Technische Universität Hamburg-Harburg
Institut für Softwaresysteme

Ein System zum inhaltlichen Erschließen hierarchisch angeordneter Textfragmente

Diplomarbeit

Thomas Klingberg

Januar 2007

1. Gutachter : Prof. Dr. Ralf Möller
2. Gutachter : Prof. Dr. Joachim W. Schmidt

Inhaltsverzeichnis

1 Einleitung **5**

1.1 Motivation 5

1.2 Problemstellung und Zielsetzung 6

1.3 Gliederung der Arbeit 7

2 Modell der menschlichen textbasierten Kommunikation **9**

3 Maschinenverständliche Repräsentation von Wissen **13**

3.1 Beschreibungslogiken 14

3.2 Ableiten impliziten Wissens 17

3.2.1 Ableitung der direkten Typen eines Individuums 18

3.2.2 Ableitung von Erklärungen für erkannte Zustände 19

4 Maschinelle Extraktion von Informationen aus Text **23**

4.1 Erkennen atomarer Texteinheiten 23

4.2 Linguistische Analyse 25

4.2.1 Bestimmung der verwendeten Sprache 25

4.2.2 Bestimmen von Wortformen 26

4.3 Extraktion von Informationen 26

4.3.1 Extraktion von Konzepten 27

4.3.2 Extraktion von Relationen 28

5 System zur Klassifikation von Textfragmenten **33**

5.1 Repräsentation von Domänenwissen 33

5.2 Klassifikation hierarchisch angeordneter Textfragmente 35

5.3 HOTClassifier 40

6 Zusammenfassung und Ausblick **45**

6.1 Zusammenfassung 45

6.2 Probleme 45

6.3 Ausblick 46

A Beispiel für Ontologie und Wissensbasis **53**

B Eigenschaften des Testsystems **63**

C Inhaltstyp `Textelement` **65**

D Danksagung **69**

E Eidesstattliche Erklärung **71**

1 Einleitung

1.1 Motivation

Zwar ist die Vision des papierlosen Büros noch nicht vollständig realisiert worden, die meisten Firmen benutzen jedoch informationstechnische Systeme inzwischen nicht nur zum vereinfachten Erstellen von Dokumenten, wie Briefen oder CAD-Zeichnungen, sondern betten diese gleichzeitig in definierte Arbeitsabläufe ein, in denen festgelegt ist, welche Teilnehmer in welcher Abfolge welche Aufgaben mit dem Dokument durchzuführen haben. Dabei wird nicht nur die innerbetriebliche Kommunikation informationstechnisch abgebildet, sondern kooperierende Firmen verbinden sich und Firmen bieten ihren Kunden diesen einfachen Kommunikationsweg als Service an.

Die Suche in den immer zahlreicher werdenden Informationsquellen stellt den Benutzer vor ein Auswahlproblem [Koc04]. Im typischen Fall läuft eine klassische Suchanfrage folgendermaßen ab (nach [MF03]):

1. Die anfragende Person formt mental eine inhaltliche Anfrage.

2. Die Person kodiert die Anfrage in eine Kombination von Wörtern oder Sätzen, von denen angenommen wird, dass sie Dokumente charakterisieren, die die benötigte Information enthalten.

3. Das Computersystem bestimmt eine Menge an nach Relevanz geordneten Dokumenten, die der textuellen Anfrage entsprechen.

4. Die Person betrachtet einige der als am relevantesten eingeordneten Dokumente, liest diese und ergründet ihre Bedeutung.

5. Falls die inhaltliche Anfrage beantwortet werden konnte, ist der Prozess erfolgreich abgeschlossen. Ansonsten wird das gerade erworbene Wissen verwendet, um die textuelle Anfrage umzuformulieren und den Prozess erneut zu durchlaufen.

Das Interesse des Benutzers ist es, möglichst genaue und vollständige Informationen präsentiert zu bekommen [BYRN99]:

- **Vollständigkeit (Recall)**: Ist $D = d_1, \ldots, d_m$ eine Menge von Dokumenten, $q \in Q$ eine Anfrage aus der Menge aller möglichen Anfragen Q und D_a die Menge der in D zur Anfrage q gefundenen Dokumente. Ist ferner R_q die Menge der zur Anfrage q relevanten Dokumente, so berechnet sich die Vollständigkeit zu

$$R(q, D) = \frac{|D_a \cap R_q|}{|R_q|}$$

  Anschaulich betrachtet ist die Vollständigkeit der Quotient aus der Anzahl der gefundenen relevanten Dokumente und der Anzahl aller relevanten Dokumente im Informationsraum.

- **Genauigkeit (Precision)**: Verhältnis der gefundenen relevanten Dokumente zu der Anzahl aller gefundenen Dokumente

$$P(q, D) = \frac{|D_a \cap R_q|}{|D_a|}$$

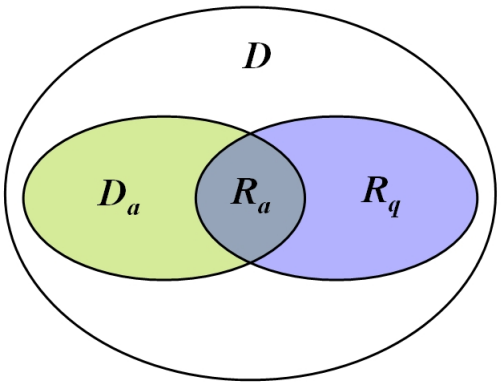

Abb. 1.1.1: Menge der relevanten zurückgelieferten Dokumente R_a

1.2 Problemstellung und Zielsetzung

Es gibt mehrere Ansätze, um Precision des Retrieval gegenüber der Volltextsuche zu Erhöhen. Eine davon ist die Klassifikation, d.h. die Zuordnung von Dokumenten zu Themengebieten. Dies erlaubt die Eingrenzung, dass nach einem Begriff nur in einem bestimmten Themengebiet gesucht wird und verringert die Rate der für den Benutzer irrelevanten zurückgelieferten Dokumente. Manuelle Ansätze der Kategorisierung gibt es z.B. bei Yahoo.

