den Fall, dass ein Dokument mehrere Strukturen wie beispielsweise Produkte beschreibt, kann der Benutzer weitere Formulare über den Punkt 'New Form' im 'Document'-Menü anlegen. Über Reiter am unteren Rand der Formularansicht kann man zwischen ihnen wechseln. Sobald es mehr als ein Formular gibt, kann man Vorkommnisse zwischen ihnen verschieben. Dazu drückt man über einem Textfeld, das ein Vorkommnis anzeigt, die rechte

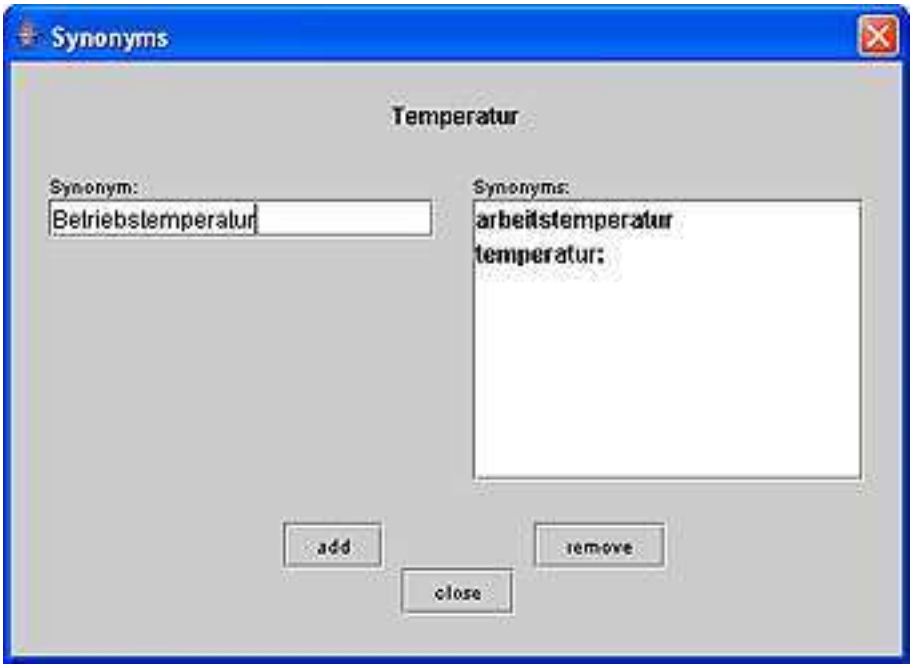


Abbildung 4.3: Dialog zur Verwaltung der Synonyme des Eintragsbezeichners "Temperatur"

Maustaste, woraufhin ein Pop-up Menü erscheint (siehe Abbildung 4.6). In diesem kann man wählen, in welches Formular es verschoben werden soll. Danach wird es aus diesem Formular entfernt und taucht im gewünschten unter demselben Eintrag wieder auf.
Gelöscht werden kann das gerade angezeigte Formular mit der Menüoption 'Remove Form', wobei alle darauf eingetragenen Vorkommnisse ebenso entfernt werden.
Änderungen am Formular selbst werden mit 'Save' im 'File'-Menü gespeichert. Die Flexibilität der Benutzeroberfläche wird erreicht, indem die Struktur des Formulars in einer XML-Datei abgelegt wird. Deshalb können, wenn auch weniger komfortabel, neue Passagen, Einträge etc. von Hand in dieses File eingetragen werden, das folgendes Aussehen hat:

```
<?xml version="1.0" encoding="UTF-8"?>
<entries>
    <passage>
        <label>Rührwerksauslegung</label>
        <entry type="catchword">
            <label>Rühraufgabe</label>
```

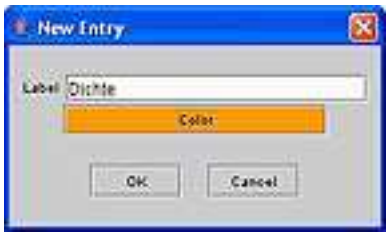


Abbildung 4.4: Erstellen eines neuen Eintrags

Abbildung 4.5: Erstellen einer neuen Passage

```
            <color>-6710887</color>
            <synonym>Rühraufgabe:</synonym>
            <catchword>Homogenisieren</catchword>
        </entry>
        <entry type="digit">
            <label>Dichte</label>
            <color>-3342337</color>
            <synonym>Dichte:</synonym>
            <synonym>düchte</synonym>
            <measure>kg/m3</measure>
        </entry>
    </passage>
    <passage>
    .
    .
    .
</entries>
```

Das 'entries'-Tag umschließt alle für das Formular wichtigen Daten. Ihm untergeordnet sind die Passagen, unter denen der jeweilige Passagenbezeichner innerhalb des 'label'-Tags und die zugehörigen Einträge eingeschlossen von 'entry'-Tags zu finden sind. Innerhalb des 'entry'-Tags ist das 'type'-Attribut

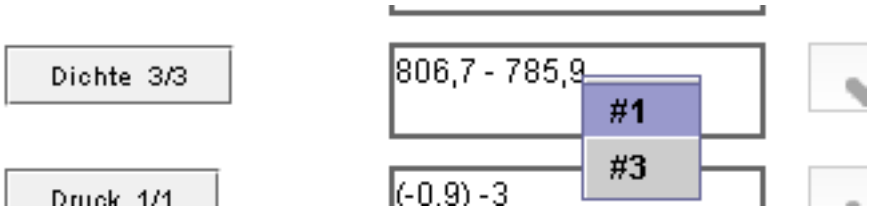


Abbildung 4.6: Verschieben eines Vorkommnisses

abgespeichert, das den Vorkommnistyp beschreibt (siehe Kapitel 4.8). Ansonsten werden, dem Eintrag untergeordnet, innerhalb der jeweiligen Tags folgende Daten abgelegt:

Tag	*Bedeutung*
<label>	der Eintragsbezeichner
<color>	die Farbe
<synonym>	ein Synonym des Eintragsbezeichners
<measure>	eine mit dem Vorkommnis auftretende Maßeinheit
<catchword>	ein Stichwort

Die Umsetzung in Java erfolgte mit Interfaces wie org.w3c.dom.Document, org.w3c.dom.Element und der Klasse javax.xml.parsers.DocumentBuilder, die einen Elementbaum gemäß dem *'Document Object Model'* [27] aufbauen. Da die Navigation darin zum Teil träge und ineffektiv verläuft, werden Daten wie die Zusammengehörigkeit von Passagen und Einträgen plus Synonymen oder die Information darüber, welche Strings einen Eintrag bezeichnen könnten, in zusätzlichen Hashtables und Vektoren gespeichert, da diese Daten häufig benötigt werden und dem Benutzer keine allzu langen Wartezeiten, beispielsweise beim Ausführen des automatischen Auszeichnens (Kapitel 4.7), zugemutet werden sollen.
Wenn PROMETHEUS bzw. der Benutzer nun ein neues Formular anfordert, wird anhand der XML-Vorlage ein grafisches Formular mitsamt Funktionspaket erstellt, dem aktuellen Dokument zugewiesen und angezeigt.

4.3 Suche

Eine Suchfunktion ist bei der interaktiven Informationextraktion unerlässlich, da ein Geschäftsbrief keine standardisierte Länge aufweist, sondern durchaus einen Umfang von hundert Seiten oder mehr besitzen kann. Durch die Suche nach einem relevanten Begriff wie einem möglichen Eintragsbezeichner kann zu einem möglicherweise informativen Abschnitt

des Dokuments gesprungen werden. Dort findet der Benutzer wahrscheinlich mehr als nur ein Vorkommnis, da der Mensch dazu neigt, Daten zu organisieren und zu bündeln.
Ein Problem bei der Suche stellt die OCR-Behandlung des Dokuments dar. Da der Benutzer nicht gezwungen sein soll, von Hand alle denkbaren aus der OCR herrührenden Fehler durchzuprobieren, wird der Levenshtein-Abstand (siehe Kapitel 3.7) zugezogen. Der User darf durch einen Regler selbst bestimmen, wie groß dieser Abstand zwischen seinem Suchbegriff und der Fundstelle sein darf. Um möglichst nachvollziehbare Resultate zu gewährleisten, richtet sich der maximal einzustellende Betrag nach der Länge des eingegebenen Strings. Bei einem Wort, das weniger als vier Zeichen hat, darf gar keine Abweichung eingestellt werden, ansonsten maximal $\lceil \frac{n}{2} \rceil - 1$ bei einer Stringlänge von n.

Eine Schwierigkeit stellte die Implementierung dar. Da die Fundstelle

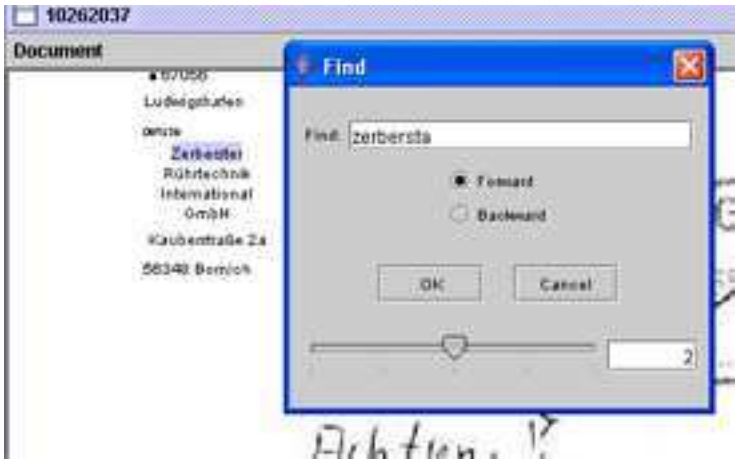


Abbildung 4.7: Suchfunktion

durch Prometheus markiert werden sollte, ist es nötig, auf der Elementebene (siehe Kapitel 3.5.1) zu arbeiten. Bei der Vorwärtssuche muss zunächst das CharacterElement ausfindig gemacht werden, in dessen Indexbereich (Start- bis Endindex) sich der Cursor gerade befindet. Ab der Cursorposition bis zum Endindex werden dann alle Teilstrings der Länge *Suchwortlänge - Levenshteinabstand* bis *Suchwortlänge + Levenshteinabstand* gebildet und der Levenshtein-Abstand des Teilstrings mit dem Suchbegriff berechnet. Falls dieser innerhalb des eingestellten Abstands liegt, wird das betroffene Textstück im Dokument markiert, und die Ansicht springt dorthin. Ansonsten werden alle darauffolgenden CharacterElemente

iteriert und mit den Zeichenketten, die sie repräsentieren, genauso verfahren. Wenn bis Dokumentende kein passendes Äquivalent gefunden worden ist, wird dieses mit einem Dialog bekanntgegeben.
Die Rückwärtssuche gestaltet sich ähnlich, nur dass die Teilstrings natürlich vom Elementende, beim ersten Vergleich von der Position des Cursors oder, falls Text im Dokument markiert ist, vom Index des ersten markierten Zeichens − 1 aus gebildet werden. Wenn man die Suche mit einem sich bewegenden Fenster vergleichen würde, würde es sich in diesem Fall nach links bewegen.
Aufgerufen wird die Suchfunktion entweder über den Punkt 'Find...' im 'Document'-Menü des Dokuments oder über das Tastaturkürzel `Strg + f`.

4.4 Anzeige des Originaldokuments

Wie bereits in Kapitel 3.6 erwähnt, kann es durch OCR zu Schreibfehlern kommen, z.T. sogar zu nahezu unleserlichen Interpretationen, falls ein aufzubereitendes Textstück voll von Namen oder Begriffen ist, die nicht im benutzten Wörterbuch auftauchen oder die Grafik zu viel Rauschen enthält. Außerdem kann es vorkommen, dass die Struktur des Geschäftsbriefes, d.h. die Anordnung von Text und Bild, vollkommen unzureichend gelöst wurde.
Um ein Arbeiten mit Prometheus dennoch zu gewährleisten, kann man sich über den Menüpunkt 'Open original document' oder, falls bereits geladen, den Punkt 'Show original document' das Dokument vor OCR direkt in Prometheus anschauen.
Unterstützt sind derzeit zwei Formate: *TIFF* und *Pdf*.

4.4.1 TIFF

Falls im Dateiauswahldialog eine TIFF-Datei gewählt wurde, öffnet sich ein Fenster des Prometheus eigenen Typs OriginalDocumentInternalFrameTiff. Das Fenster selbst scrollt automatisch zur aktuellen Seite des zugehörigen Dokuments, so dass ein Vergleich umgehend möglich ist.
Realisiert wird dies, indem auf Element-Ebene (siehe Kapitel 3.5.1) die Elemente des Dokuments bis zum Element, innerhalb dessen Grenzen sich der Cursor befindet, durchlaufen wird und dabei die Vorkommen von *HR*-Tags (siehe Kapitel 3.2) gezählt werden. Das funktioniert natürlich nur, falls die HTML-Version des Dokuments bei der Überführung in HTML vom OCR-Programm mit diesen Markierungen versehen wurde.
Der *TIFF-Viewer* bietet über Buttons die Option, die Grafik zu vergrößern oder zu verkleinern, oder es kann direkt ein Prozentsatz der ursprünglichen

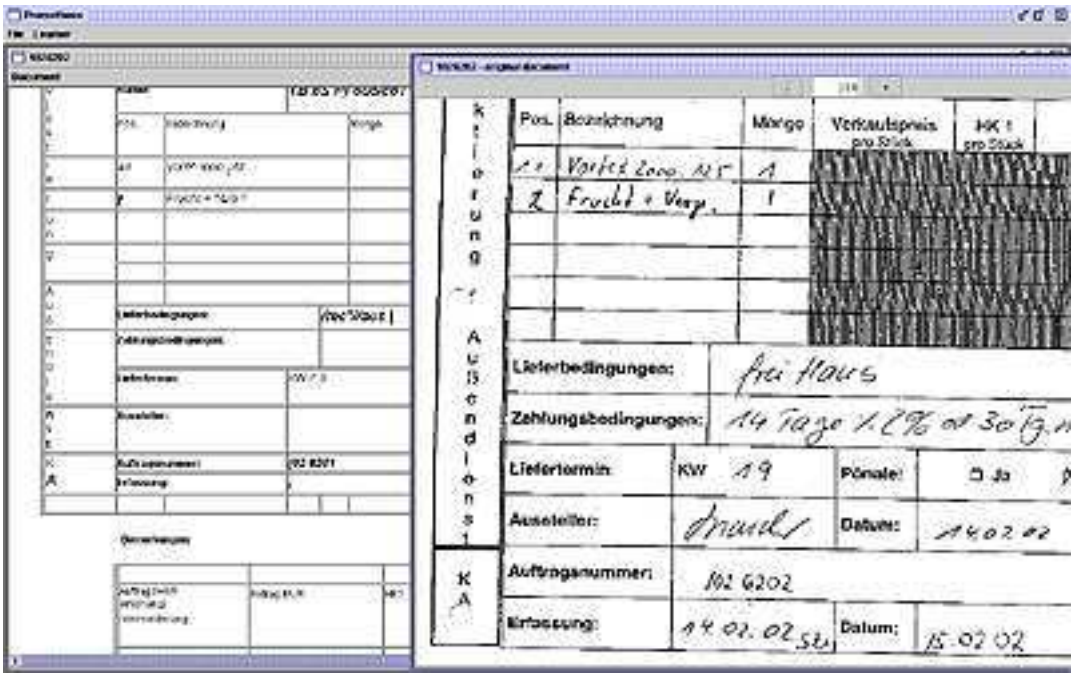


Abbildung 4.8: Links HTML-, rechts Originaldokument

Größe eingegeben werden, auf den das Bild des Dokuments dann skaliert wird.
OriginalDocumentInternalFrameTiff greift dabei auf Klassen der *Java Advanced Imaging (JAI) API* zurück. JAI erweitert die Java Plattform inklusive der Java 2D API durch eine leistungsstarke Bildverarbeitung, die Java Programmen zugänglich gemacht wird. Java Advanced Imaging besteht aus einer Menge von Klassen, die Funktionalitäten bieten, die weit über diejenigen der Java 2D und der Java Foundation Klassen hinaus gehen, wobei sie die Kompatibilität mit diesen wahren [28]. So gibt es über 100 Operationen, wovon die meisten aus Leistungsgründen in Native Code optimiert sind. Einen wichtigen Zusatz stellen die Image Encoder/Decoder (Codec) Klassen dar, die für eine Reihe von beliebten Dateiformaten implementiert worden sind. Zu diesen unterstützten Formaten zählen: BMP, GIF (nur Decoder), FlashPix (nur Decoder), JPEG, PNG, PNM, TIFF, und WBMP.