In dieser Diplomarbeit wurde untersucht, inwiefern die Klassifikation von Texten automatisierbar ist. Die betrachteten Texte weisen dabei eine Besonderheit auf: Sie sind nicht unabhängig voneinander, sondern hierarchisch angeordnet. Es wird angenommen, dass ein Kindelement jeweils einen Teilaspekt der Thematik behandelt, die Inhalt seines Elternelements ist. Der Inhalt eines Textes steht ist also im Gesamtkontext der Inhalte der ihm übergeordneten Texte zu sehen. Deswegen wird nachfolgend von *Textfragmenten* gesprochen bzw. aufgrund ihrer hierarchischen Anordnung von *Hierarchically Ordered Textfragments* (HOTs). Solche hierarchisch angeordneten Textfragmente finden sich beispielsweise in Gesetzestexten, die sich in Kapitel, Paragraphen und Absätze unterteilen. HOTs treten ebenfalls in Kommunikationssträngen auf, wie sie beispielsweise in Newsgroups zu finden sind: In einer ersten Nachricht wird von einer Person ein Thema erörtert und andere Personen antworten auf diese Nachricht. Die nachfolgenden Nachrichten sind dann im Kontext der ersten zu sehen.

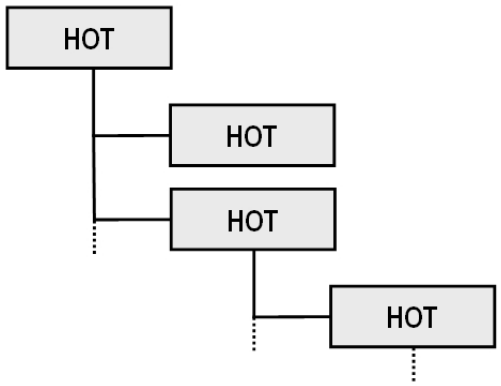

Abb. 1.2.1: Hierarchisch angeordnete und inhaltlich abhängige Textfragmente

1.3 Gliederung der Arbeit

Kapitel 2 erläutert, wie sich die menschliche textuelle Kommunikation modellieren läßt. Kapitel 3 zeigt, wie konzeptuelles Wissen maschinenverständlich repräsentiert werden kann, in Kapitel 4 wird erörtert in welchem Umfang sich die im Text enthaltenen Informationen extrahieren und wie diese mit Hilfe des vorhandenen konzeptuellen Wissens ausgewertet und zur Klassifikation des Textfragments verwendet werden können. Kapitel 5 stellt das entwickelte System zum inhaltlichen Erschließen der hierarchisch angeordneten Textfragmente dar. In Kapitel 6 werden die Ergebnisse dieser Arbeit zusammengefasst und es wird ein Ausblick auf zukünftige Anwendungsgebiete gegeben.

2 Modell der menschlichen textbasierten Kommunikation

Um nachfolgend die Möglichkeiten einer maschinellen semantischen Interpretation der zu klassifizierenden Textfragmente erörtern zu können, wird in diesem Kapitel zunächst betrachtet, wie sich die menschliche textbasierte Kommunikation modellieren läßt.

Wie Menschen Informationen mit der Hilfe von Texten kommunizieren wurde bereits eingehend in den Forschungsfeldern der Philosophie und Linguistik untersucht. Ein in der Computerlinguistik häufig verwendetes Modell basiert auf dem ursprünglich von Odgen und Richards beschriebenen *Bedeutungsdreieck* (engl.: meaning triangle) [MSS+03] (siehe Abb. 2.0.1). Das Bedeutungsdreieck modelliert das bestehende Zusammenspiel zwischen *Symbolen*, *Konzepten* und *Sachen*. Eine *Sache* ist ein in der Welt existierendes Objekt, dass sowohl stofflich (wie beispielsweise ein Schiff) als auch nicht-stofflich (wie beispielsweise die Kälte) sein kann. Von diesem hat der Mensch ein Verständnis - ein *Konzept*, dass sich auf die *Sache* bezieht und dessen aus direkten oder indirekten Erfahrungen gewonnene charakteristische Eigenschaften abbildet. Um sich mitzuteilen, abstrahiert der Mensch *Konzepte* durch *Symbole* wie Worte oder optische Zeichen, die aufgrund der Tatsache, dass sie *Konzepte* benennen, auch als deren *Deskriptoren* bezeichnet werden. Aufgrund der zwischen *Konzept* und *Sache* gegebenen Beziehung steht ein *Deskriptor* damit indirekt auch für eine *Sache*.

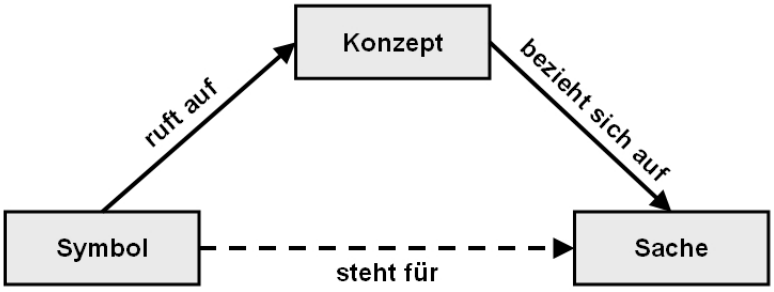

Abb. 2.0.1: Bedeutungsdreieck (nach [MSS+03])

Ein Konzept kann dabei durch mehrere unterschiedliche Deskriptoren benannt werden (*Synonymie*), insbesondere durch Deskriptoren unterschiedlicher Sprachen (deutsch: „Schiff“, englisch: „ship“). Zudem stehen die Konzepte in Beziehung zueinander: Sie sind hierarchisch geordnet („Ein Schiff ist ein Wasserfahrzeug.“) und werden verwendet, um andere Konzepte zu spezifizieren („Ein Segelschiff ist ein Schiff, das ein Segel besitzt.“).

Zusammen mit einer komplexen Grammatik erlaubt dies dem Menschen, eine Aussage auf unterschiedliche Weise textuell auszudrücken. Er kann als *Generator* eines Textes

- eine referenzierte Sache direkt mit einer ihrer synonymen Deskriptoren benennen: „Das ist ein Motorschiff.“
- eine referenzierte Sache präzise beschreiben, so dass diese eindeutig bestimmt ist: „Das Schiff wird von einem Motor angetrieben.“
- eine referenzierte Sache unpräzise beschreiben, so dass nicht eindeutig geklärt ist, um welches Konzept s sich handelt:
  „Das Schiff hat einen Motor.“
  (Ein Segelschiff kann ebenfalls einen Motor haben - einen Hilfsantrieb.)

Der *Generator* drückt also einen Sachverhalt in menschlicher Sprache aus und speichert diese in textueller Form. Er erzeugt bzw. generiert in diesem Sinne einen Text. Der *Rezeptor* (Empfänger) nimmt die in textueller Form enkodierten Informationen auf, extrahiert und interpretiert dieses mit Hilfe seines Wissens. Da das Wissen eines Menschen durch seine persönlichen direkten oder indirekten Erfahrungen geprägt ist, haben Generator und Rezeptor nicht unbedingt die gleiche Vorstellung von einem Konzept.

Abb. 2.0.2: Interpretationsebenen (nach [Fuh06])

Eine Maschine in die Lage zu versetzen, Texte wissensbasiert inhaltlich zu klassifizieren erfordert, dass Wissen maschinenverständlich erfasst wird und Verfahren zur Verfügung stehen, die es erlauben, im Text enthaltenen Aussagen maschinell zu extrahieren. Folgende Sachverhalte sind dabei hinderlich:

- *Synonymie*: Unterschiedliche Begriffe können dasselbe Konzept referenzieren. Dem System müssen also alle Begriffe bzw. Deskriptoren bekannt sein, mit denen ein Konzept bezeichnet werden kann. Sind diese nicht bekannt, so kann von dem Wort keine Verbindung zu dem Konzept gezogen werden, obwohl dieses hinterlegt ist.

Konsequenz: Der Text wird nicht mit dem Konzept assoziiert, obwohl er dieses beinhaltet.

- *Polysemie*: Auf der anderen Seite kann ein und dasselbe Wort ganz unterschiedliche Bedeutungen aufweisen, die erst aus dem Zusammenhang des Textes erkenntlich werden.
- *Umschreibungen*: Wie oben erwähnt wird nicht immer der exakte Begriff verwendet, sondern das Konzept wird umschrieben, indem z.B. wichtige und charakteristische Eigenschaften von diesem genannt werden. Hier muss die Transferleistung erbracht werden, von den Eigenschaften auf das Konzept zu schließen.
- *Negation*: Sprachlich kann ausgedrückt werden, dass eine Sache nicht enthalten bzw. betrachtet wird („nicht“, „verzichten“). Die Frage ist, wie damit umgegangen werden soll: Ein Argument für die Nichtbeachtung von Negationen ist, dass die negierte Erwähnung eines Konzepts auch eine Aussage darstellt und so eine Klassifikation des Textes in dieses negiert erwähnte Konzepts angebracht ist.

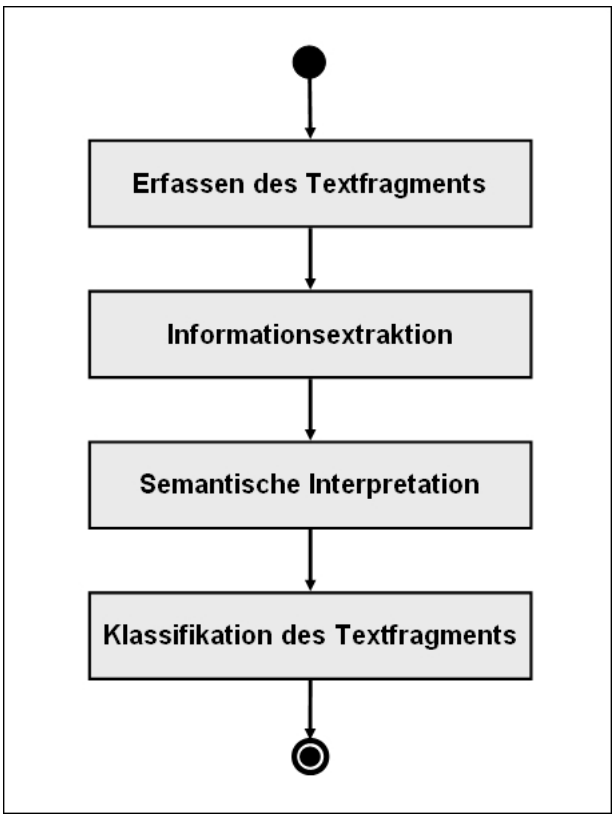

Abb. 2.0.3: Arbeitsschritte zur Klassifikation eines Textfragments

3 Maschinenverständliche Repräsentation von Wissen

Um eine maschinelle semantische Interpretation und Klassifikation der Textfragmente zu ermöglichen, ist es wie im vorangegangenen Kapitel 2 beschrieben notwendig, vorhandenes konzeptuelles Wissen maschinenverständlich zu hinterlegen.