4.4.2 Pdf

Bei der Auswahl einer Pdf-Datei öffnet sich ein internes zu PROMETHEUS gehörendes Acrobat-Reader-Fenster, das das Originaldokument anzeigt. Ob-

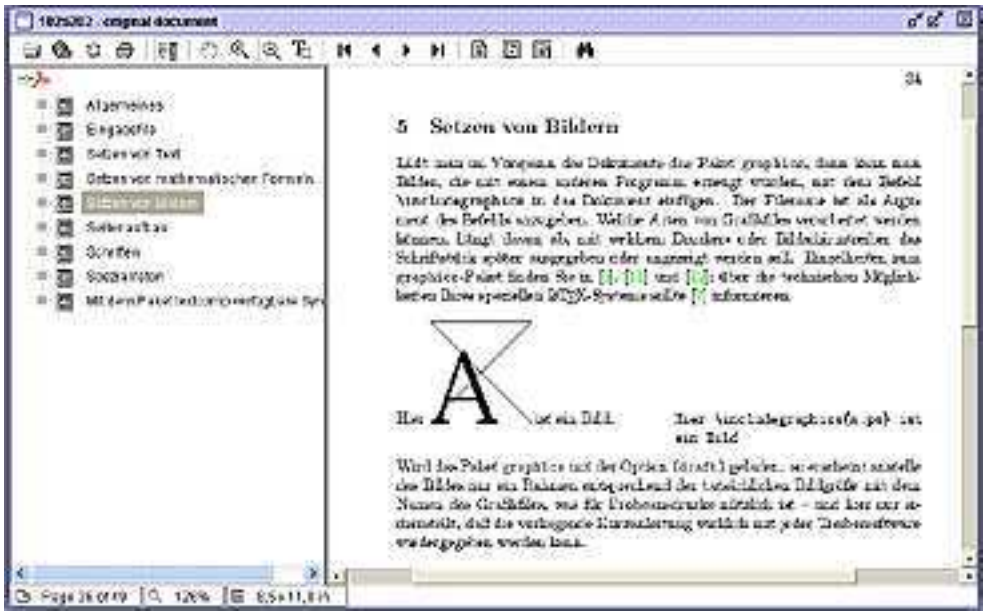


Abbildung 4.9: Anzeige eines Pdf-Dokuments

wohl es von Adobe für diese Zwecke entwickelt wurde, wurde das Projekt in den letzten Jahren fallen gelassen und nicht weiter fortgeführt. So ist dieser Reader, der als Java-Bean verfügbar war, ziemlich fehleranfällig, was glücklicherweise bei PROMETHEUS nicht schwer ins Gewicht fällt. Außerdem verfügt er bei weitem nicht über die Funktionsvielfalt, die in seiner API beschrieben und angepriesen wird. Deswegen lässt es sich nicht realisieren, beim Öffnen dieses Fensters auf die aktuelle Seite zu springen wie etwa beim *TIFF-Viewer*. Zumindest erlaubt es das Bean, dass der Benutzer selbst per Mausklick Seitenwechsel vollführen kann, so dass es seinen Zweck grundsätzlich erfüllt.

4.5 Excel-Anbindung

Um extrahierte Daten weiterverarbeiten zu können bietet PROMETHEUS eine Excel-Schnittstelle. Durch Auswahl des entsprechenden Menüpunkts wird ein Excel-Sheet aus allen Passagen und Einträgen erstellt, bei denen mindestens ein Vorkommnis verzeichnet ist. Dieses wird auch sofort in Excel, sofern installiert, angezeigt; die Tabellenkalkulation wird von PROMETHEUS gestartet. Das Sheet weist dabei dieselbe Struktur wie das PROMETHEUS-Formular auf: Vorkommnisse sind ihren Einträgen untergeordnet und Einträge erschei-

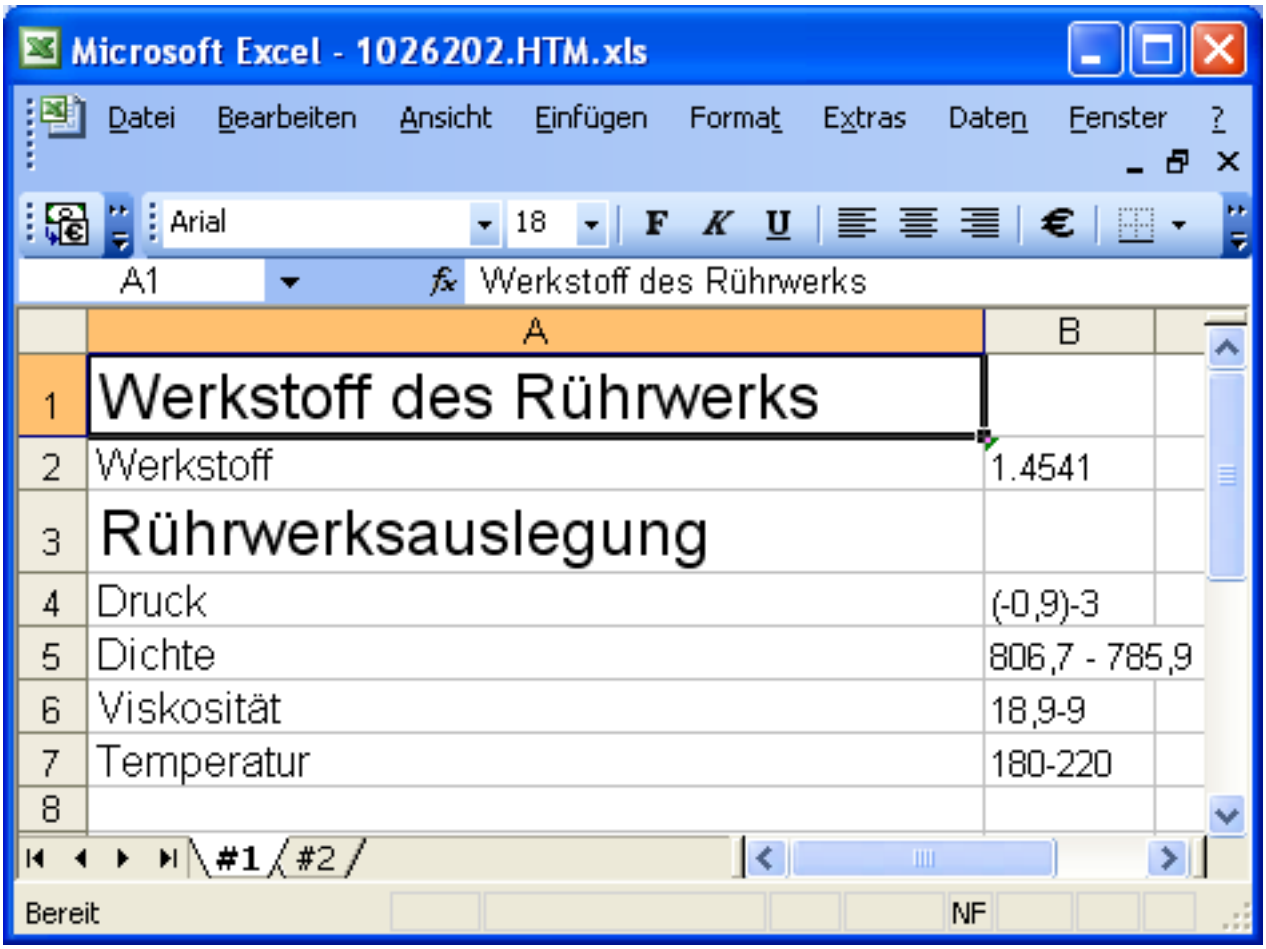


Abbildung 4.10: Mit PROMETHEUS extrahierte Daten nach dem Export in Microsoft Excel

nen unter ihren Passagen. Außerdem werden auch soviele Sheets angelegt, wie es Formulare gibt (siehe Abbildung 4.10).
Realisiert wurde dieses Feature mit *Jarkarta POI-HSSF - Java API To Access Microsoft Excel Format Files*. Das Jarkarta Projekt bietet eine Palette von Open-Source Lösungen für Java an und ist Teil der Apache Software Foundation (ASF), die einen gemeinschaftlichen, einvernehmlichen Entwicklungsprozess unter einer offenen Software Lizenz unterstützt.
Das POI-Projekt besteht aus Java-APIs zum Erstellen und Bearbeiten von Dateiformaten, die auf dem Microsoft-Dateiformat »OLE-2 Compound Document« beruhen. Dateien in diesem Format sind unter anderem die meisten Microsoft-Office-Dateien, wie zum Beispiel Excel- und Word-Dateien.

Zu diesen APIs gehört *HSSF*, wobei es sich um die pure Java Implementierung des Excel '97(-2002) Dateiformats handelt. HSSF stellt Mittel und Wege zur Verfügung, um XLS Spreadsheets zu erstellen, modifizieren, lesen und zu schreiben. So lassen sich unter Java recht einfach Arbeitsmappen, Tabellen und Zellen kreieren, man kann sie ausrichten, mit verschiedenen Farben versehen, den Font einstellen und schließlich füllen.
[29]

4.6 Einstellen des Levenshtein-Abstands

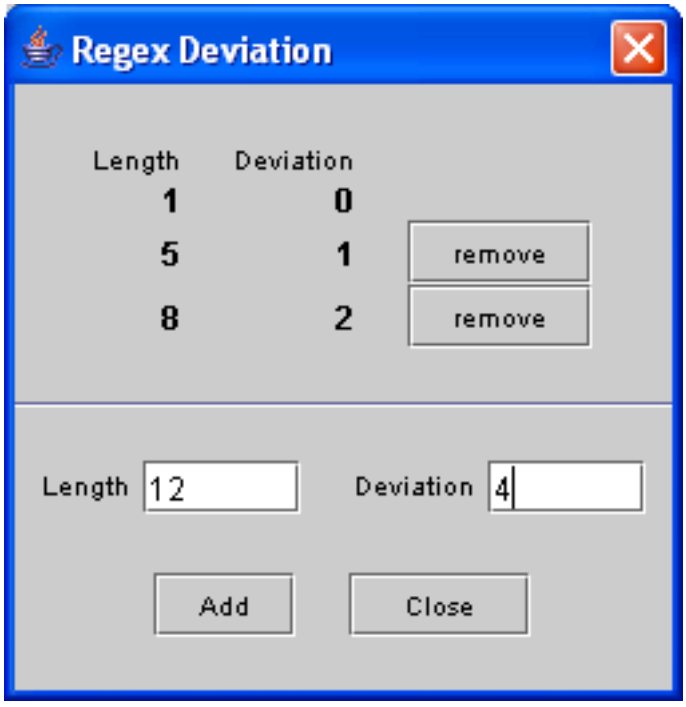


Abbildung 4.11: Dialog zum Einstellen des Levenshtein-Abstands

Die beiden hierauf folgenden Kapitel 4.7 und 4.8 beschreiben Prozesse, die während ihrer Ausführung immer wieder einen Passagen- oder Eintragsbezeichner bzw. eines seiner Synonyme darauf untersuchen, ob der Levenshtein-Abstand zwischen ihm und einer anderen Zeichenkette innerhalb eines bestimmten Wertes liegt. Dieses Kapitel soll vorab erläutern, wie sich dieser Wert in PROMETHEUS einstellen lässt.
Über den Menüpunkt 'Levenshtein deviation' des 'Learner'- Menüs kann ein

Dialog aufgerufen werden (Abbildung 4.11). Hier können maximal zu erlaubende Levenshtein-Abstände in Abhängigkeit zur Wortlänge des Bezeichners eingetragen werden. Diese werden in der Anzeige an die richtige Position einsortiert und ggf. überflüssige Einträge gelöscht. Falls beispielsweise schon das Paar
`Länge: 12  Abstand: 4`
existiert, und der Benutzer
`Länge: 10  Abstand: 4`
eingibt, dann wird natürlich der erstgenannte Eintrag überflüssig und deswegen entfernt. Bis auf den ersten kann man diese auch jederzeit mit dem daneben liegenden 'remove'-Button löschen.

4.7 Automatisches Auszeichnen mit Hilfe des Versionenraums

Das automatische Auffinden und Markieren von Vorkommnissen ist eine weitere Komponente von PROMETHEUS, die den Arbeitsprozess vereinfacht und beschleunigt. Dazu gibt es zwei Ansätze, zum einen eine semi-automatische Lernmethode und eine Variante, die mit regulären Ausdrücken arbeitet, die im nächsten Kapitel vorgestellt wird.
Der Lerner basiert auf dem *Candidate-Elimination Algorithmus*, ein Verfahren, das den Vorteil hat, dass die Datenstruktur , nämlich der *Versionenraum* (s.u.), auf dem der Klassifizierer seine Entscheidungen basiert, so ausgelegt ist, dass er sich ausgesprochen schnell, effizient und inkrementell an die Eingabe einer neuen Instanz anpasst. Dabei reichen schon wenige Beispiele aus, um Resultate zu erzielen, was diesen Algorithmus ob der geringen Menge an zur Verfügung gestellten Beispieldokumenten gerade so attraktiv macht.
Der nächste Abschnitt soll zunächst die Theorie, größtenteils nach [30], erläutern.

4.7.1 Theorie

Es geht bei der folgenden Lernmethode darum, ein *Konzept* zu lernen, wobei festzuhalten bleibt, dass der Begriff Konzept von der linguistischen Seite her sehr unscharf ist. Die Definition des *Concise Oxford Dictionary*([31]) besagt z.B.:

> **concept** *n.* **1** a general notion; an abstract idea (*the concept of evolution*). **2** colloq. an idea or invention to help sell or publicize a commodity (*a new concept in swimwear*). **3** *Philos.* an idea or

> mental picture of a group or class of objects formed by combining all their aspects.

Man muss feststellen, dass sich die Vorstellung eines Konzepts nicht exakt definieren lässt, und es eher philosophischen Überlegungen darüber bedarf, was ein Konzept ist und was nicht [32, Kapitel 2].
Ein Konzept ist also eine abstrakte Idee, die auf eine bestimmte Menge von Objekten zutrifft. Dabei lässt es sich jedoch für jedes beliebige Objekt sagen, ob es diesem Konzept angehört, was einer booleschen Funktion entspricht. Beim Konzeptlernen geht es genau um das Auffinden dieser Funktion, die über eine Menge von Instanzen X definiert ist. Diese sind durch Attribute beschrieben, und man spricht von einem positiven Beispiel, wenn es zum Zielkonzept $c : X \to \{0, 1\}$ gehört, d.h. $c(x) = 1$, und von einem negativen, wenn $c(x) = 0$.
Ein Konzept könnte z.B. "Tage, an dem sich mein Kumpel Peter gern ein Bier zuviel genehmigt" sein. Beispiele dazu können Tabelle 4.1 entnommen werden. Ziel ist es nun basierend auf den Attributen zu lernen, wie das Ergebnis an einem beliebigen Tag ausfällt.
Der Lerner soll also anhand der Trainingsbeispiele schätzen, d.h. Hypothesen darüber anstellen, wie c aussieht, um schlußendlich alle x aus X richtig klassifizieren zu können. All diese Hypothesen H stellen jeweils eine boolesche Funktion über X dar, d.h. $h : X \to \{0, 1\}$. Gefunden werden soll also diejenige Hypothese h, für die gilt $h(x) = c(x)$ für alle x in X.
Wie H selbst nun aussieht, hängt von dessen Repräsentation ab. So kann beispielsweise (wie in PROMETHEUS) jede Hypothese aus einer simplen Konjunktion von Beschränkungen auf ein jedes Instanzattribut sein. Im obigen Beispiel würde eine Hypothese also ein Vektor sein, der die Werte der fünf Attribute spezifiziert. Für jedes Attribut kann diese Beschränkung entweder "?" (jeder Wert ist zulässig), " ∅" (kein Wert ist zulässig) oder ein spezifischer Wert sein (z.B. *Gefüllt*). Eine mögliche Hypothese sähe also so aus:

$$\langle ?,\ ?,\ ?,\ ?,\ \textit{Geöffnet} \rangle$$

Das Aussehen einer Hypothese wird selbstverständlich auch davon beeinflusst, welche Attribute man überhaupt wählt, um eine Instanz zu beschreiben. So wäre es schließlich im obigen Beispiel auch möglich gewesen statt des Attributs *Wetter* ein Attribut *Fußballübertragung* oder *Karaokeabend* zu nehmen. Eine sorgfältige Auswahl ist also unerlässlich.
All diese Entscheidungen können letztendlich darüber entscheiden, ob die "richtige" Hypothese h überhaupt in H existiert.

Beispiel	*Wetter*	*Wochentag*	*Frau*	*Geldbörse*	*Kneipe*	*zuviel Bier*
1	Warm	Dienstag	Ärgerlich	Gefüllt	Geöffnet	Ja
2	Kalt	Samstag	Sauer	Leer	Geöffnet	Ja
3	Regen	Donnerstag	Ungemütlich	Gefüllt	Geschlossen	Nein

Tabelle 4.1: Positive und negative Instanzen, d.h. Tage, zum Zielkonzept *zuviel Bier*

4.7.1.1 Definitionen

Das Lernen eines Konzepts wird als eine Suche im Hypothesenraum gesehen. Um diese zu organisieren, werden zunächst Vergleichsoperatoren definiert, die Beziehungen zwischen zwei Hypothesen ausdrücken.

- Seien h_j und h_k zwei boolesche Funktionen über X. Dann ist h_j **more_general_than_or_equal_to** h_k (geschrieben $h_j \geq_g h_k$) dann, wenn $(\forall x \quad \varepsilon \quad X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$
- h_j ist (strictly) **more_general_than** h_k (geschrieben $h_j >_g h_k$) dann, wenn $(h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j)$
- h_j ist **more_specific_than** h_k, wenn $h_k >_g h_j$

Formell definiert die $\geq_g$ Relation eine partielle Ordnung über den Hypothesenraum H (die Relation ist reflexiv, antisymmetrisch und transitiv). Die Struktur ist nur partiell, da es Hypothesenpaare h_1 und h_2 geben kann, dass $h_1 \not\geq_g h_2$ und $h_2 \not\geq_g h_1$.

Zusätzlich benötigt die Beschreibung des Algorithmus noch einige Definitionen.

- eine Hypothese h ist **konsistent** mit einer Menge von Trainingsbeispielen D, wenn $h(x) = c(x)$ für jedes Beispiel $\langle x, c(x)\rangle$ in D.

$$Consistent(h, D) \equiv (\forall \langle x, c(x)\rangle \varepsilon D) \quad h(x) = c(x)$$

- Der **Versionenraum**, der in Bezugnahme auf den Hypothesenraum H und den Trainingsbeispielen D mit $VS_{H,D}$ bezeichnet wird, ist eine Untermenge von H bestehend aus denjenigen Hypothesen, die mit den Trainingsbeispielen D konsistent sind.

$$VS_{H,D} \equiv \{h \varepsilon H | Consistent(h, D)\}$$

- Die **allgemeine Grenze** G ist die Menge der allgemeinsten Elemente von H, die konsistent mit D sind.