Dies geschieht mit der Erstellung einer sogenannten *Ontologie*. Dieser Begriff stammt ursprünglich aus der Philosophie und bezeichnet dort das Studium des Seins durch die Identifikation und Beschreibung real existierender Dinge, ihrer Eigenschaften und ihrer Beziehungen untereinander. In der Informatik versteht man unter einer Ontologie eine „explizite und formale Spezifikation der gemeinsamen Konzeptualisierung" eines Wissensgebietes (der *Domäne*) [AH04]. In ihr strukturiert eine Gruppe von Personen (meist vertreten durch einige Experten) die in der Domäne auftretenden Objekte, Eigenschaften und Beziehungen in Konzepte und schafft damit ein gemeinsames abstraktes Modell und Verständnis für diesen Weltausschnitt. Eine Abstraktion erfolgt insofern, als dass das Wissensgebiet normalerweise nicht vollständig mit allen möglichen Details und Ausprägungen erfasst wird, sondern in einem für die angestrebte Verwendung relevanten abgegrenzten Umfang.

Zur Formulierung einer Ontologie stehen mittlerweile mehrere Beschreibungssprachen zur Verfügung, von denen die *Web Ontology Language* (OWL) vom *World Wide Web Consortium* (W3C) zum Standard[1] erhoben wurde [MGH04]. OWL baut auf dem *Resource Description Framework* (RDF) und *RDF Schema* (RDFS) auf und verwendet eine auf der *eXtensible Markup Language* (XML) basierende Syntax (siehe Abb. 3.0.1 und Anhang A). OWL besteht zudem aus drei aufeinander aufbauenden Komplexitätsstufen, die sich in ihrer jeweiligen Ausdrucksstärke unterscheiden: *OWL Full*, *OWL DL* und *OWL Lite*. Von diesen ist inbesondere *OWL DL* von Interesse, das zwar nicht den vollen Sprachumfang von *OWL Full* besitzt[2], durch diese Einschränkungen jedoch algorithmische Entscheidbarkeit auf der Basis von *Beschreibungslogiken* gewährleistet und so logische Schlußfolgerungen mit Hilfe einer *Inferenzmaschine* gestattet (siehe Kapitel 3.1).

1 Das W3C definiert *Recommendations* („Empfehlungen"), deren Status den von anderen Organisation erlassenen Standards entspricht (siehe `http://www.w3.org/2003/06/Process-20030618/tr.html`).

2 Eine Einschränkung von *OWL DL* gegenüber *OWL Full* ist beispielsweise, dass hier keine *Reifikation* (das Treffen von Aussagen über Aussagen) erlaubt ist [Kun05].

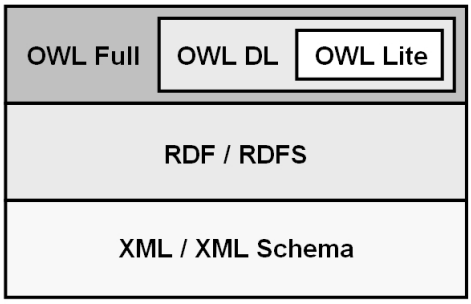

Abb. 3.0.1: Sprachbasis der *Web Ontology Language* (OWL) (nach [AH04])

Der Prozess der Ontologie-Erstellung kann dabei durch softwaretechnische Entwicklungsumgebungen wie Protégé [HKR+04] oder KAON [GSV04] unterstützt werden, die durch Visualisierung der Inhalte und geeignete Eingabemasken ein von der Beschreibungssprache abstrahiertes Arbeiten ermöglichen und zudem teilweise über Zusatzmodule verfügen, die ein halb-automatisches Erstellen von Ontologien aus bestehenden textbasierten Datenbeständen ermöglichen (z.B. OntoLT [BOS03] bei Protégé, TextToOnto [Hal04b] bei KAON).

Eine gute Einführung in das methodische Vorgehen bei der manuellen Erstellung von Ontologien gibt [NM01]. Noy und McGuiness beschreiben dort einen iterativen Ansatz, bei dem ein grober erster Entwurf der Ontologie in mehreren Durchläufen evolutionär überarbeitet und verfeinert wird.

3.1 Beschreibungslogiken

Beschreibungslogiken (engl.: Description Logics, DL) sind Teilmengen der Prädikatenlogik erster Stufe [Hal04a] und bilden eine Familie von formalen Sprachen zur Repräsentation und Verarbeitung der Semantik natürlichsprachlicher Ausdrücke. Aus diesem Grund werden sie auch als *terminologische Logiken* bezeichnet [Kun05].

Beschreibungslogiken unterscheiden zwischen zwei Komponenten: der *terminologischen Komponente* (der sogenannten *T-Box*) und der *assertionalen Komponente*[3] (der *A-Box*).

Terminologische Komponente

Die T-Box beinhaltet die Konzeptualisierung des betrachteten Weltausschnitts und stellt somit eine Ontologie von diesem dar. Die grundlegendste per Beschreibungslogik in einer T-Box definierbare Einheit ist ein *atomares Konzept*. Dieses steht für eine Menge von in der Domäne auftretenden gleichartigen Objekten[4] bzw. *Individuen*. Beispielsweise lässt

3 Der Fachbegriff „assertionale Komponente“ leitet sich von der englischen Bezeichnung „assertional component“ ab („to assert sth.“ = „etw. feststellen“/„etw. erklären“).

4 In Analogie zu der Begriffswelt des *OOAD* (Objektorientierte Analyse und Design) werden die Konzepte der Beschreibungslogiken in der Literatur auch häufig als *Klassen* bezeichnet, Individuen als *Instanzen*.

sich ein Konzept mit dem Namen Ship einführen, dass die Menge aller real existierenden Schiffe (die Individuen „MS Deutschland“, „Olle Deern“, ...) repräsentieren soll. Mit Hilfe des *Äquivalenz-Operators* ($\equiv$), des *Subsumtions-Operators*[5] ($\sqsubseteq$), sowie durch *Konjunktion* ($\sqcap$), *Disjunktion* ($\sqcap$) und *Negation* ($\neg$), lassen sich einfache Beziehungen zwischen den Konzepten darstellen. So kann durch die folgenden Aussagen festlegt werden, dass ein Schiff ein Fahrzeug ist und dass Segelschiffe und Motorschiffe beides Schiffe sind:

$$\begin{aligned} \mathsf{Ship} &\sqsubseteq \mathsf{Vehicle} \\ \mathsf{SailingShip} &\sqsubseteq \mathsf{Ship} \\ \mathsf{MotorShip} &\sqsubseteq \mathsf{Ship} \end{aligned}$$

Fügt man nun noch die Aussage

$$\mathsf{Ship} \sqsubseteq \mathsf{SailingShip} \sqcup \mathsf{Motorship}$$

hinzu, so ist das Konzept Ship *abgeschlossen* (engl.: complete) und ihm können keine Individuen angehören, die nicht ein Segelschiff oder ein Motorschiff sind.

Komplexere Eigenschaften von Konzepten, als die eben gezeigte einfache Spezialisierung bzw. Generalisierung, lassen sich mit Hilfe von *Rollenrestriktionen* abbilden. Eine *Rolle* ist eine binäre Beziehung zwischen zwei Individuen, für die ein *Einsatzbereich* (engl.: *domain*; die Konzepte, auf deren Individuen die Rolle angewandt werden kann) und eine *Reichweite* (engl.: *range*; die Konzepte, deren Individuen diese Rolle einnehmen können) definiert ist. Mit der folgenden Aussage läßt sich durch die Rolle hasPart und Verwendung des Existenzquantors ($\exists$) zum Beispiel festlegen, dass ein Segelschiff ein Schiff ist, das ein Segel haben muss (SailingShip muss hierfür im Einsatzbereich der Rolle hasPart liegen, Sail in deren Reichweite):

$$\mathsf{SailingShip} \equiv \mathsf{Ship} \sqcap \exists \mathsf{hasPart}.\mathsf{Sail}$$

Weitere in Beschreibungslogiken gängige Konstrukte und die aus der Unterstützung dieser Konstrukte abgeleiteten Bezeichnungen der DL-Sprachen, sind in Tab. 3.1.1 dargestellt. Die einzelnen Bezeichner werden je nach Ausdrucksmächtigkeit einer Beschreibungslogik zusammengesetzt. Die Ausdrucksmächtigkeit von *OWL DL* entspricht der Beschreibungslogik $\mathcal{SHOIQ}$.

5 Die *Subsumtion* $A \sqsubseteq B$ zweier Konzepte A und B sagt aus, dass A eine Spezialisierung und damit eine Teilmenge von B ist.

Konstrukt	Syntax	Sprache			
Atomares Konzept	A	$\mathcal{FL}_0$	$\mathcal{FL}^-$	$\mathcal{AL}$	$\mathcal{S}$
Atomare Rolle	R				
Konzeptkonjunktion	$C \sqcap D$				
Qualifizierter Allquantor	$\forall R.C$				
Existenzquantor	$\exists R$				
Allgemeinstes Konzept	$\top$				
Speziellstes Konzept	$\bot$				
Atomare Negation	$\neg A$				
Negation	$\neg C$	$\mathcal{C}$			
Konzeptdisjunktion	$C \sqcup D$	$\mathcal{U}$			
Qualifizierter Existenzquantor	$\exists R.C$	$\mathcal{E}$			
Zahlenrestriktion	$(\geq nR)$, $(\leq nR)$	$\mathcal{N}$			
Nominal	$\{a_1 \ldots a_n\}$	$\mathcal{O}$			
Rollenhierarchie	$R \subseteq S$	$\mathcal{H}$			
Inverse Rolle	R^-	$\mathcal{I}$			
Qualifizierte Zahlenrestriktion	$(\geq nR.C)$, $(\leq nR.C)$	$\mathcal{Q}$			

Tab. 3.1.1: Gängige Konstrukte von Beschreibungslogiken (nach [GPFLC04, HST00])

OWL ermöglicht es darüber hinaus, dass Konzepte mit bestimmten Arten von *Vermerken* (engl.: annotations) versehen werden können. Mit Vermerken des Typs `rdfs:label`, den *Label Properties*, lassen sich einem Konzept beispielsweise sprachabhängige Deskriptoren anfügen:

```
<owl:Class rdf:ID="Ship">
    <rdfs:label xml:lang="en">ship</rdfs:label>
    <rdfs:label xml:lang="de">Schiff</rdfs:label>
</owl:Class>
```

Assertionale Komponente

A-Boxen[6] enthalten Aussagen, die über spezielle Individuen vorgenommen werden und entsprechen damit nach Definition von Maedche, Staab, u.a. [MSS+03] *Wissensbasen* (siehe Tab. 3.1.2). In ihnen werden Individuen Konzepten zugeordnet und bestehende Relationen zwischen Individuen erfasst.

6 Einer T-Box können mehrere unterschiedliche A-Boxen zugeordnet werden.

	Ontologie	Wissensbasis
Aussagen	generell, intensional	speziell, extensional
Inhalt	nahezu statisch	kontinuierlich erweitert
Beschreibungslogik	T-Box	A-Box

Tab. 3.1.2: Unterschiede zwischen Ontologie und Wissensbasis (nach [MSS+03])

Im Kontext dieser Arbeit ergeben sich die einer A-Box hinzugefügten Aussagen aus dem gemeinsamen Auftreten von Konzept-Deskriptoren in dem analysierten HOT: Wird der Deskriptor eines Konzepts im Text erkannt, so wird ein Individuum des entsprechenden Konzepts angelegt und mit dem Textfragment assoziiert. Wird zudem eine Beziehung zwischen zwei Konzept-Deskriptoren detektiert, so werden die entsprechenden Individuen mit der die erkannte Beziehung repräsentierenden Rolle verknüpft. Die inhaltliche Klassifikation der Textfragmente ergibt sich somit aus der Konzeptzugehörigkeit der für sie angelegten Individuen[7].