  $G \equiv \{g \varepsilon H | Consistent(g, D) \wedge (\nexists g' \varepsilon H)[(g' >_g g) \wedge Consistent(g', D)]\}$

- Die **spezifische Grenze** S ist die Menge der speziellsten Elemente von H, die konsistent mit D sind.

  $S \equiv \{s \varepsilon H | Consistent(s, D) \wedge (\nexists s' \varepsilon H)[(s >_g s') \wedge Consistent(s', D)]\}$

4.7.1.2 Versionenräume und der Candidate-Elimination Algorithmus

Der *Candidate-Elimination Algorithmus* findet alle beschreibbaren Hypothesen, die mit den Trainingsbeispielen konsistent sind, also den *Versionraum*, ohne sie jedoch alle explizit aufzählen zu müssen. Er repräsentiert den Versionenraum durch die Speicherung der allgemeinsten und speziellsten Elemente. Mit diesen beiden Mengen G und S ist es möglich, alle Elemente des Versionenraums aufzulisten, indem man die Hypothesen, die zwischen G und S liegen, mit Hilfe der partiellen Ordnung $<_g$ generiert.

Theorem 4.7.1.2.1 *Versionenraum Repräsentationstheorem.*

Sei X eine beliebige Menge von Instanzen und sei H eine Menge von booleschen Hypothesen, die über X definiert sind. Sei $c : X \rightarrow \{0, 1\}$ ein beliebiges Zielkonzept, das über X definiert ist, und sei D eine beliebige Menge von Trainingsbeispielen $\{\langle x, c(x)\rangle\}$. Für jedes X, H, c und D, dass S und G wohldefiniert sind, gilt

$$VS_{H,D} = \{h \varepsilon H | (\exists s \varepsilon S)(\exists g \varepsilon G)(g \geq_g h \geq_g s)\}$$

(für weiteres siehe [30])

Der Algorithmus fängt damit an, dass er die G Grenze mit der generellsten Hypothese aus H und die S Grenze mit der speziellsten initialisiert.

$$G_0 \leftarrow \{\langle ?, ?, .., ?\rangle\}$$
$$S_0 \leftarrow \{\langle \emptyset, \emptyset, .., \emptyset\rangle\}$$

Damit begrenzen diese beiden Grenzen den gesamten Hypothesenraum H, was bedeutet, dass auch der Versionenraum alle Hypothesen beinhaltet. Mit jedem neuen Trainingsbeispiel werden die Mengen der G und S Grenze spezialisiert bzw. generalisiert, was jede Hypothese, die nicht konsistent mit dem Beispiel ist, aus dem Versionenraum entfernt.
Der Vorteil dieses Vorgehens ist, dass man tatsächlich zu jedem Zeitpunkt ein Ergebnis in Form der Hypothesen innerhalb des Versionenraums hat.

4.7.1.3 Algorithmus

Initialisiere G mit den allgemeinsten Hypothesen in $H \quad G_0 \leftarrow \{\langle ?, ?, .., ? \rangle\}$
Initialisiere S mit den speziellsten Hypothesen in $H \quad S_0 \leftarrow \{\langle \emptyset, \emptyset, .., \emptyset \rangle\}$
Für jedes Trainingsbeispiel d:

- Falls d ein positives Beispiel ist:
  - Entferne aus G jede mit d inkonsistente Hypothese
  - Für jede Hypothese s aus S, die nicht mit d konsistent ist:
    - Entferne s aus S
    - Füge S alle minimalen Verallgemeinerungen h von s hinzu, so dass
      - h konsistent mit d ist, und irgendeine Hypothese aus G allgemeiner als h ist.
    - Entferne aus S alle Hypothesen, die allgemeiner sind als irgendeine andere aus S.
- Falls d ein negatives Beispiel ist:
  - Entferne aus S jede mit d inkonsistente Hypothese.
  - Für jede Hypothese g aus G, die nicht konsistent mit d ist:
    - Entferne g aus G.
    - Füge G alle minimalen Spezialisierungen h von g hinzu, so dass
      - h konsistent mit d ist, und irgendeine Hypothese aus S spezifischer ist als h.
    - Entferne aus G alle Hypothesen, die spezifischer sind als irgendeine andere Hypothese aus G.

4.7.1.4 Bemerkungen zu Versionenräumen und Candidate-Elimination

Der Versionenraum, der mit Candidate-Elimination gelernt wird, konvergiert zu der Hypothese, die das Zielkonzept korrekt beschreibt, wenn diese in H existiert. Sobald dem Lerner hinreichend genügend Beispiele eingegeben wurden, konvergieren S und G zu einer identischen Hypothese. Jedoch nur unter der Voraussetzung, dass keine Fehler in der Trainingsmenge vorkommen. Unglücklicherweise wird der Algorithmus dann mit Sicherheit die richtige Hypothese aus dem Versionenraum entfernen. Das macht sich insofern bemerkbar, dass S und G zu einem leeren Versionenraum konvergieren, d.h.

keine Hypothese in H war konsistent mit allen gegebenen Beispielen. Dasselbe Phänomen tritt auf, wenn die Beispielmenge zwar korrekt ist, das Zielkonzept jedoch nicht in der Hypothesen Repräsentation beschrieben werden kann.
Für weitere Ausführungen und Beispiele sei dem geneigten Leser wärmstens [30] ans Herz gelegt.

4.7.2 Repräsentation

Das Studium der Beispieldokumente legt den Schluß nahe, dass zu extrahierende Informationen größtenteils in Tabellenform vorliegen oder zumindest in einer Struktur, die ein OCR-Programm als tabellenähnlich interpretiert und mit entsprechendem Markup versieht. Das Aussehen der Instanzen, also die Attribute, die ein (relevantes) Datum (Vorkommnis) beschreiben, legt den Fokus darauf, wie ein Vorkommnis zu "seinem" Eintrag bzw. dessen Bezeichner in einer Tabelle in Verhältnis steht. Durch diese Vorgehensweise wird natürlich auch die Form der Hypothesen festgelegt, die, wie in 4.7.1 bereits erwähnt, aus einer simplen Konjunktion von Beschränkungen auf ein jedes Instanzattribut bestehen.

Bei der Erstellung einer Repräsentation, die den Kern einer tabellarischen Darstellung von Daten bzw. eines einzelnen Tabelleneintrags einfängt, wird die Frage aufgeworfen, was Menschen überhaupt mit Tabellen ausdrücken wollen und, was an dieser Stelle von besonderen Interesse ist, wie sie das bewerkstelligen.
Im Allgemeinen illustriert diese Art der Darstellung Relationen, Kategorisierungen oder Verbindungen zwischen einem Attribut und einem Wert. Es kann sich aber auch um eine schlichte Auflistung handeln, die meistens noch von einer Überschrift begleitet wird, was wiederum eine Beziehung aufzeigt. Dabei obliegt es dem Ersteller, wie die genannten Beziehungen angeordnet werden. Die Anordnung muss dabei einer unterschwelligen Struktur folgen, da andere Menschen schließlich fähig sein sollen, die Informationen, die teilweise eine Bündelung komplexer Sachverhalte/Ausarbeitungen darstellen, schnell aufzunehmen. Struktur hilft dabei.
So fallen meistens alle Einträge einer Spalte/Zeile in eine semantische Kategorie. Werte, die mit einem Attribut assoziiert werden, stehen entweder unter oder rechts von diesem. Gleichzeitig können diese Werte noch mit anderen Begriffen in Beziehung gesetzt werden, indem man diese an den linken Rand der Zeile oder am Kopf der Spalte positioniert. Attribute oder aber jene Ausdrücke, zu denen eine Beziehung erstellt werden soll, können ebenso kategorisiert werden. Da Attribute meistens in einer Spalte

oder Zeile aufgereiht sind, setzt man eine Bezeichnung wiederum an eine ausgezeichnete Stelle, wie z.B. in eine ansonsten leere Zeile, aber in der gleichen Spalte darüber; oder beispielsweise in eine eigentlich leere Spalte daneben.
Um zusätzlich auszudrücken, dass ein Tabelleneintrag von gleicher/unterschiedlicher Art ist, kann man verschiedene Schriftarten wählen, d.h. Attribute tauchen beispielsweise in Fettdruck auf, Werte in einem anderen Font und die Überschrift, unter der die Attribute laufen, in einer anderen Schriftgröße und kursiv.

Beispiel 4.7.1 *Bundesligatabelle*

Abschlusstabelle 2005/06

		Spieltag	*Tore*	*Punkte*	
1.	**Borussia Dortmund**	34	128:8	87	
2.	**MSV Duisburg**	34	83:37	68	
3.	**1. FC Köln**	34	90:45	67	
	⋮				
17.	**FC Bayern München**	34	23:192	18	
18.	**FC Schalke 04**	34	56:45	0	*

* Alle Partien des FC Schalke 04 werden aufgrund von Lizenzverstößen mit 0:2 gewertet.

Die Bundesligatabelle bietet zunächst einmal eine Überschrift, die mit dem Inhalt der Tabelle assoziiert wird; sie wird als solche wahrgenommen wegen ihrer Position und der unterschiedlichen Schriftart und -größe.
In der Tabelle selbst wird eine Zugehörigkeit zur selben semantischen Kategorie dadurch realisiert, dass Werte in derselben Spalte stehen. Diese erfahren durch Worte wie *Spieltag*, *Tore* und *Punkte* eine Kategorisierung, die an ausgezeichneten Stellen und in derselben Spalte erscheinen, nämlich ganz oben. Dass die Kategorisierungen keinen üblichen Wert darstellen, erkennt man auch am unterschiedlichen Schrifttyp und daran, dass die Zeichenkette einem ganz anderen Muster unterliegt (Buchstaben $\leftrightarrow$ Zahlen). Diese Worte werden außerdem zu Bezeichnern von Attributen, die den Bundesligavereinen zugeschrieben werden, einfach dadurch, dass diese durch den Fettdruck und ihrer Position fast am linken Rand das vorherrschende Element einer Zeile sind. (Das Weltwissen eines Menschen kommt dem natürlich auch noch zu Gute; man weiß, dass einem Verein diese Attribute zugewiesen werden können.) Die Werte zu diesen Attributen finden sich natürlich in derselben Zeile in der entsprechenden Spalte.

Durch die Struktur, sowohl die semantische als auch die typographische, lassen sich also Tabellenzellen in Verbindung setzen. Die Idee ist, dass Zellen mit ähnlichen bzw. äquivalenten Inhalt ähnliche Eigenschaften haben und sich zu benachbarten Zellen ähnlich verhalten.
Zellen, die potentiell mit einer zu untersuchenden Zelle in Verbindung gebracht werden können, liegen nach dem vorhergehenden Überlegungen also direkt neben dieser, d.h. darüber, darunter, links oder rechts davon; in derselben Zeile am linken Rand der Tabelle; am Kopf derselben Spalte, oder am linken oberen Rand (ggf. eine Überschrift), siehe Abbildung 4.12 für eine visuelle Verdeutlichung. Die Verbindung kann nun so aussehen, dass

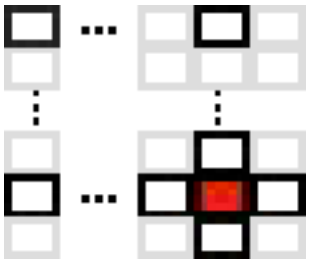


Abbildung 4.12: Die schwarzen Zellen können theoretisch mit der rot markierten in Verbindung gebracht werden

alle links liegenden Zellen nur Zahlen beinhalten und einen anderen Font haben als der Text der beobachteten Zelle.
Da es sich bei PROMETHEUS in seiner jetzigen Form um ein Werkzeug zur Informationsextraktion von eher technischen Daten handelt, sind aufgrund deren Eigenarten vornehmlich Attribut-Wert-Paare zu erwarten, d.h., um in der Terminologie von PROMETHEUS zu bleiben, Einträge bzw. Eintragsbezeichner und Vorkommnisse, weshalb der Schluss nahe liegt, die Distanz, in der sie zueinander stehen, festzuhalten. Dasselbe gilt für die Passage bzw. Passagenbezeichner und Vorkommnisse.
So ergeben sich 116 potenzielle Attribute für eine Tabellenzelle. Diese setzen sich wie folgt zusammen: Da ein Attributvektor bzw. das Vorkommnis, das er beschreibt, mit einem Eintrag assoziiert ist, der auf der anderen Seite einer Passage untergeordnet ist, beschäftigen sich die ersten Attribute hiermit. Es werden die absoluten Positionen von Passagen- und Eintragsbezeichner, d.h. Zeile und Spalte, innerhalb der aktuellen Tabelle festgehalten. Hinzu kommen die relativen Distanzen zum Vorkommnis (d.h. zu dessen Zelle), ebenso in Zeilen und Spalten.
Darauf folgen Eigenschaften von Zellen. Zunächst wird die erste Zelle einer Tabelle, also diejenige, die sich am linken oberen Rand befindet,

begutachtet. Sie wird mit drei Attributen beschrieben. Dabei handelt es sich um die Art ihres Inhalts (siehe Kapitel 4.7.2.1), um einen Vergleich der Font-Familien, mit denen der Text der Vorkommnis-Zelle und eben jener Zelle dargestellt wird, und um einen Vergleich der Font-Größen. Dabei kann das Font-Familien-Attribut die Werte {?, gleich, unterschiedlich} und das Font-Größen Merkmal {?, gleich, größer, kleiner} annehmen.
Komplettiert werden diese durch fünf Sets von Attributen, die auf die gleiche Weise nacheinander die Eigenschaften der Zelle des Vorkommnisses, dessen oberen, unteren, linken sowie rechten Nachbarn erfassen. Jedes dieser Sets hat zunächst ein Merkmal, das Auskunft darüber gibt, ob die betreffende Zelle am oberen oder unteren Rand der Tabelle bzw. weder noch liegt. Ein ähnliches Attribut beschreibt, ob sich die relevante Zelle an der linken bzw. rechten Kante oder aber in der Mitte befindet. Außerdem gibt es ein Attribut, das Aussagen über die Art des Inhalts macht (siehe Kapitel 4.7.2.1). Daraufhin folgen jeweils die Attribute 'Art des Inhalts', 'Vergleich mit der Font Familie' und 'Vergleich mit der Font-Größe' für die Zellen, die gemessen an der beobachteten Zelle darüber, darunter, links davon, rechts davon, in der ersten Zeile, gleiche Spalte und in derselben Zeile, gleiche Spalte liegen.
Basierend auf Experimenten können von dieser Menge von Attributen nicht aussagekräftige Merkmale entfernt werden.
Um die Auswahl zu erleichtern gibt es in PROMETHEUS die Möglichkeit, Attribute für den Lerner nicht zuzulassen, oder aber, falls schon gelernt, bei der Klassifizierung nicht mehr zu berücksichtigen. Dies erfolgt selbstverständlich über eine grafische Benutzeroberfläche. Diese zeigt zunächst alle möglichen Attribute an, inklusive Beschreibungen und Piktogrammen, die bei der Orientierung helfen sollen (siehe Abbildung 4.13). Ein Piktogramm enthält acht Kästchen, jedes steht für eines der oben angeführten Zellen. Im Allgemeinen ist mit der Zelle in der Mitte, diejenige des Vorkommnisses gemeint (, es sei denn eine andere Zelle ist grün eingefärbt). Eine rot ausgefüllte Zelle bedeutet, dass sich das Attribut auf diese Zelle bezieht und Vergleiche von ihr ausgehen. Bei Piktogrammen mit grün markierten Zellen ist die Zelle im Zentrum nun diejenige Zelle, welche im vorhergehenden Piktogramm durch die rote Zelle markiert ist, die Position der grünen ist also relativ zu dieser Zelle zu sehen. Diese Piktogramme zeigen an, dass das jeweilige Attribut aus einem Vergleich zwischen der roten und der grünen Zelle herrührt. In Abbildung 4.14 wird beispielsweise ein Vergleich zwischen den Font-Familien der Zelle, die links neben der Vorkommnis-Zelle liegt, und derjenigen, die sich über besagter linken Zelle befindet, gezogen.
In der Oberfläche kann man nun Merkmale durch Knopfdruck ausschließen, die sodann mit der Farbe Rot unterlegt werden. Diese lassen sich durch