Der Text „*Das Schiff hat ein Segel.*“ ließe sich beispielsweise in die folgenden A-Box Assertionen umsetzen[8]:

$$\begin{aligned} \mathsf{ship}_1 &: \mathsf{Ship} \\ \mathsf{sail}_1 &: \mathsf{Sail} \\ \mathsf{hasPart}(\mathsf{ship}_1, \mathsf{sail}_1) & \end{aligned}$$

Damit wäre der Inhalt des Textfragments also polymorph mit den Konzepten Ship und Sail klassifiziert.

3.2 Ableiten impliziten Wissens

Mit Beschreibungslogiken lässt sich jedoch nicht nur explizites Wissen ausdrücken. Der besondere Vorteil ihres Einsatzes bei der Wissensrepräsentation besteht darin, dass sich auf bestimmten Untermengen von ihnen entscheidbare Algorithmen definieren lassen, die implizit enthaltenes Wissen ableiten und zugänglich machen können [ACD+06, HST00, SS04, DFH03]. Diese Algorithmen werden von *Inferenzmaschinen*, den sogenannten *Reasonern*[9], implementiert und als Dienste bereitgestellt. Bekannte Reasoner über Beschreibungslogiken sind beispielsweise *RacerPro* [HMW05], *Fact++* [TH06] und *Pellet* [PS04]. Die beiden grundlegenden Inferenzdienste sind die *Konsistenzprüfung*, bei der die Widerspruchsfreiheit der getroffenen Definitionen und Aussagen kontrolliert wird, sowie die *Subsumtionsprüfung*, bei der getestet wird, ob alle Instanzen eines Konzepts A

7 Das in dieser Arbeit entwickelte Verfahren zur Klassifikation von Textfragmenten wird ausführlich in Kapitel 5.2 beschrieben.

8 Wie in Texten enthaltene Aussagen extrahiert werden können, wird im nachfolgenden Kapitel 4 erörtert.

9 Die Bezeichnung *Reasoner* leitet sich von dem englischen Verb „to reason“ („schlussfolgern“) ab.

auch Instanzen eines Konzepts B sind und somit $A \sqsubseteq B$ gilt. Wird die Subsumtionsprüfung für alle Konzepte einer T-Box durchgeführt und so die Hierarchie der Konzepte ermittelt, spricht man von einer *Klassifikation* der Ontologie.

Entsprechend der von ihnen unterstützten Beschreibungslogik bieten die Reasoner darüber hinaus noch weitere, teils unterschiedliche Inferenzdienste an [ACD+06, Hal04a]. Im folgenden werden zwei Inferenzmechanismen vorgestellt, die im Kontext dieser Arbeit von besonderer Bedeutung sind: die Ableitung der *direkten Typen* eines Individuums und die Ableitung von Erklärungen für erkannte Zustände.

3.2.1 Ableitung der direkten Typen eines Individuums

Als *direkte Typen*[10] (engl.: *direct types*) eines Individuums werden die Namen der *speziellsten Konzepte* (engl.: *most-specific concepts*) bezeichnet, deren definierte Eigenschaften das Individuum erfüllt. Die Ableitung der speziellsten Konzepte erfolgt durch *deduktives Schließen*[11] basierend auf einer zuvor durchgeführten Klassifikation der T-Box. So läßt sich aus den nachfolgend gesetzten T-Box Definitionen und A-Box Assertionen schließen, dass es sich bei dem Individuum ship_1 um ein Segelschiff handeln muss, ohne dass dieses Wissen explizit durch eine Assertion vorgegeben wurde:

$$\mathsf{SailingShip} \equiv \mathsf{Ship} \sqcap \exists \mathsf{hasPart}.\mathsf{Sail}$$

$$\begin{aligned} \mathsf{ship}_1 &: \mathsf{Ship} \\ \mathsf{sail}_1 &: \mathsf{Sail} \\ \mathsf{hasPart}(\mathsf{ship}_1, \mathsf{sail}_1) & \end{aligned}$$

$$\Longrightarrow \mathsf{ship}_1 : \mathsf{SailingShip}$$

Wurde das Textfragment, auf dessen Analyse die A-Box Assertionen basieren, mit dem Individuum ship_1 assoziiert, so ist dessen Inhalt nun mit dem spezielleren Konzept SailingShip klassifiziert.

Dabei ist zu beachten, dass die abgeleiteten *speziellsten Konzepte* die im Moment des Schließens aufgrund des aktuell verfügbaren Wissens ableitbaren speziellsten Konzepte darstellen. Sollten später weitere Information über das Individuum bekannt und als Assertionen der A-Box hinzugefügt werden, so lässt sich eventuell die Zugehörigkeit zu noch spezielleren Konzepten ableiten.

10 Zur Ableitung der direkten Typen eines Individuums stellt RacerPro den Befehl `individual-direct-types` zur Verfügung.

11 Bei einer *Deduktion* wird vom Allgemeinen auf das Besondere geschlossen. Das Gegenteil bildet die in mathematischen Beweisen häufig verwendete *Induktion*, bei der vom Besonderen auf das Allgemeine geschlossen wird.

Um ein solches deduktives Ableiten von Konzeptzugehörigkeiten für Individuen zu ermöglichen, ist es erforderlich, dass die in der T-Box definierten Konzepte nicht nur notwendige Bedingungen (in Form von Zuweisungen mit dem Subsumtions-Operator $\sqsubseteq$), sondern zwingend notwendige und hinreichende Bedingungen (gekennzeichnet durch den Äquivalenz-Operator $\equiv$) aufweisen müssen. Nur durch das Erfüllen einer notwendigen und hinreichenden Bedingung ist eine deduktive Folgerung möglich.