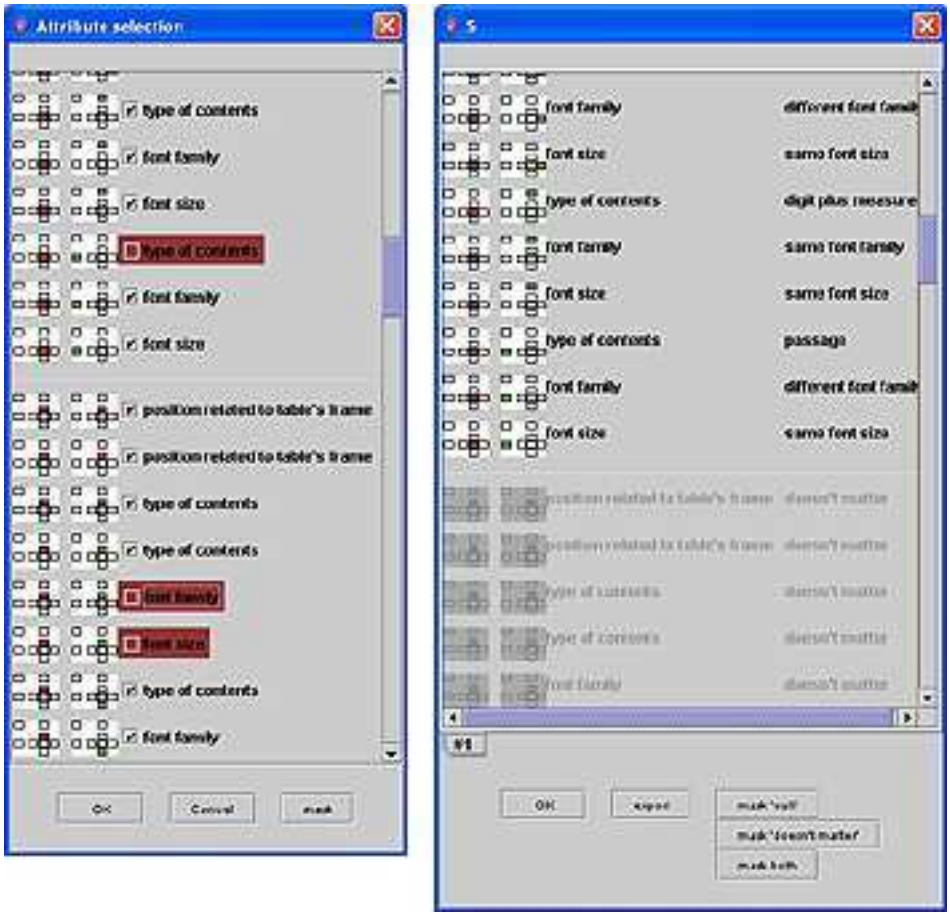


Abbildung 4.13: Links der Dialog zur Attributauswahl, rechts die Anzeige der S Grenze des Versionenraums

die 'mask'-Funktion komfortabel ausblenden, um einen besseren Überblick über die überhaupt eingesetzten Attribute zu haben. Nun werden dem Lerner Beispiele übergeben, die folglich den Versionenraum verändern. Den Zustand dessen kann man sich in PROMETHEUS auch ständig über einen Menüpunkt ansehen. Dieser Dialog zeigt nicht nur die Attribute mit ihren aktuellen Beschränkungen der S und G Schranke an, sondern gesteht dem Benutzer zu, "unwichtige" Merkmale auszublenden. Damit sind Attribute gemeint, dessen Beschränkungen in der S Grenze den Wert '?' angenommen haben oder bei 'Ø' geblieben sind. Praktischerweise lassen sich diese Erkenntnisse durch einen Klick auf den 'Export'-Button in den Attributauswahl Dialog übernehmen, so dass diese Merkmale ausgeschlossen werden. Um verschiedene Variationen ausprobieren zu können, lässt sich

Abbildung 4.14: Ein Attribut im Dialog der Attributauswahl

sowohl die Attributauswahl als auch der aktuelle Versionenraum abspeichern und bei Bedarf wieder laden. Unterstützend kommt hinzu, dass man sich den Attributvektor eines gerade angezeigten Vorkommnisses in PROMETHEUS anzeigen lassen kann, indem man mit der rechten Maustaste auf den entsprechenden Eintrags-Button drückt. Die Ansicht entspricht in etwa der des Dialogs, der die Grenzen des Versionenraums anzeigt.
Die Versuche ergeben, dass lediglich fünf Attribute entscheidend sind, nämlich der Zellen- sowie Spaltenabstand zum entsprechenden Eintragsbezeichner, die Art des Inhalts der Vorkommnis-Zelle, die Art des Inhalts der links daneben liegenden Zelle und die Art des Inhalts der äußerst links liegenden Zelle.
Dieses Ergebnis überrascht wenig, spiegelt es doch in etwa die einfachste Möglichkeit wieder, wie man intuitiv Etwas mit Etwas anderem in Schriftform in Verbindung setzt: man schreibt es nebeneinander.

4.7.2.1 Attribute im Detail

Wie in Kapitel 4.7.1 gezeigt, ist der Versionenraum in hierarchischer Form organisiert. Deswegen ist es angebracht, die Attributwerte der verwendeten Merkmal ebenso in einer Hierarchie anzulegen, damit sich mit ihnen die Relation $\geq_g$ realisieren lässt. So wird bei der Verallgemeinerung/Spezialisierung der Hypothesen der Versionenraum-Grenzen bei jeder einzelnen Beschränkung, wobei es sich schließlich um einen dieser Attributwerte handelt (außer ? und $\emptyset$), die Hierarchie der Werte herangezogen und in der entsprechenden Richtung durchlaufen.
Wie gesehen gibt es lediglich zwei Typen von Attributen. Als erstes wäre da die Art des Inhalts zu nennen. Dessen Werte sind wie in Abbildung 4.15 angeordnet.
Angenommen, der Lerner wird mit einem positiven Beispiel konfrontiert, dessen Attributvektor diese Form besitzt: <?,?, passage, digit, text> und eine Hypothese h aus der S Grenze wie folgt aussieht: <?,?, entry, digit, text>. Dann wird die dritte Beschränkung von h auf *passage or entry* generalisiert, so dass aus h die minimale Generalisierung <?,?, passage or entry, digit, text> wird.

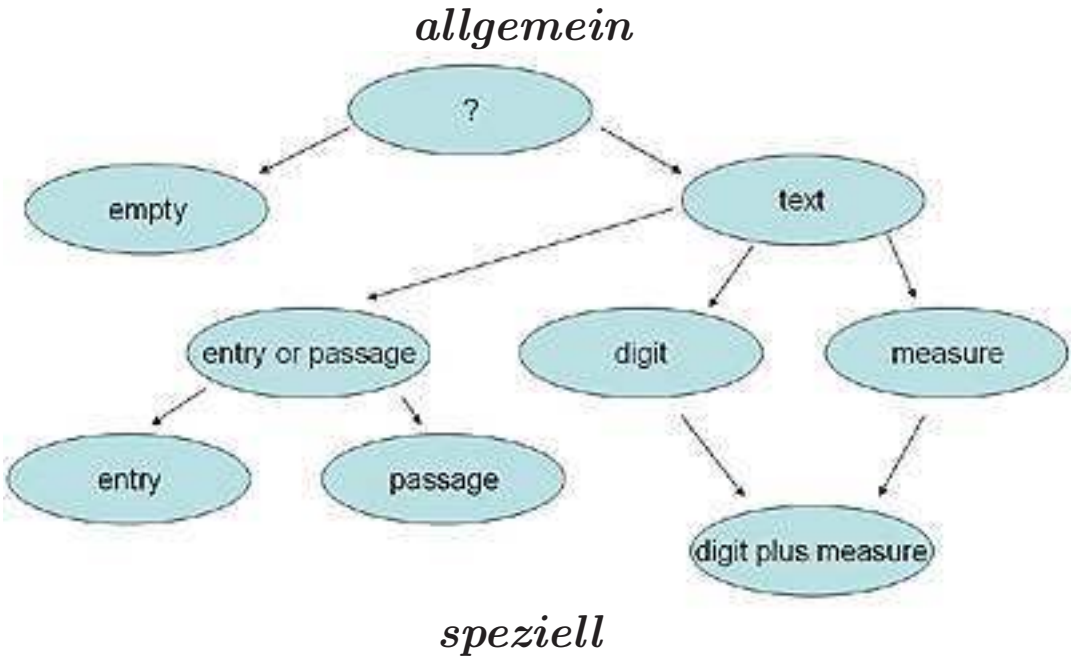


Abbildung 4.15: Hierarchie der Werte des Inhalt-Attributs

Falls beispielsweise das besagte 3. Attribut in d den Wert *text* hat und h an dieser Stelle *measure or digit* dann ergeben sich zwei minimale Generalisierungen <?,?, digit, digit, text> und <?,?, measure, digit, text>.
Bei einem negativen Beispiel d =<..., digit or measure, ...> und einer Hypothese g aus G mit den Beschränkungen <..., text, ...> entstehen die minimalen Spezialisierungen <..., digit, ...> und <..., measure, ...>.
Selbstverständlich werden bei mehreren unterschiedlichen Merkmalen alle möglichen Kombinationen berechnet.

4.7.2.2 Intervallattribut

Die Distanz-Attribute, also einmal der Zeilenabstand von Vorkommnis zum jeweiligen Eintrag und der entsprechende Spaltenabstand, sind als Intervalle abgelegt. Bei Instanzen umfassen diese Intervalle nur einen Wert, bei den Beschränkungen auf die Attribute innerhalb einer Hypothese im Hypothesenraum können sie von $-\infty$ bis $+\infty$ reichen. Diese Intervalle besagen, in was für einem Zellenabstand der Eintragsbezeichner auftaucht; im Zeilenabstandsattribut bedeutet ein negativer Wert n, dass er in $|n|$ Zellen über der Vorkommnis-Zelle liegt, ein positiver Wert n, dass er sich n Zellen darunter befindet. Beim Spaltenabstand steht ein negativer Wert n dafür, dass sich

der Eintragsbezeichner $|n|$ Zellen links befindet und ein positiver, dass er n Schritte rechts zu finden ist. Das Intervall kann sich natürlich über mehrere Positionen erstrecken. So bedeutet zum Beispiel $[-2;3]$, dass sich der zugehörige Eintragsbezeichner in bis zu zwei Zellen Abstand links und bis zu drei rechts des Vorkommnisses auftaucht.

Intervalle eigenen sich für den *Candidate Elimination* Algorithmus insofern,

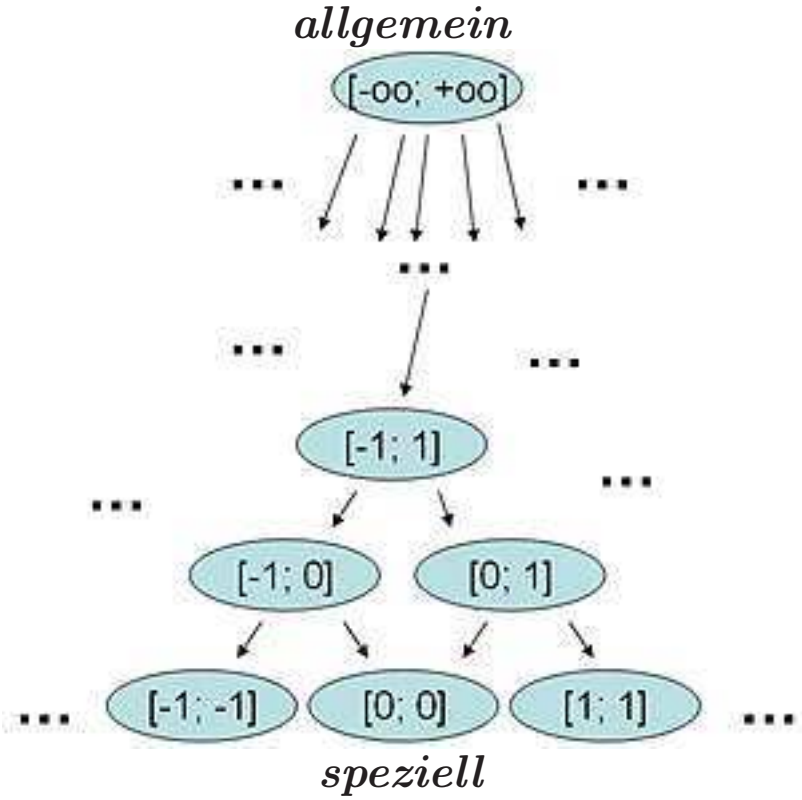


Abbildung 4.16: Hierarchie der Werte eines IntervallAttributs

dass sie eine natürliche Hierarchie mit sich bringen, wobei die einelementigen $[i;i]$ am speziellsten, die Intervalle, die ein anderes beinhalten wie $[i-1;i+2]$ allgemeiner und $[-\infty;+\infty]$ am allgemeinsten sind (siehe Abbildung 4.16). Da diese Hierarchie unendlich ist, kann sie natürlich nicht von Hand im Speicher abgelegt werden. Stattdessen mussten andere Wege gefunden werden, um die Operationen *Spezialisieren* und *Generalisieren* zu realisieren.

Bei Generalisierungen einer Beschränkung eines Intervallattributes b_S, ein Intervall, der Hypothese h aus S, die durch ein positives Beispiel d angestoßen wurde, werden die Grenzen des Intervalls aus b_S derart erweitert, dass das damit resultierende Intervall das Intervall aus d, das ja nur einen

Wert umschließt, beinhaltet. Beispiel: Sei $d =< [-4;-4], \ldots >$ und eine Hypothese h aus $S < [3;8], \ldots >$. Dann wäre die minimale Generalisierung $< [-4;8], \ldots >$.
Bei der Spezialisierung einer Beschränkung eines Intervallattributes b_S der Hypothese h aus G verursacht durch das negative Beispiel d wird das Intervall aus b_S ggf. so geteilt, dass es das aus d nicht mehr abdeckt. Beispiel: Sei $d =< [-4;-4], \ldots >$ und eine Hypothese h aus $G < [-7;8], \ldots >$. Dann wären die minimalen Spezialisierungen $< [-7;-5], \ldots >$ und $< [-3;8], \ldots >$.

4.7.3 Erstellen des Attributvektors

Der Attributvektor eines Vorkommnisses wird in dem Moment erstellt, in dem der Benutzer ein markiertes Textfragment durch Drücken eines Eintrags-Buttons in das Formular übernehmen will. Für diese Berechnung wird nun intern auf dem *Document Model* gearbeitet (siehe Kapitel 3.5.1). Unter Bezugnahme auf die Markierungsposition wird zunächst das *CharacterElement* ermittelt, das die markierten Daten enthält. Daraufhin wird überprüft, ob sich dieses innerhalb einer Tabellenstruktur befindet. Falls nicht, wird lediglich ein simples Text-Vorkommnis abgelegt (siehe Kapitel 4.9). Ansonsten wird die Tabellenposition, Zeilen- und Spaltennummer ermittelt. Da die Repräsentation davon ausgeht, dass sich der Eintragsbezeichner in unmittelbarer Nähe befindet, beginnt innerhalb der Tabelle die Suche nach diesem oder einem seiner Synonyme. Dafür werden die einzelnen Tabellenzellen durchlaufen, bis eine ein Wort enthält, das entweder dem Eintragsbezeichner oder einem Synonym gleicht, wobei alle Vergleiche keine Rücksicht auf Groß- und Kleinschreibung nehmen. Das Wort wird für spätere Verarbeitungszwecke gespeichert (siehe Kapitel 4.9).
Das Iterieren der Tabellenzellen funktioniert durch Ausnutzen der Baumstruktur, in der die Elemente des *Document Models* liegen. Ein *CharacterElement* ist Kind eines `TD`-Elements, dessen Elter ein `TR`-Elements ist, das wiederum einem `TABLE`-Element untergeordnet ist. Hieran sieht man eindeutig die Äquivalenz zur HTML-Repräsentation. Glücklicherweise spiegelt auch die Reihenfolge der Kinder die Darstellung wieder; so steht das erste Kind des `TABLE`-Elements, ein `TR`-Element, für die erste Zeile der Tabelle. Die Reihenfolge, in der die einzelnen Zellen nun besucht werden, gestaltet sich wie folgt: Zuerst wird die Zeile, in der das Vorkommnis liegt, durchsucht und zwar ausgehend von der Vorkommnis-Zelle, zunächst nach links bis zum Tabellenrand; danach geht es nach rechts. Diese Prozedur wird für alle Zeilen, die sich über dieser befinden und danach für darunter liegende wiederholt. Falls dann kein Fund vorliegt, versucht PROMETHEUS selbst ein passendes Synonym zu raten, da das Vorkommen eines Eintragsbezeichners für die Re-

präsentation nun einmal essentiell ist. Außerdem ist dieses Vorgehen auch als Service für den Benutzer zu verstehen, da er diese dann nicht von Hand eintragen muss.
In welchen Bereich der Tabelle die Suche nach einem Synonym starten soll, richtet sich nach dem aktuellen Zustand des Versionenraums. Hierfür werden die Intervall-Beschränkungen einer Hypothese aus S herangezogen. Dieser Bereich wird wie oben durchlaufen, d.h. zuerst nach links, dann nach rechts in den Zeilen mit einem Abstand zur Vorkommnis-Zelle ≤ 0, wobei vom größten Wert aus gestartet wird, und dann dasselbe noch einmal in den Zeilen mit Abstand > 0, wobei die Suche bei dem kleinsten Wert beginnt.
Gesucht werden Zeichenketten, die innerhalb eines vorher definierten (siehe Kapitel 4.6) Levenshtein-Abstands mit dem Eintragsbezeichner oder eines seiner Synonyme identisch sind. Wenn dieser String Teil eines Wortes ist, wird das gesamte Wort als neues Synonym eingetragen und die umschließende Zelle als Bezugspunkt für Distanzberechnungen genommen.
Falls nach diesem Vorgehen kein Treffer gelandet worden ist, wird in demselben Bereich nach einem Wort Ausschau gehalten, das mit einem Großbuchstaben beginnt (im Deutschen ein Merkmal von Nomen), jedoch keine Maßeinheit (s.u.) ist, und bei Erfolg ebenso verfahren wie oben. Sofern dies auch keine Resultate birgt, werden diejenigen Zellen aufgesucht, die in der Richtung liegen, die die Beschränkungen der Hypothese der S Grenze tendenziell vorgeben. Ist das Intervall der Beschränkung für das Spaltendistanz-Attribut beispielsweise $[-5; -2]$, werden auch erst die links von der Vorkommnis Zelle liegenden Zellen besucht. Wenn es wiederum kein zufriedenstellendes Ergebnis gibt, wird die ganze Tabelle einer Suche unterzogen. Auch hier wird von der Vorkommnis-Zelle aus gestartet und in dem bekannten Suchmuster vorgegangen.
Grundsätzlich ist die Suchrichtung insofern belastet, dass "links" und "oben" bevorzugt behandelt werden, da diese Anordnung zwischen Eintragsbezeichner und Vorkommnis für Menschen natürlicher scheint.
Mit dem Ergebnis können nun die ersten beiden Attribute angelegt werden. Zunächst das Intervall für den Zeilenabstand, das nur einen Wert abdeckt, nämlich die Differenz zwischen der Eintragsbezeichnerzeilennummer und der Vorkommniszeilennummer. Demgemäß stellt sich auch das zweite Intervallattribut dar, das jedoch den Spaltenabstand beschreibt und das Ergebnis der Differenz aus Eintragsbezeichnerspaltennummer und Vorkommnisspaltennummer enthält. Falls die Tabelle auch unter Berücksichtigung der Levenshtein-Distanz weder einen Eintragsbezeichner oder eine einem Synonym ähnliche Zeichenkette noch ein groß geschriebenes Wort enthält, werden diese Attribute mit '?' belegt.
Die restlichen drei Attribute beschreiben die Art des Inhalts der Vorkommnis-

Zelle, der links daneben liegenden und der Zelle in der ersten Spalte derselben Zeile. Wenn die beiden letztgenannten nicht vorhanden sind, weil das Vorkommnis selbst am Rand der Tabelle liegt, dann erhalten sie den Wert '?'. Um die Art des Inhalts zu bestimmen wird der String der entsprechenden Zelle untersucht. Wenn es sich um eine leere Zelle handelt, dann erhält das Merkmal den Wert `empty`. Falls nicht, wird zunächst überprüft, ob es sich bei der Zeichenkette um ein Passagen- oder Eintragsbezeichner (bzw. ein Synonym) handelt, was zu `passage` oder `entry` führen würde. Danach wird mit Hilfe eines regulären Ausdrucks ermittelt, ob Zahlen und Maßeinheiten im String enthalten sind. Die verwendeten Maßeinheiten m_1, m_2, ..., m_n sind:

m kg g mg Mg t Kt s min h d 1/s Hz A K C cd mol mol/m l N Pa N/m bar J W S

Sie können zusammen mit den folgenden Präfixen p_1, p_2, ..., p_n auftreten:

Y Z E P T G M k h da d c m μ n p f a z y

Der folgende reguläre Ausdruck sieht so aus:

$$\texttt{\textbackslash b(}p_1\texttt{|}p_2\texttt{|...}p_n\texttt{)?(}m_1\texttt{|}m_2\texttt{|...}m_n\texttt{)\textbackslash b}$$

Um herauszufinden, ob Zahlen vorkommen, genügt der Ausdruck:

$$\texttt{\textbackslash d}$$

Falls sowohl Zahlen als auch Maßeinheiten existieren, wird das Attribut mit den Wert `digit plus measure` belegt. Bei Zahlen ohne Maßeinheiten erhält es den Wert `digit` und bei Maßeinheiten ohne begleitende Zahlen `measure`. Wenn beides überhaupt nicht vorhanden ist, dann `text`.

4.7.4 Klassifizierung

Beim automatischen Auszeichnen wird das *Model* des Dokuments (siehe Kapitel 3.5.1) zunächst nach Tabellen durchsucht. Diese werden wortweise auf Vorkommen von Eintragsbezeichnern und Synonymen untersucht. Dabei darf ein String bis zu einer voreingestellten Levenshtein-Distanz (siehe Kapitel 4.6) abweichen. Alle Tabellenzellen werden nun mit all diesen Funden in Beziehung gesetzt, um den bekannten Attributvektor zu erstellen. Dieser wird an den Klassifizierer übergeben. Sobald er als positive Instanz eingeschätzt wird, erstellt Prometheus daraus ein entsprechendes Vorkommnis-Objekt (siehe Kapitel 4.9), zeigt es im entsprechenden Feld des Formulars an und markiert den Text der Zelle in der Farbe des Eintrags.

Dabei wird für all diese Vorkommnisse zusätzlich ein neues Formular erstellt, um die Ergebnisse des automatischen Auszeichnens schnell auf einen Blick zu haben. Die Vorkommnisse können selbstverständlich bei Bedarf auf andere Formulare verschoben werden (siehe Kapitel 4.2 und Abbildung 4.6).
Eine Instanz wird als positiv eingestuft, wenn der Attributvektor die Beschränkungen einer Hypothese h_S der S Grenze des aktuellen Versionenraums erfüllt oder, wenn man seine Werte als Beschränkungen sieht und ihn als Hypothese h_A betrachtet, $h_S \geq_g h_A$ ist.
Ursprünglich basierte diese Entscheidung auf einem Mehrheitsentscheid aller Hypothesen des Versionenraums, wobei zugunsten des Recalls schon bei einer positiven Einschätzung von $\frac{n}{2}$ Hypothesen für positiv votiert wurde. Es gab einige Gründe, die gegen diese Realisierung sprachen. Zu allererst wären da die Intervallattribute zu nennen. Da die Beschränkungen der Hypothese in G zu Anfang $[-\infty;+\infty]$ betragen, ist der Versionenraum unendlich, wobei dieses Problem einfach damit zu lösen ist, an diesen Stellen die maximal zu erzielenden Abstände der Tabelle einzusetzen, aus welcher der zu klassifizierende Attributvektor seine Werte bezieht. D.h. bei einer Tabelle von z.B. 12x12 Zellen, $[-11;+11]$ bei beiden Intervallbeschränkungen. Sofern dem Lerner noch kein negatives Beispiel übergeben wurde, was eher dem Normalfall entspricht (siehe Kapitel 6), gibt es bei beiden relevanten Beschränkungen jeweils nur eine Intervallbeschränkung in allen Hypothesen aus G, i_{G_1} und i_{G_2}, und jeweils eine in den Hypothesen aus S, i_{S_1} und i_{S_2} (siehe Verallgemeinerung/Spezialisierung in Kapitel 4.7.2.2). i_{G_1}, i_{G_2}, i_{S_1} und i_{S_2} sind wie gesehen Intervalle. Jegliche Intervallbeschränkungen B_{I_1} des Versionenraums bezüglich des ersten Attributs (zweites analog) bestehen dann aus allen Intervallen, die i_{S_1} enthalten und von i_{G_1} enthalten werden.
Ein Mehrheitsentscheid bei n Hypothesen des Versionenraums macht jedoch in dieser Konstellation keinen Sinn, da ein einstelliges Intervall, wie es eben bei Instanzen der Fall ist, immer mindestens $\frac{n}{2}$ mal in B_{I_1} enthalten ist.
Beweis. Zunächst wandelt man alle Intervalle in Mengen derjenigen ganzen Zahlen um, die sie umfassen, insbesondere entstehen I_{S_1} aus i_{S_1} und I_{G_1} aus i_{G_1}. Entfernt man nun aus all diesen Mengen alle Zahlen aus I_{S_1}, entsteht die Potenzmenge $\mathcal{P}_1$ der Zahlen $I_{G_1} - I_{S_1}$. Damit enthalten natürlich $\frac{|\mathcal{P}_1|}{2}$ Mengen die Zahl $x \quad \epsilon \quad I_{G_1} - I_{S_1}$.

Somit werden die Intervallattribute (in dieser Konstellation) nie zu einem negativen Entscheid führen, was sich folglich auf die Precision ausschlägt, die durch diesen Ansatz noch schlechter wird, was ob der gemessenen Werte (siehe Kapitel 6) vermieden werden sollte.
Was ebenso dagegen sprach, war die erhöhte Laufzeit. Schließlich müssen

für jede neue Tabelle wegen der Anpassungen der Intervallbeschränkungen alle Hypothesen des Versionenraums neu berechnet werden. Es siegte also Benutzerfreundlichkeit gegen Adäquatheit, da dem Benutzer nicht zugemutet werden soll, dass sein Arbeitsfluss unter Umständen Minuten unterbrochen wird - PROMETHEUS sollte ein interaktives System bleiben.

4.7.5 Benutzergeführte Eingriffe in den Versionenraum

Neben dem Auswählen von positiven und negativen Beispielen kann der Benutzer noch mit anderen Mitteln in das Aussehen des Versionenraums eingreifen.
So besteht die Möglichkeit, den aktuellen Zustand des Versionenraums mit 'Save/Load versionspace' im 'Lerner'-Menü erst abzuspeichern und zu einem späteren Zeitpunkt zu laden. Auf diese Weise lässt sich das Konzept von Suchmustern realisieren.
Da der Versionenraum in dieser Diplomarbeit und auch im Allgemeinen (siehe Kapitel 4.7.1.4) recht anfällig dafür ist, dass er bei einem negativen Beispiel, das aber den Werten des Zielkonzepts sehr nahe kommt, kollabiert, kann man mit 'Remove negative examples' alle in diesem Arbeitsgang als negativ eingeordnete Attributvektoren aus ihm entfernen, und er wird aus den Vektoren der positiven Beispiele dieser Session wieder aufgebaut.
Außerdem lässt er sich mit 'Clear versionenspace' wieder in den Anfangszustand versetzen.

4.8 Regex Auszeichnen

Da es selbstverständlich sein kann, dass relevante Informationen in einem Dokument nicht in Tabellenform vorliegen, und da das Problem des Auffindens wahrscheinlich ohnehin einfacher gelagert ist, wurde das *Regex Auszeichnen* eingeführt. Hier wird der Text des Dokuments mit Hilfe von regulären Ausdrücken untersucht.
Dabei gibt es drei unterschiedliche Vorgehensweisen, die sich nach der vom Benutzer erwarteten Art des Vorkommnisses für einen Eintrag richten. Zu diesem Zweck bleibt ihm die Wahl zwischen einem Zahlenwert, einfachem Text oder um eine vorher definierte Menge von Zeichenketten.
Diese Einstellungen werden im 'Type'-Dialog vorgenommen (siehe Abbildung 4.17). Dieser bietet in der linken oberen Ecke die Typ-Auswahl. Bei den Optionen `catchword` und `digit` gibt es die Möglichkeit, im darunter liegenden Textfeld Stichworte oder Maßeinheiten einzufügen, die bei der Suche

Abbildung 4.17: Dialog zur Auswahl des Typs der Vorkommnisse eines Eintrags

berücksichtigt werden. Durch Markieren von Elementen in der Auflistung zur Rechten und anschließender Betätigung des 'remove'-Buttons können diese wieder entfernt werden.
Das Auffinden von Vorkommnissen wird durch die Menüoption 'Regex tagging' im Menü 'Learner' angestoßen. Daraufhin werden intern alle Eintragsbezeichner (und ihre Synonyme) durchlaufen. Dabei wird zuerst festgestellt, was für eine Option für den aktuellen Eintrag eingestellt worden ist, was den weiteren Fortlauf bestimmt. Alle haben gemein, dass wiederum zunächst auf Element-Ebene (siehe Kapitel 3.5.1) gearbeitet wird, um an zu untersuchende Zeichenketten zu gelangen, um letztendlich korrekte Indizes für die Benutzeroberfläche zu erhalten.
Bei den Einstellungen `digit` und `text` wird im Dokument nach einem String gesucht, der innerhalb eines vorher eingestellten Levenshtein-Abstands (siehe Kapitel 4.6 dem aktuellen Eintragsbezeichener/Synonym gleicht. Wird dieser gefunden, wird bei `digit` überprüft, ob der reguläre Ausdruck

```
\s*:?\s*-?\s*\d+((\.|,)\d+
```

auf die darauf folgenden Zeichen passt. Er bedeutet, dass nach einer beliebige Anzahl von Whitespaces und ggf. einen Doppelpunkt eine Zahlenfolge kommt, die unter Umständen negativ ist und von einem Punkt oder einem Komma unterbrochen wird. Wenn vorher vom Benutzer die Maßeinheiten $m_1, m_2, \ldots, m_n$ eingetragen worden sind, sieht der Ausdruck so aus:

$$\texttt{\textbackslash s*:?}\underbrace{\texttt{(}m_1\texttt{|}m_2\texttt{|...}m_n\texttt{|\textbackslash s)*}}_{Maßeinheit(en)}\underbrace{\texttt{-?\textbackslash s*\textbackslash d+((\textbackslash .|,)\textbackslash d+)?}}_{Zahl}\underbrace{\texttt{(}m_1\texttt{|}m_2\texttt{|...}m_n\texttt{|\textbackslash s)*}}_{Maßeinheit(en)}$$

Dann dürfen vor und nach oben beschriebener Zahl besagte Maßeinheiten ggf. getrennt durch Whitespaces stehen.
Falls es einen Treffer gibt, wird lediglich die Zahl markiert und ein entsprechendes *Regex Vorkommnis* erstellt.
Bei der Option `text` lautet der reguläre Ausdruck, mit dem die folgenden Zeichen des Eintragsbezeichners untersucht werden:

```
(:|\s+)((\.\S)|([^\.|\n|\x0B|\f|\r]))+
```

Hiermit wird jeglicher Text gefunden, der zwischen einem möglichen Doppelpunkt und einem Zeilenumbruch bzw. einem Punkt, auf dem ein Whitespace folgt, liegt. Dieser wird dann markiert.
Bei dem Typ `catchword` wird das Dokument nach Zeichenketten, die den eingetragenen Stichworten unter Berücksichtigung der Einstellungen für die Levenshtein-Distanz (siehe Kapitel 4.6) gleichen, durchsucht, diese bei einem Treffer markiert und natürlich ein Vorkommnis-Objekt angelegt.

4.9 Vorkommnisse

Wie bereits in Kapitel 4.1 erläutert, ist ein *Vorkommnis* der Wert der Informationsextraktion bzw. die Stelle im Text eines Dokuments, die der Anwender markieren würde, um ein relevantes Datum zu kennzeichnen. Beispielsweise wäre dies eine Zahl, die die Dichte eines Werkstoffs darstellt.
Im Grunde gibt es vier Arten von Vorkommnissen, die sich durch ihre Art des Zustandekommens und in ihrer Verwaltung unterscheiden. Für den Benutzer weisen sie jedoch beim Umgang mit ihnen in der grafischen Benutzeroberfläche keine großen Differenzen auf. Alle können ohne Probleme abgespeichert und wieder geladen werden. Auch bei einem Export wie in Kapitel 4.5 werden alle gleich behandelt.

4.9.1 Text-Vorkommnis

Diese Art von Vorkommnissen ist die wohl schlichteste. Es handelt sich dabei einfach um vom Benutzer eingegeben Text. Diesen tippt er in ein *leeres*

Textfeld einer Eintrags-Zeile ein. Dem Benutzer soll hiermit die Gelegenheit eingeräumt werden, Textinformationen zu extrahieren, d.h. Expertenwissen einfließen zu lassen, das ggf. bei der Weiterverarbeitung wichtig ist; schließlich ist die Arbeit mit PROMETHEUS nur ein Schritt unter vielen.
Da hierdurch dieser Text mit keiner konkreten Position im Dokument assoziiert wird, gibt es natürlich auch keinen Attributvektor, der in den Lerner einfließen könnte. Aus diesem Grund sind die Lerner-Buttons auch für diesen Typ nicht benutzbar.

4.9.2 Markiertes Vorkommnis

Das wohl häufigste Vorkommnis-Objekt entsteht, sobald der Benutzer selbst ein Stück Text im Dokument markiert und diesen durch das Aktivieren eines Eintrags-Buttons mit eben diesem Eintrag in Verbindung setzt. Dann wird je nachdem, ob die Option 'Highlight cell' im 'Document'-Menü gesetzt ist, entweder die ganze Zelle mit der entsprechenden Farbe des Eintrags markiert und der gesamte Zellentext im Textfeld übernommen, oder diese Vorgänge werden nur für den markierten String erledigt. Der übernommene Text kann aber selbstverständlich noch von Hand innerhalb des Textfeldes bearbeitet werden. Dabei wird sogar die Markierung entsprechend erweitert, falls sich der eingegebene Text mit dem im Dokument exakt gleicht.
Egal welche Option gewählt ist, wird immer ein Attributvektor wie in Kapitel 4.7.3 beschrieben erstellt, wobei auch der Begriff, auf den sich die Intervallattribute des Vektors beziehen, gespeichert wird, sei es nun der Eintragsbezeichner oder eines seiner Synonyme. Der Attributvektor wird dem Lerner nun als positives Beispiel präsentiert, wenn die Option 'Add as positive example' im 'Document'-Menü gesetzt ist. Dieses Übernehmen wird dadurch gekennzeichnet, dass der 'Positives Beispiel'-Button grün aufleuchtet. Durch Drücken eben jenes Buttons kann man dieses Beispiel aus dem Lerner wieder entfernen. D.h. der Versionenraum wird in denjenigen Zustand gesetzt, in dem er wäre, wenn dieses Beispiel gar nicht aufgetaucht wäre. Dies möchte man beispielsweise durchführen, wenn man nach Begutachtung des Attributvektors (was sich mit einem Rechtsklick auf den Eintrags-Button bewerkstelligen lässt), zu dem Schluss gelangt, dass dieses Beispiel die S Grenze des Versionenraums allzu sehr verallgemeinern würde. Natürlich lässt sich dieses Vorkommnis durch nochmaliges Betätigen des Buttons auch wieder in den Lerner einspeisen.
Beim Löschen eines solchen Vorkommnisses passiert etwas Ähnliches wie beim Deaktivieren des 'Positives Beispiel'-Buttons. Außer dass natürlich der Text aus dem Textfeld gelöscht und, falls vorhanden, zum nächsten Vorkommnis umgeschaltet wird, wird auch hier der betreffende Attributvektor

aus dem Lerner entfernt.
Etwas Interessantes passiert, wenn das Synonym, das für das Vorkommnis verantwortlich zeichnet, d.h. als Bezugspunkt genommen wurde, im Synonym-Dialog des Eintrags, dem das Vorkommnis zugeschrieben worden ist, gelöscht wird. Dann nämlich wird der Attributvektor auch aus dem Lerner entfernt, da er offensichtlich auf falschen Angaben fußt. Zusätzlich wird die zugehörige Markierung im Dokument gelöscht und das Vorkommnis in ein Text-Vorkommnis umgewandelt, da die angezeigten Daten sicherlich relevant sind und auch bei der Weiterverarbeitung zur Verfügung stehen sollen. Mit all diesen Mitteln hat der Benutzer stets Kontrolle über den Lernprozess. Der 'Negatives Beispiel'-Button ist beim *Markierten Vorkommnis* übrigens deaktiviert, denn es besteht kein Grund anzunehmen, dass der Benutzer absichtlich einen falschen Wert in das Formular übernehmen will.