Bei der Definition von Konzepten ist desweiteren zu beachten, dass die meisten Inferenzmaschinen eine *Open World Assumption* (OWA) treffen. Diese besagt, dass etwas, das nicht als *wahr* bewiesen werden kann, nicht automatisch als *falsch* angenommen wird. Vorsicht ist daher bei der Verwendung von negierten Ausdrücken in der Konzeptdefinition geboten: Die Zugehörigkeit eines Individuums zu diesem Konzept kann in diesem Fall nur dann deduktiv geschlossen werden, wenn die negierte Aussage über das Individuum auch explizit als Assertion in der betrachteten A-Box enthalten ist. Ist beispielsweise ein Wasserfahrzeug als ein Fahrzeug definiert, das weder ein Landfahrzeug, noch ein Luftfahrzeug ist

$$\mathsf{WaterVehicle} \quad \equiv \quad \mathsf{Vehicle} \sqcap \neg\,(\mathsf{LandVehicle} \sqcup \mathsf{AirVehicle})$$

so kann für ein Individuum $\mathsf{vehicle}_1 : \mathsf{Vehicle}$ nur dann geschlossen werden, dass es ein WaterVehicle ist, wenn explizit die Assertionen $\neg(\mathsf{vehicle}_1 : \mathsf{LandVehicle})$ und $\neg(\mathsf{vehicle}_1 : \mathsf{AirVehicle})$ zur A-Box hinzugefügt wurden. Noch problematischer ist die Verwendung negierter Rollenrestriktionen: Wird zum Beispiel ein Motorschiff als ein Schiff definiert, dass kein Segel besitzt

$$\mathsf{MotorShip} \quad \equiv \quad \mathsf{Ship} \sqcap \neg\,(\exists \mathsf{hasPart}.\mathsf{Sail})$$

so kann für ein Individuum $\mathsf{ship}_2 : \mathsf{Ship}$ selbst in dem Fall, dass eine Assertion $\neg\mathsf{hasPart}(\mathsf{ship}_2, \mathsf{sail}_2)$ (mit $\mathsf{sail}_2 : \mathsf{Sail}$) in der A-Box enthalten ist, nicht deduktiv abgeleitet werden, dass ship_2 ein MotorShip ist, da bei gegebener OWA nicht ausgeschlossen ist, dass zu einem späteren Zeitpunkt eine hasPart-Relation zu einem anderen Individuum des Konzepts Sail hinzugefügt wird.

3.2.2 Ableitung von Erklärungen für erkannte Zustände

Mit Hilfe von deduktivem Schliessen auf Basis von Aussagen in Beschreibungslogik ist es möglich, implizite Konzeptzugehörigkeiten von bereits bekannten Individuen abzuleiten. Mit den bislang vorgestellten Techniken lässt sich jedoch nicht erreichen, dass ein bestimmter Satz von Assertionen die Existenz von weiteren, bislang nicht erfassten Individuen und Relationen impliziert. Wird beispielsweise beobachtet, dass aus einem Rohr Öl austritt, so lässt sich dies mit Hilfe der Konzepte Pipe und Oil, sowie einer

Relation drops ausdrücken:

$$\begin{aligned} \mathsf{pipe}_1 &: \mathsf{Pipe} \\ \mathsf{oil}_1 &: \mathsf{Oil} \\ \mathsf{drops}(\mathsf{pipe}_1, \mathsf{oil}_1) & \end{aligned}$$

Der Schluss, dass in einem solchen Fall eine Leckage (Konzept Leakage) am Rohr vorliegt, lässt sich per Deduktion allerdings nicht erreichen.

Erklärungen für erkannte Zustände lassen sich jedoch durch *abduktives Schliessen* [ACD+06, EKS06] gewinnen. Im Abduktionsprozess wird versucht, für einen Satz von Assertionen Γ eine Menge zusätzlicher Assertionen Δ abzuleiten, die Γ vor dem Hintergrund einer T-Box $\mathcal{T}$ und einer A-Box $\mathcal{A}$ erklären:

$$\mathcal{T} \cup \mathcal{A} \cup \Delta \models \Gamma$$

Als Basis für den Abduktionsprozess dienen in der Ontologie definierte *DL-sichere Regeln*[12]. Eine solche Regel kann für das oben beschriebene Beispiel folgendermaßen lauten:

$$\begin{aligned} \mathsf{drops}(\mathsf{X},\mathsf{Y}) \leftarrow\ & \mathsf{Pipe}(\mathsf{X}),\ \mathsf{Oil}(\mathsf{Y}),\ \mathsf{Leakage}(\mathsf{Z}), \\ & \mathsf{hasFailure}(\mathsf{X},\mathsf{Z}),\ \mathsf{causesEmission}(\mathsf{Z},\mathsf{Y}) \end{aligned}$$

Die Regel sagt für sich genommen aus, dass im dem Fall des Vorhandenseins der auf der rechten Seite angegebenen Assertionen (*Prämisse*) die auf der linken Seite angegebene Assertion (*Konklusion*) zur Wissensbasis hinzugefügt werden soll. Im Abduktionsprozess wird die Regel jedoch in der Weise ausgewertet, dass bei gegebener linker Seite, diejenigen Assertionen der rechten Seite zur A-Box hinzugefügt werden, die nicht bereits in dieser vorhanden sind. Im Beispiel ergibt sich für das gegebene

$$\Gamma := \{\mathsf{pipe}_1 : \mathsf{Pipe},\ \mathsf{oil}_1 : \mathsf{Oil},\ \mathsf{drops}(\mathsf{pipe}_1, \mathsf{oil}_1)\}$$

dementsprechend ein

$$\Delta := \{\mathsf{leakage}_1 : \mathsf{Leakage},\ \mathsf{hasFailure}(\mathsf{pipe}_1, \mathsf{leakage}_1),\ \mathsf{causesEmission}(\mathsf{leakage}_1, \mathsf{oil}_1)\}$$

das die Wissensbasis um die Erkenntnis erweitert, dass an dem betrachteten Rohr pipe_1 eine Leckage vorliegt.