4.9.3 Vorgeschlagenes Vorkommnis

Hierbei handelt es sich um ein vom Lerner vorgeschlagenes Vorkommnis, das beim automatischen Auszeichnen zustande kam. Da dieses logischerweise die S Grenze des Versionenraums nie verändern könnte, ist bei diesem folglich der 'Positives Beispiel'-Button gesperrt. Allerdings kann der Benutzer hier den 'Negatives Beispiel'-Button betätigen, was zur Folge hat, dass der zugehörige Attributvektor dem Lerner als negatives Beispiel zugeführt wird. Falls dadurch ein Widerspruch verursacht wird, kann der Benutzer über einen dann erscheinenden Dialog verfügen, dass der Lerner das Beispiel nicht berücksichtigt. Daraufhin wird das Vorkommnis samt Markierung aus der Ansicht gelöscht.
Wenn das Synonym, das für das Vorkommnis verantwortlich zeichnet, entfernt wird, wird das gesamte Vorkommnis gelöscht, da es ja auf falschen Annahmen basiert.
Auch hier kann der Text selbstverständlich noch von Hand innerhalb des Textfeldes bearbeitet werden. Dabei wird die Markierung entsprechend erweitert, falls sich der eingegebene Text mit dem im Dokument exakt gleicht.

4.9.4 Regex Vorkommnis

Diese Art Vorkommnis entsteht beim *Regex Auszeichnen* (Kapitel 4.8). Da diese Methode eine Alternative zum Auszeichnen mit dem Versionenraum darstellt, gibt es auch hier keinen Attributvektor, weswegen die Lerner-Buttons deaktiviert sind.
Da das *Regex Vorkommnis* auf eine bestimmte Textstelle im Dokument zurückzuführen ist, besitzt es auch eine Markierung an besagter Stelle, die

natürlich ebenso wie der im Textfeld angezeigte Text entfernt wird, falls es zur Löschung des Vorkommnisses kommt.
Bei den Typ-Einstellungen `digit` und `text` (siehe Kapitel 4.8) kann es beim *Regex Auszeichnen* dazu kommen, dass das Vorkommnis aufgrund eines Synonyms eines Eintragsbezeichners zustande kommt. Deswegen wird das gesamte Vorkommnis gelöscht, wenn der Benutzer das besagte Synonym herausnimmt.
Auch hier kann der Text selbstverständlich noch von Hand innerhalb des Textfeldes bearbeitet werden. Dabei wird die Markierung entsprechend erweitert, falls sich der eingegebene Text mit dem im Dokument exakt gleicht.

Kapitel 5

Beispielhafter Arbeitsvorgang

Dieses Kapitel soll an einem realen Dokument beispielhaft zeigen, wie man mit PROMETHEUS arbeitet.

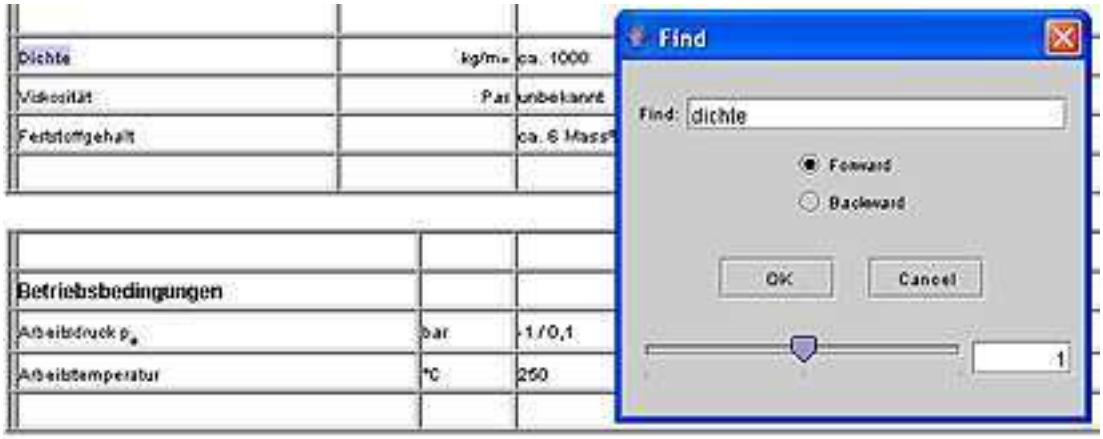
Vorverarbeitung

Zuerst muss der Geschäftsbrief eingescannt werden, wonach das Dokument z.B. als TIFF vorliegt. Diese Grafik-Datei dient als Eingabe für ein OCR-Programm. Als Ausgabe werden verschiedene Formate angeboten. PROMETHEUS erwartet eine HTML-Datei, die auf JavaScript verzichtet, ansonsten aber soviel Markup bietet wie möglich, am besten inklusive der Kennzeichnung von Seitenwechseln, die mit Einfügen von <HR> Tags realisisert werden.

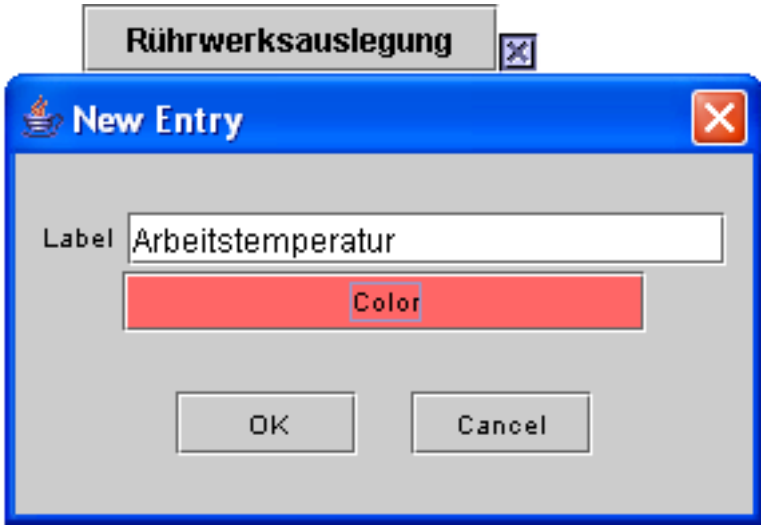
Prometheus

Nach dem Start von PROMETHEUS wird die HTML-Datei geöffnet.

In dieser sucht man jetzt nach dem erwarteten Begriff "dichte". Dieser wird gefunden, woraufhin man den daneben stehenden Wert markiert und durch einen Druck auf den entsprechenden Eintrags-Button übernimmt. Da man durch die Suche in die Gegend von relevanten Informationen gelangt ist, weil diese oft in Tabellen organisiert sind, kann man nun weitere Textteile markieren und eintragen, wie den Wert für die Arbeitstemperatur.



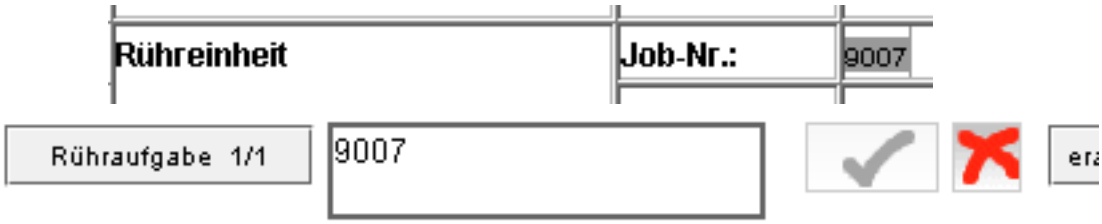
Den im Formular fehlenden Eintrag erstellt man durch Drücken des zugehörigen Passagen-Buttons, hier *Rührwerksauslegung*.



Weil dadurch von PROMETHEUS durch diese beiden Vorgänge gelernt wurde, setzt man nun das automatische Auszeichnen ein. Da in einem voran gegangenen Arbeitsgang bereits das Synonym "Arbeitsdruck" unter dem Eintrag *Druck* vermerkt wurde, wird wie zu erwarten der über der Arbeitstem-

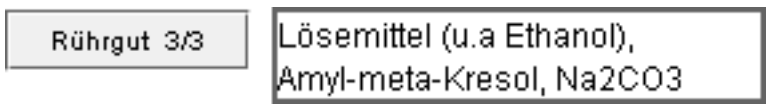
peratur stehende Wert richtig zugeordnet. Mit einem Rechtsklick auf das Vorkommnis-Textfeld kann man diesen in das Arbeitsformular `#1` portieren.

Unglücklicherweise wurde auch ein Wert für den Eintrag *Rührаufgabe* vermerkt.



Ein Blick in die Synonymliste des Eintrags verrät, dass das Wort "Rühreinheit" fälschlicherweise mal vermerkt worden war. Damit ein derart inkorrektes Zuweisen in der Zukunft nicht mehr geschieht, wird das Synonym einfach gelöscht, was auch das Vorkommnis entfernt.
Nun benutzt man das Regex Auszeichnen. Man sieht die gelieferten Vorschläge schnell mit Klicks auf die Eintrags-Buttons durch und stößt z.B. auf wertvolle Resultate für das Rührgut (hervorgerufen durch das Synonym "Medium" und die Typ-Einstellung `text`).

Da durch den Punkt, dem ein Whitespace folgt, nicht der ganze relevante Text übernommen wird, fügt man diesen schnell von Hand ein (mit *copy, paste*).

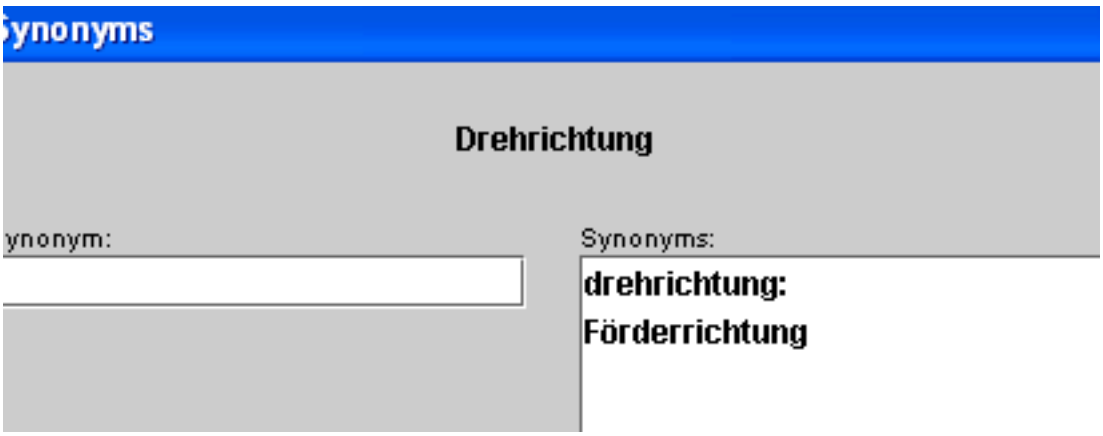


Außerdem ergab diese Art des Auszeichnens einen Wert für die *Rührаufgabe*, nämlich "Homogenisieren", was durch das gleichlautende Stichwort und die Typ-Einstellung `catchwort` zustande kam. Zudem wird noch der Wert "Schiebenabe" für den Eintrag *Befestigung* angezeigt, der ebenfalls von einem Stichwort herrührt und den man wieder von Hand ergänzt. Da bei der

Ansicht von Vorkommnissen stets zu dessen Position in der Dokumentansicht gesprungen wird, landet man dadurch höchstwahrscheinlich an Stellen des Textes, wo sich noch andere, unmarkierte Informationen verbergen. So auch im letzten Fall, denn über der Befestigung findet sich ein Wert für die *Förderrichtung*.

Förderrichtung : vom Antrieb
Befestigung : Schiebenabe in Sterilausführung,
2 Blätter abschraubbar.

Diesen markiert man umgehend und drückt den Button des Eintrags *Drehrichtung*, da einem das Expertenwissen sagt, dass "Förderrichtung" ein Synonym von "Drehrichtung" ist. Ein Blick in den Synonym-Dialog beweist, dass PROMETHEUS das tatsächlich automatisch erkannt hat, so dass das dann schon bei den nächsten Dokumenten (erfreuliche) Auswirkungen auf die Methoden des automatischen Auszeichnens hat.



Alle wichtigen Vorkommnisse werden nun auf das erste Formular geschoben und die restlichen gelöscht.

Abschließend scrollt man nochmal schnell durch den Text, um sicherzustellen, dass einem nichts entgangen ist oder wegen OCR-Fehlern schlichtweg nicht zugänglich war. Tatsächlich stößt man auf die verstümmelte Stelle aus Abbildung 5.1. Das Zuschalten des Originaldokuments sorgt für Aufklärung (Abbildung 5.2). Neben dem abgebildeten Sendebericht sorgten auch einige Textpassagen, die auf dem Kopf stehen, für dieses mangelhafte OCR-Ergebnis.

Da sich aber hier keine nennenswerten Informationen befinden, kann man die Arbeit an diesem Dokument abschließen und die Resultate nach Excel portieren (Abbildung 5.3).

mo	siNaaoaa
i	**'S**
zz^OO	xiaz'an
SZ'LI **60/EO**	**XiaZ'dMV**
TTTATTAT /-1 C) YT JT \T C2	foTJT C YT^^IT^ TTTITYIT k0 A 1 JZ/ I j^1 M^ Lk^> J1k^J
	aSS3H(IVN383N
US9E TS2S 6fr+	**3113XSN3D3D 'HM3Ü**
60TE	**HN 13/3S**
	MO DNÜQN3S
•I> "I• •!■ •!■ •[S SJ. Sf. •I> SJ> SJS «I> J|> «I> •I> «I> ■!^ •t» ^T• •! **# iHoiaaaaaMas	****** ***
**********	

Abbildung 5.1: Missglücktes OCR

```
                    **********************
                    ***   SENDEBERICHT   ***
                    **********************
SENDUNG OK

SE/EM NR                          3108
RUFNR. GEGENSTELLE                      +49 [illegible]
NEBENADRESSE
NAME GEGENSTELLE
ANF. ZEIT                         03/09 17:20
ÜB. ZEIT                          00'43
S.                                   2
ERGEBNIS                          OK
```

Abbildung 5.2: Des Rätsels Lösung

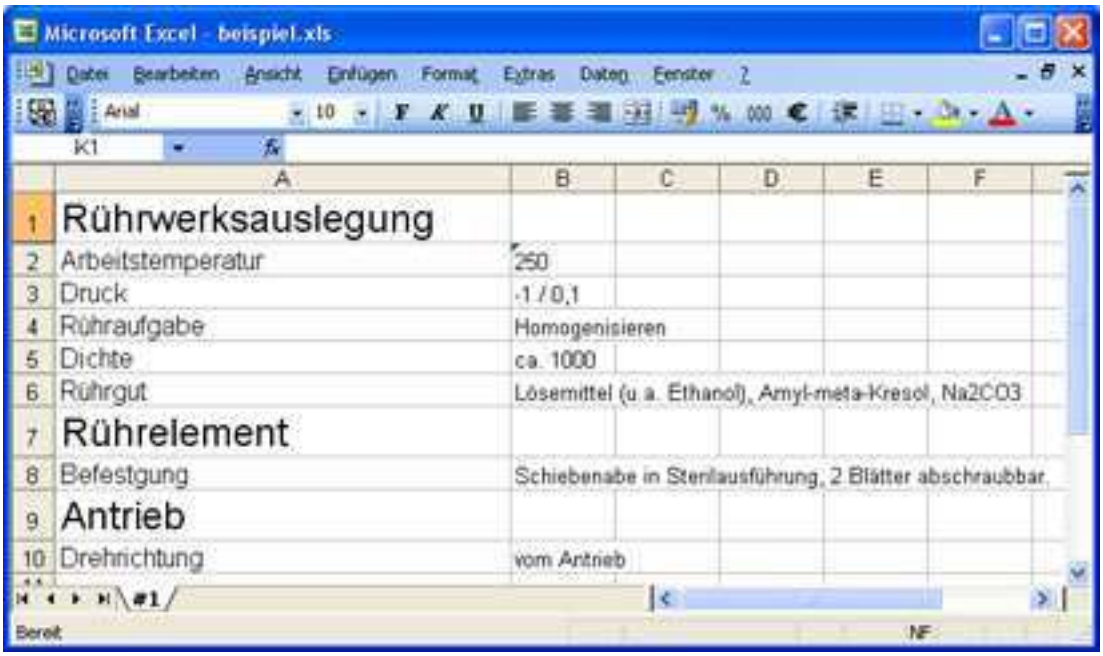


Abbildung 5.3: Resultate des Arbeitsvorgangs

Kapitel 6

Versuche

Automatisches Auszeichnen mit Hilfe des Versionenraums

Die Tests des automatischen Auszeichnens nach Kapitel 4.7 wurden an drei Dokumenten mit insgesamt neunzehn Vorkommnissen durchgeführt. Diese zeichneten sich dadurch aus, dass sie relevante Daten vornehmlich in Tabellen hielten. Ermittelt wurden die *Precision* und *Recall* Werte. Diese sind hier wie folgt belegt:

$$Recall = \frac{Anzahl\ korrekt\ identifizierter\ Vorkommnisse}{Anzahl\ zu\ identifizierender\ Vorkommnisse}$$

$$Precision = \frac{Anzahl\ korrekt\ identifizierter\ Vorkommnisse}{Anzahl\ aller\ vorgeschlagenen\ Vorkommnisse}$$

Dabei gab es bei n Beispiel-Vorkommnissen $n-1$ verschiedene Test-Klassen, die sich durch die Anzahl der verwendeten Trainingsinstanzen unterschieden. Inspiriert durch die Kreuzvalidierung [32, Kapitel 5], die für geringe Datenmengen geeignet ist, gestaltete sich das Testverfahren so, dass bei den *Recall* Messungen die n Beispielinstanzen durchlaufen wurden und jede jeweils als Testinstanz für einen Lerner diente, der nacheinander mit 1, 2,..., $n-1$ Vertretern der restlichen Vorkommnisse trainiert worden war. Die Trainingsmenge wurde zufällig gewählt, sollte sich jedoch nicht wiederholen, d.h falls einmal eine Kombination aus k Beispielen für das Training benutzt wurde, wurde sie nicht noch einmal verwendet, es sei denn, alle $\binom{n-1}{k}$ Möglichkeiten für eine Testinstanz wurden bereits eingesetzt. Dann wurde unter diesen wiederum eine zufällig gewählt.

Danach wurde die Testinstanz klassifiziert und das Ergebnis im Gesamt-Recall der jeweiligen Test-Klasse berücksichtigt.
Bei der *Precision* wurde, um Rechenzeit zu sparen, auf die Lerner aus den

Anzahl Trainings-Instanzen	Anzahl Tests *Recall*	Anzahl Tests *Precision*
1	19	57
2	19	57
3	19	57
4	19	57
5	19	57
6	19	57
7	19	56
8	19	54
9	19	56
10	19	55
11	19	54
12	19	52
13	19	48
14	19	46
15	19	44
16	19	38
17	19	30
18	19	19

Tabelle 6.1: Anzahl der Tests

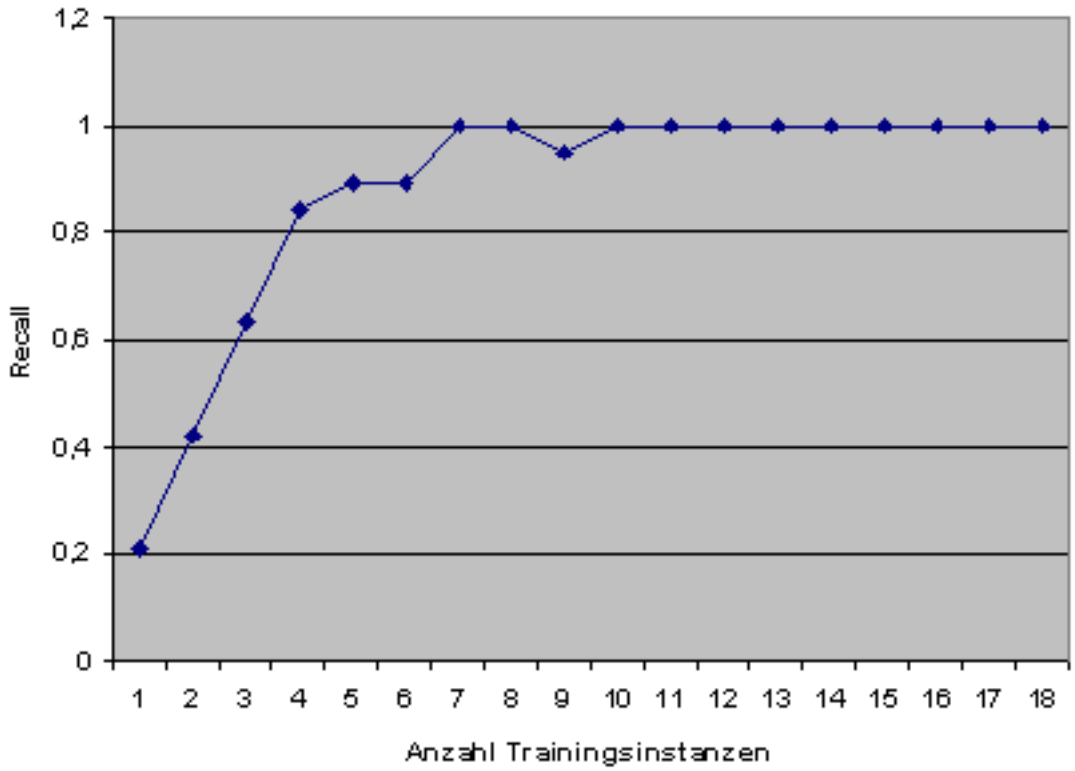


Abbildung 6.1: Recall

Recall Verfahren zurückgegriffen, es sei denn, dass der oben erwähnte Fall der Erschöpfung der Möglichkeiten auftrat. Dann wurde eine Trainingsmenge zufällig zusammengestellt, die jedoch auf jeden Fall die Testinstanz der *Recall* Bestimmung beinhalten musste, da dieser Fall nur auftreten kann, wenn diese Instanz noch nie beim Training verwendet worden ist. So wurde logischerweise sichergestellt, dass dem Lerner eine noch nicht verwendete Beispielmenge präsentiert wurde. Mit dessen Hilfe wurde nun jedes Beispieldokument wie in Kapitel 4.7.4 beschrieben ausgezeichnet, jedoch nur, wenn nicht all seine Vorkommnisse in der Trainingsmenge waren, was bei einer erhöhten Zahl von Trainingsbeispielen automatisch zu weniger Tests führte; die *Precision* für nur eine Trainingsinstanz setzte sich aus $(n-1)*d$ Versuchen (d bezeichnet die Anzahl der Dokumente), diejenige für $n-1$ Trainingsinstanzen aus natürlich $n-1$ Tests zusammen. Die Anzahl der Versuche in den einzelnen Test-Klassen kann Tabelle 6.1 entnommen werden.
Nach dem Auszeichnen wurde die Zahl der ausgezeichneten Vorkommnisse

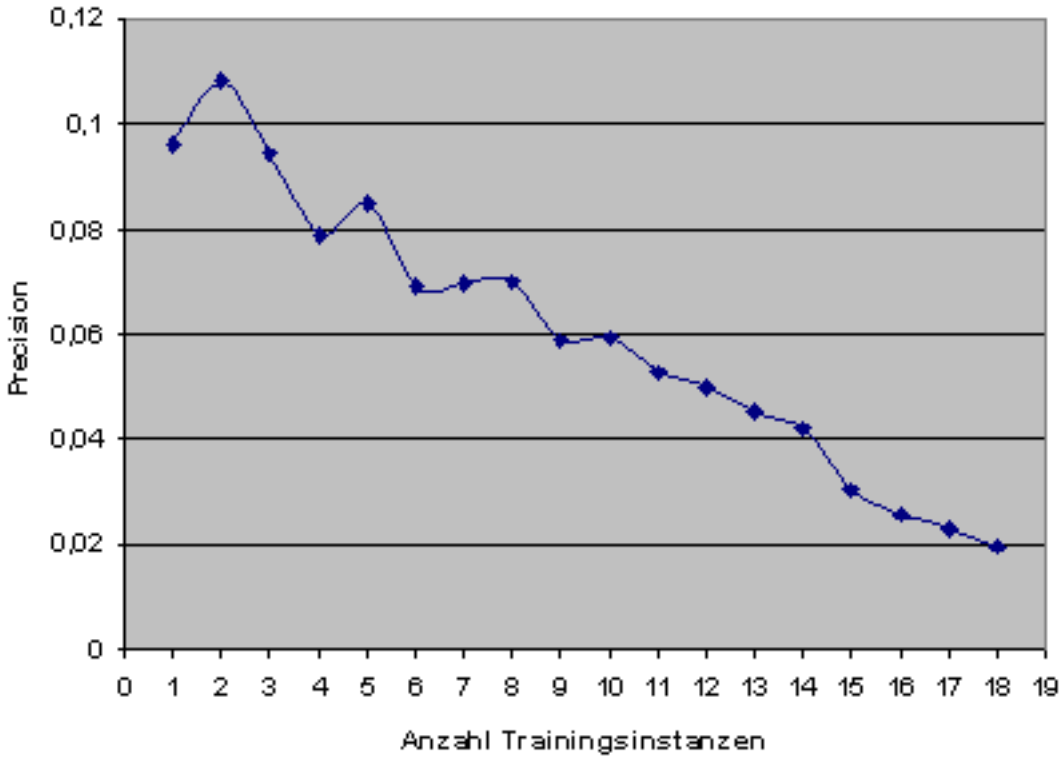


Abbildung 6.2: Precision

ermittelt, die denjenigen des Dokuments glichen, die nicht im Training verbraucht wurden. Diese wurde durch die Anzahl aller Vorschläge geteilt, wobei davon die Vorschläge abgezogen wurden, die Trainings-Instanzen abdeckten. Das Ergebnis dieser Division wurde in der Gesamt-Precision der jeweiligen Test-Klasse berücksichtigt.
Die Ergebnisse finden sich in Abbildung 6.1 und 6.2.
Die Resultate für den *Recall* sind höchst erfreulich, besagen sie doch, dass PROMETHEUS schon bei einer geringen Zahl von Beispielen die meisten, wenn nicht gar alle, wichtigen Informationen unter seinen Vorschlägen hat, immer vorausgesetzt, dass diese in Tabellen abgelegt sind, versteht sich. Dies entspricht genau der Anforderung dieses Projekts.
Getrübt wird dieser Eindruck nur ein wenig von der äußerst bescheidenen *Precision*, die jedoch besonders für viele Trainings-Instanzen zu erwarten war, da der Versionenraum durch fast jedes positive Beispiel verallgemeinert wird. Auch sind diese Werte mit Vorsicht zu genießen. Die von PROMETHEUS vorgeschlagenen Vorkommnisse sind bei genauerer Betrachtung nicht immer

schlechte oder falsche Zuordnungen; der Partner, der für die Festlegung der Instanzen eines Dokuments verantwortlich zeichnete, hat diese nur nicht ausdrücklich genutzt. Auch fällt eine magere *Precision* bei der Arbeit mit PROMETHEUS nicht unbedingt ins Gewicht, da die Durchsicht der Vorschläge tatsächlich sehr schnell vonstatten geht, und es selbst bei einer falschen Zuordnung sein kann, dass die Ansicht bei diesem Vorkommnis auf einen Bereich des Dokuments springt, der gefüllt ist mit relevanten Daten, die der Benutzer dann rasch selbst erfassen kann. Außerdem hängt die *Precision* auch stark von der Größe des Formulars ab, wenn man dieses nur auf die Felder beschränkt, in dem Daten gefunden werden sollen, verbessert diese sich natürlich, was bei diesen Versuchen jedoch nicht getan wurde. Bei jedem Dokument wurde mit dem gleichen Formular getestet, das so viele Einträge aufwies, dass alle Beispiel-Vorkommnisse darin unterkommen konnten, nämlich zwölf.
Dennoch sollte man das Fazit ziehen, dass sich der Benutzer mit der Auswahl positiver Beispiele eher einschränken sollte, was ihn natürlich auch entlastet. Stattdessen sollte der Versionenraum beispielsweise schon nach einigen Beispielen abgespeichert werden. Wenn man mehrere solcher Dateien anlegt, kann man nacheinander ihre Wirksamkeit bei neuen Dokumenten ausprobieren, womit man bei einem ähnlichen Dokumentaufbau sicherlich zügig Erfolge verzeichnen kann. Die automatische Auswahl eines Versionenraums wäre für nachfolgende Versionen von PROMETHEUS wünschenswert (siehe Kapitel 7).

Weitere Versuche machten darüber hinaus deutlich, dass der Lerner durch Zuführung von negativen Beispielen eigentlich nie verbessert wurde, sondern dass meistens der Versionenraum zusammenbrach. Auf dieses Problem wird schon in 4.7.1.4 hingewiesen. Wenn man jedoch darüber nachdenkt, ist dies gar kein sonderlich großes Problem, denn dem Benutzer wird die Last und Verantwortung genommen, den Lerner auch noch mit negativen Beispielen zu versorgen. Er lässt es einfach, das Verfahren funktioniert wie gesehen auch so!

Regex Auszeichnen

Versuche für diese Art des Auszeichnens sind schwierig auszuwerten, da das *Regex Auszeichnen* von einer Vielzahl von Faktoren abhängt. So sind die bereits eingetragenen Synonyme, die Einstellungen im 'Type'-Dialog, die Werte des Levenshtein-Abstands im entsprechenden Dialog und, wegen der *Precision*, die Anzahl der Einträge im Formular (s.o.) entscheidend.

	ohne Synonyme, etc.		***mit*** Synonyme, etc.	
	Recall	*Precision*	*Recall*	*Precision*
Dokument 1	0,2727	0,4666	0,6	0,4444
Dokument 2	0,3846	1	0,6363	0,9
Dokument 3	0	-	0,75	0,8
Dokument 4	0	0	0,75	0,875
Dokument 5	0,5454	1	0,7272	1
gesamt	0,24054	0,61665	0,6927	0,80388

Tabelle 6.2: Ergebnisse des Regex Auszeichnens

Es wurden jeweils zwei Testreihen an fünf Dokumenten durchgeführt. Bei diesen wurde das Formular vorher jeweils auf ein Mindestmaß gekürzt, d.h. das Formular eines Dokuments, das fünf Test-Vorkommnisse lieferte, hatte letztendlich nur noch fünf Einträge. Die Einstellungen für die Levenshtein-Distanz waren bei beiden Testreihen gleich, nämlich

```
Länge: 1  Abstand: 0
Länge: 5  Abstand: 1
Länge: 8  Abstand: 2
```

Bei der ersten Testreihe wurden nun die Typ-Einstellungen inklusive Maßeinheiten und Stichworte (siehe Kapitel 4.8) sowie die Synonyme an alle gesehenen Beispiele angepasst, wobei diese Einstellungen für alle Dokumente gleichermaßen galten. Bei der zweiten Testreihe wurde gänzlich auf Synonyme, Maßeinheiten und Stichworte verzichtet. Bei der Bewertung der Vorschläge wurde so verfahren, dass schon eine richtige Teil-Markierung als Treffer zählte, da der Benutzer von da aus leicht den Rest selbst übernehmen kann; die Position im Dokument ist entscheidend. Die Ergebnisse sind in Tabelle 6.2 abgebildet.

Die Versuche haben gezeigt, dass schon ein falsches bzw. für einen Geschäftsbrief ungeeignetes Synonym fatale Auswirkungen auf die *Precision* haben kann (*Dokument 1*). Aber sie haben auch die Erkenntnis gebracht, dass fortwährendes Arbeiten mit Prometheus lohnenswert ist und dieses Verfahren verbessert, da dadurch immer mehr Synonyme, Maßeinheiten und Stichworte hinzu kommen. Auf jeden Fall erfüllt auch das *Regex Auszeichnen* seinen Zweck, dass es, wenn es schon nicht alle relevanten Informationen findet, den Benutzer zumindest meistens in die Nähe dieser bringt, sobald er die Resultate begutachtet.

Kapitel 7

Zusammenfassung und Ausblick

PROMETHEUS ist die Realisierung einer interaktiven Informationsextraktion. Es ermöglicht, ein HTML-Dokument zu laden und anzuzeigen, woraufhin der Benutzer, durch Markieren von Text und einem simplen Knopfdruck Daten strukturiert in einem Formular erfassen kann. Außerdem ist es ihm freigestellt, auch das Aussehen des Formulars selbst zu bestimmen und damit festzulegen, *was* für Informationen überhaupt extrahiert werden sollen. Durch diese Flexibilität wird eine hohe Anpassung an den Anwender erreicht. Zudem ist PROMETHEUS dadurch nicht speziell auf nur eine Domäne ausgerichtet. Auch der Übergang in eine andere Sprache ist so theoretisch problemlos, dazu muss einfach nur ein neuer Eintrag bzw. lediglich ein Synonym angelegt werden.
PROMETHEUS bietet darüberhinaus eine auf Tabellen ausgerichtete Lernkomponente, die Hypothesen darüber erstellt, wie Merkmalsbezeichner zu ihren Werten stehen. Dabei unterbreitet diese Komponente dem Anwender schon anhand von wenigen Beispielen brauchbare Vorschläge, wo sich relevante Daten befinden. Die Vorschläge erreichen die bestmögliche Abdeckung, sind jedoch zuweilen ungenau, was aufgrund der Schnelligkeit, mit der ihre Durchsicht erfolgen kann, aber akzeptabel ist. Der Lerner wird während des Arbeitens mit PROMETHEUS automatisch mit den vom Benutzer selbst markierten Beispielen trainiert, wobei er aber stets die Möglichkeit hat, in den Lernprozess einzugreifen und so dessen Resultate zu lenken.
Da die Lernkomponente des Systems sich stark auf die Bezeichner des Formulars bezieht, ist, bei einer ähnlichen Anordnung von Merkmalsbezeichnern und Werten innerhalb eines Dokuments, die Flexibilität auch hierauf übertragbar.
Dasselbe gilt für das *Regex Auszeichnen*. Dieses Verfahren ermittelt mit Hilfe von regulären Ausdrücken weitere Vorschläge und "belohnt" quasi das fortwährende Arbeiten mit PROMETHEUS, da es sich durch Hinzufügen von

Synonymen und dem Pflegen seiner Einstellungen graduell verbessert.
PROMETHEUS verfügt über eine Export-Funktion nach Microsoft Excel, womit eine Weiterverarbeitung der entnommenen Daten gewährleistet ist.
Außerdem gibt es eine robuste Suchfunktion sowie die Möglichkeit, sich die angezeigten HTML-Dokumente in der Fassung anzusehen, in der sie vor einem möglichen OCR-Schritt vorlagen.
Da davon ausgegangen wird, dass dieser stattfand, wird mit Zuhilfenahme des Levenshtein-Abstandes in fast allen Bereichen versucht, den durch OCR verursachten Fehlern entgegenzuwirken. So kann der User bei der Suche durch einen Regler selbst bestimmen, wie groß dieser Abstand zwischen seinem Suchbegriff und der Fundstelle sein darf. Auch können die automatischen Auszeichnungsverfahren mit Einstellungen versehen werden, die das Levenshtein-Maß bei ihrer Ausführung berücksichtigen.
Für zukünfige Erweiterungen von PROMETHEUS sollte deshalb die Berechnung der Levenshtein-Distanz weiter verfeinert werden. Dazu könnten die Kosten der Editieroperationen (siehe Kapitel 3.7) den OCR-typischen Fehlern angepasst werden, indem man sie symbolabhängig macht. Außerdem ließen sich weitere Operationen einführen, wie beispielsweise das Verschmelzen zweier Symbole oder das Splitten eines Symbols. Arbeiten zum Thema OCR-Nachkorrektur wie [33] weisen dabei die Richtung.
Um den gesamten Arbeitsprozess zu optimieren und den Anwender weiter zu entlasten, ließe sich die gesamte Vorverarbeitung, also das Umwandeln in eine HTML-Datei mit OCR in PROMETHEUS integrieren. (Teure) Experten-Versionen wie beispielsweise von Abbeys FineReader bieten nach Auskunft eines Mitarbeiters eine Kommandozeilensteurung an, die dies ermöglichen würde.
Eine weitere Verbesserung würde das automatische Auszeichnen betreffen. Wie in Kapitel 6 vorgeschlagen, sollte man in Betracht ziehen, den Versionenraum zu gegebener Zeit abzuspeichern. Bei einem neuen Dokument kann man dann nacheinander die Wirksamkeit der verschiedenen Versionenräume, also Schemata, wie Daten abgelegt sind, ausprobieren. Auf diese Weise wird implizit festgestellt, dass ein Schema besser zu dem einen Typ Dokument, ein anderes besser zu einem anderen Typ passt. Dies ließe sich in einem ersten Schritt auch erlernen, so dass beim Öffnen eines Dokuments gleich ein passendes Schema zur Verfügung steht, das der Benutzer natürlich auch weiter verändern oder ganz ablehnen können sollte.
Zu guter Letzt wäre es mit dem Vorhandensein von viel mehr ausgezeichneten Dokumenten sicherlich zudem interessant, andere Lernalgorithmen einzusetzen. Dabei sollte das Ziel verfolgt werden, die Qualität der Vorschläge, also die *Precision*, zu verbessern. Außerdem könnte so eine Loslösung von der Fixierung auf Tabellen erreicht werden, um diesen Weg der Unterstützung des

Benutzers für jegliche Arten von Dokumenten verfügbar zu machen. Darüberhinaus könnten dann Ideen aus [14] wie das Ordnen von Vorschlägen oder die Propagierung von Korrekturen aufgegriffen und zumindest getestet werden.

Anhang A

Prometheus im Wandel der Zeit

Zu Beginn war eine derartig komfortable Benutzeroberfläche wie in der derzeitigen Version nur bedingt vorgesehen. Stattdessen sollte sich das Projekt auf eine starke computerlinguistische Komponente stützen. Unter Umständen mehrere Lerner sollten über tausende(!) Dokumente lernen, wo sich in einem neuen Dokument Zonen mit relevanten Daten befinden und wie diese erfasst werden können. Zum Schluss sollten sie lediglich ausgegeben werden.
Schnell wurde jedoch klar, dass aufgrund der Ermangelung an Trainingsinstanzen eine benutzergestützte Lernmethode angebracht ist, die auch den entscheidenden Vorteil hat, dass das Auszeichnen der Dokumente den Experten, d.h. Leuten, die in der Domäne arbeiten, überlassen bleibt.
In monatelanger Arbeit entstand ein Prototyp, nach dem exakten Ebenbild einer Vorlage des Partners, einem Excel-Sheet, siehe Abbildung A.1. Schon dieser bot eine Vielzahl von Funktionen, die jetzt in PROMETHEUS in einem ausgereifteren Stadium auftauchen. Es gab natürlich schon eine Excel-Anbindung, das Anzeigen eines Pdf-Originaldokuments und eine Vorstufe einer Methode, um über die perfekt angeordneten Buttons markierte Textstücke mit Labeln zu versehen. Das Aussehen der Benutzeroberfläche weist auch Ähnlichkeiten zu PROMETHEUS auf. Die Anzeige war zweigeteilt, die Dokumentansicht eines HTML-Files und und bei Bedarf mehrere Formulare, die aus Buttons, Textfeldern und raffinierten, erweiterbaren Listen bestanden.
Da der Partner seine eigenen Vorgaben für unbrauchbar hielt, entstand die Idee eines flexiblen Formulars, das auch durch seine Anpassungsfähigkeit an verschiedene Benutzer besticht und somit PROMETHEUS selbst flexibler macht.

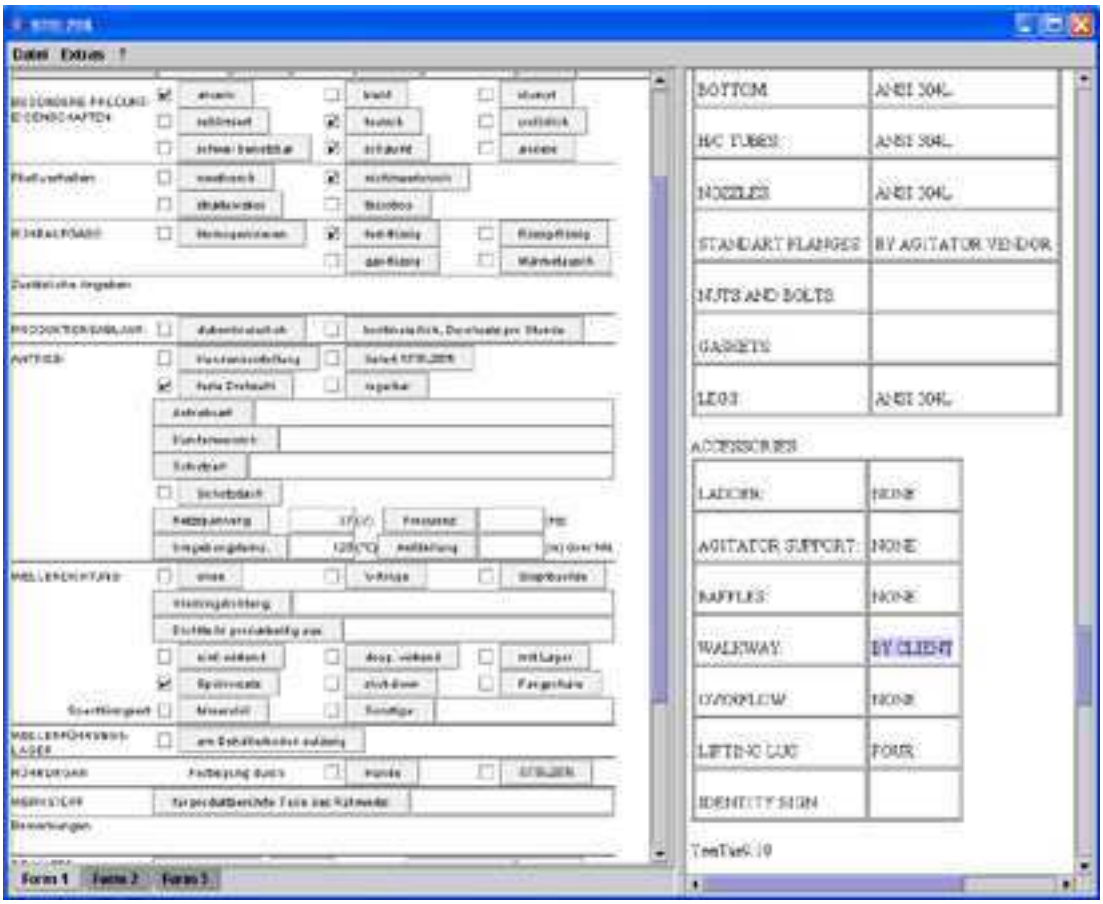


Abbildung A.1: Erster Prototyp (partnerbezogene Stellen unkenntlich gemacht)

Dieser neue Ansatz bestand zunächst wiederum aus einer getrennten Ansicht, nur dass jetzt auch mehrere Dokumente auf einmal geöffnet werden konnten, die nun auch ggf. mit mehreren Formularen assoziiert waren, siehe Abbildung A.2.
Um ein zeitgemäßes Aussehen bemüht, das dem Benutzer den Einstieg wegen der Ähnlichkeit zu anderen Anwendungen, besonders unter Windows, leichter macht, entstand die aktuelle Fenster-orientierte Oberfläche. Außerdem ist es gedanklich plausibler, einem Dokument ein einzelnes Fenster zuzuweisen. Neben den immer zahlreicher werdenden Features, gab es mit dem Umstieg auch eine Veränderung der "inneren Werte". Der Code wurde gestrafft, was zu besprochenen Mehrdeutigkeiten (siehe Kapitel 4.1), aber besonders beim Speicherverbrauch zu Fortschritten führte.

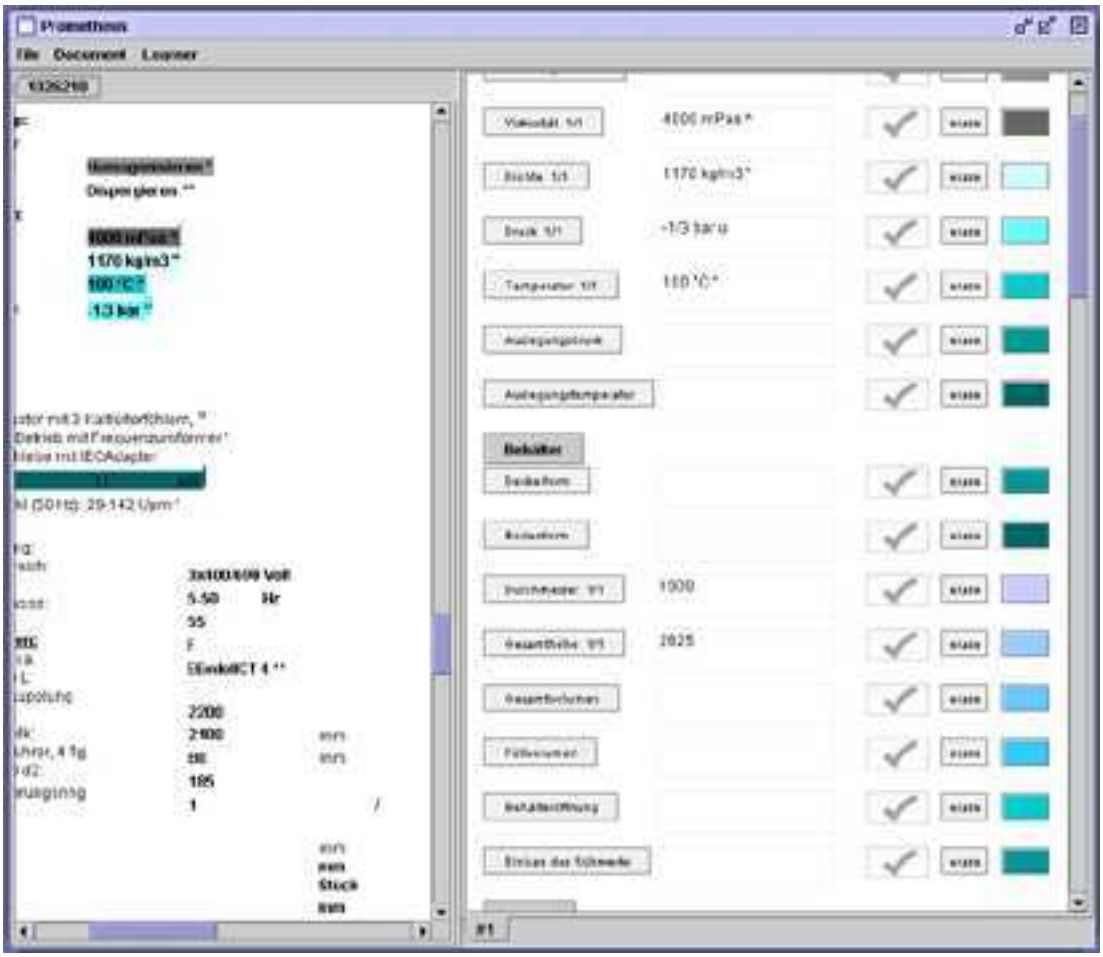


Abbildung A.2: PROMETHEUS vor Umstellung auf Fenster

Literaturverzeichnis

[1] Günter Neumann. Informationsextraktion. In *Computerlinguistik und Sprachtechnologie*, chapter 5.5. Spektrum Akademischer Verlag, 2004.

[2] *http://de.wikipedia.org/wiki/Message_Understanding_Conference*, 2005. zuletzt besucht am 1.8.2005.

[3] Overview of Information Extraction Task. http://www.cs.nyu.edu/cs/faculty/grishman/IEtask15.book_2.html#HEADING1, 1995. zuletzt besucht am 31.7.2005.

[4] Nancy Chinchor. MUC-7 Information Extraction Task Definition. *http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ie_task.html*, 1998. zuletzt besucht am 31.7.2005.

[5] Nancy Chinchor. MUC-7 Named Entity Task Definition. *http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ne_task.html*, 1997. zuletzt besucht am 31.7.2005.

[6] H.-H. Chen, Y.-W. Ding, S.-C. Tsai, and G.-W. Bian. Description of the NTU System used for MET-2. In *Proceedings of 7th Message Understanding Conference*, 1998.

[7] Zhou GuoDong. Recognizing Names in Biomedical Texts using Hidden Markov Model and SVM plus Sigmoid. In *Proceedings of 2004 Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, 2004.

[8] Behrang Mohit and Rebecca Hwa. Syntax-based Semi-Supervised Named Entity Tagging. In *Companion Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[9] Marc Rössler. Corpus-based Learning of Lexical Resources for German Named Entity Recognition. In *International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, 2004.

[10] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. NYU: Description of the MENE Named Entity System as Used in MUC-7. In *Proceedings of 7th Message Understanding Conference*, 1998.

[11] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper Induction for Information Extraction. In *Proc. Int. Joint Conf. Artificial Intelligence*, 1997.

[12] Nicholas Kushmerick. *Wrapper Induction for Information Extraction.* PhD thesis, University of Washington, 1997.

[13] Ion Muslea, Steve Minton, and Craig Knobloch. A Hierarchical Approach to Wrapper Induction. In *3rd Conference on Autonomous Agents*, 1999.

[14] Trausti Kristjannson, Aron Culotta, Paul Viola, and Andrew McCallum. Interactive Information Extraction with Constrained Conditional Random Fields. In *Nineteenth National Conference on Artificial Intelligence*, 2004.

[15] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, 2001.

[16] Aron Culotta and Andrew McCallum. Confidence estimation for information extraction. In *Proceedings of the Human Language Technologies Conference*, 2004.

[17] SELFHTML e.V. *http://de.selfhtml.org/intro/technologien/html.htm.* zuletzt besucht am 31.7.2005.

[18] *http://de.wikipedia.org/wiki/Html*, 2005. zuletzt besucht am 31.7.2005.

[19] *http://de.wikipedia.org/wiki/TIFF*, 2005. zuletzt besucht am 31.7.2005.

[20] Adobe Systems Incorporated. *http://www.adobe.de/products/acrobat/adobepdf.html*, 2005. zuletzt besucht am 31.7.2005.

[21] Matthew Robinson and Pavel Vorobiev. *Swing Second Edition.* Manning, 2003.

[22] Brian Cole, Robert Eckstein, James Elliott, Marc Loy, and David Wood. *Java Swing, 2nd Edition.* O'Reilly, 2002.

[23] Prof. Dr. Klaus U. Schulz. Nachkorrektur von Ergebnissen einer optischen Charaktererkennung, 2003. Vorlesungsskript.

[24] Benno Nieswand. *http://www.levenshtein.de.* zuletzt besucht am 31.7.2005.

[25] *http://de.wikipedia.org/wiki/Regex*, 2005. zuletzt besucht am 31.7.2005.

[26] Jeffrey E.F. Friedl. *Reguläre Ausdrücke.* O'Reilly, 1998.

[27] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document Object Model (DOM) Level 2 Core Specification. *http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113*, 2000. zuletzt besucht am 31.7.2005.

[28] Dennis Sigel and Saif Hasan. Einleitung des JAI Tutorials. *http://java.sun.com/developer/onlineTraining/javaai/jai/index.html*, 2000. zuletzt besucht am 31.7.2005.

[29] Andrew C. Oliver, Glen Stampoultzis, Avik Sengupta, and Rainer Klute. *http://jakarta.apache.org/poi/*, 2002-2005. zuletzt besucht am 31.7.2005.

[30] Thomas Mitchell. *Machine Learning.* Hanser, 2000.

[31] Della Thompson, editor. *The Concise Oxford Dictionary.* Oxford University Press, ninth edition, 1998.

[32] Ian H. Witten and Eibe Frank. *Data Mining.* Carl Hanser Verlag, 2001.

[33] Christoph Ringlstetter. OCR-Korrektur und Bestimmung von Levenshtein-Gewichten. Master's thesis, Ludwig-Maximilians-Universität München, 2003.