12 *DL-sichere Regeln* (engl.: *DL-safe rules*) sind eine Untermenge der *Semantic Web Rule Language* (SWRL) und dadurch gekennzeichnet, dass sie nur auf A-Box-Individuen anwendbar sind. Dies garantiert ihre algorithmische Entscheidbarkeit. [MSS04]

Es muss dabei möglich sein, die Existenz bestimmter Assertionen der Prämisse als Voraussetzung für die abduktive Ableitung neuer Assertionen festzulegen. Ansonsten könnte es bei der Verwendung einer Relation mit einem großen Einsatzbereich und einer großen Reichweite zu falschen Schlussfolgerungen kommen. RacerPro ermöglicht es daher, bestimmte Assertionen der rechten Seite durch einen *Guard* (Wächter) vor einer Ableitung zu schützen. Schaltet man RacerPro zudem in den Modus `reuse-old`, so werden im Abduktionsprozess gar keine neuen Individuen mehr angelegt - einzig das Bilden von Relationen zwischen den bereits bestehenden Individuen ist dann möglich.

Andererseits können unterschiedliche Regeln dieselbe linke Seite aufweisen, wenn diese Assertion in unterschiedlichen Zusammenhängen beobachtet werden kann. Wie in diesem Fall zwischen mehreren möglichen Interpretationen eine Auswahl getroffen werden kann, wird in [ACD+06] erläutert.

Momentan ist die Entwicklung entscheidbarer und effizienter Algorithmen für das abduktive Schließen jedoch noch Gegenstand der Forschung (zum Beispiel im Projekt BOEMIE [ACD+06]) und wird erst von der nächsten Generation von Inferenzmaschinen unterstützt werden.

4 Maschinelle Extraktion von Informationen aus Text

Im diesem Kapitel wird untersucht, wie sich die Inhalte von menschlichen Generatoren formulierter Textfragmente durch einen maschinellen Rezeptor extrahieren lassen.

Da bis heute kein System zur Verfügung steht, das ein komplettes Verständnis für die Grammatik der menschlichen Sprache besitzt, kann die Verarbeitung natürlicher Sprache (engl.: *Natural Language Processing*, NLP) in der Praxis nur „oberflächlich" erfolgen (engl.: *Shallow Text Processing*). In diesem Bereich wurden bereits umfangreiche Forschungsarbeiten durchgeführt, so dass mittlerweile mehrere Anwendungen (z.B. SMES [NBB+97], SPPC [NP02]) und Frameworks (z.B. GATE [CMB+02], Ellogon [PKP+02]) für die Verarbeitung natürlichsprachlicher Texte zur Verfügung stehen.

Wenngleich sie sehr unterschiedlich umgesetzt werden, sind die grundlegenden Arbeitsschritte allen Systemen gemein: Nachdem in der Zeichensequenz des betrachteten Dokuments atomare Texteinheiten detektiert wurden (Kapitel 4.1), werden diese einer mitunter mehrstufigen linguistischen Analyse unterzogen (Kapitel 4.2). Die in einem Analyseschritt gewonnenen Informationen werden jeweils als *Annotationen* mit den entsprechenden Textbausteinen verknüpft und dienen als Grundlage für die nächste Analysestufe. Die Erkenntnisse der linguistischen Analyse werden anschließend genutzt, um die im Text behandelten Konzepte und Relationen zu extrahieren (Kapitel 4.3).

4.1 Erkennen atomarer Texteinheiten

Zu Beginn der Verarbeitung stellt sich der zu analysierende Text lediglich als eine Sequenz von Zeichen dar, die gemäß einer bestimmten Vorschrift kodiert sind[1]. Anhand bestimmter Sonderzeichen (wie zum Beispiel den Satzzeichen Punkt, Komma oder Leerzeichen) werden die Grenzen zusammengehöriger Zeichen detektiert und markiert. Diese Zeichenketten-Einheiten werden als *atomare Texteinheiten* (engl.: *Atomic Text Entity*, ATE) [KS99] oder auch *Token* [CMB+06] bezeichnet.

1 Die meisten Text verarbeitenden Systeme unterstützen den *Unicode Standard UTF-8* für Zeichenkodierung, in dem nicht nur die lateinischen Schriftzeichen und arabischen Ziffern erfasst sind, sondern auch viele andere internationale Schriftzeichensysteme [Yer03].

Type	Set	Start	End	Features
Token		0	3	{kind=word, length=3, orth=upperInitial, string=Das}
SpaceToken		3	4	{kind=space, length=1, string= }
Token		4	10	{kind=word, length=6, orth=upperInitial, string=Schiff}
SpaceToken		10	11	{kind=space, length=1, string= }
Token		11	14	{kind=word, length=3, orth=lowercase, string=hat}
SpaceToken		14	15	{kind=space, length=1, string= }
Token		15	18	{kind=word, length=3, orth=lowercase, string=ein}
SpaceToken		18	19	{kind=space, length=1, string= }
Token		19	24	{kind=word, length=5, orth=upperInitial, string=Segel}
Token		24	25	{kind=punctuation, length=1, string=.}

10 Annotations (0 selected)

Das Schiff hat ein Segel.

Abb. 4.1.1: Beispiel für die Annotationen des *GATE Unicode Tokenisers*

Durch die Auswertung definierter regulärer Ausdrücke wird den Token ein bestimmter Typ zugewiesen. Der in GATE enthaltene *Unicode Tokeniser* unterscheidet beispielsweise zwischen den folgenden Typen von Token [CMB+06]:

- **Word**: Eine Menge zusammenhängender Buchstaben mit eventuell enthaltenem Bindestrich. In dem der Annotation hinzugefügten Attribut `orth` wird der Typ weiter aufgeschlüsselt:

  `upperInitial` : erster Buchstabe in Großschrift, die restlichen in Kleinschrift
  `allCaps` : alle Buchstaben in Großschrift
  `lowerCase` : alle Buchstaben in Kleinschrift
  `mixedCaps` : sonstige Kombination von Buchstaben

- **Number:** Jede Kombination aufeinanderfolgender Ziffern. Allerdings werden Basis, Exponent und Mantisse einer Zahl nicht zusammengefasst. So wird die Zeichenkette der Zahl $3,2$ in ein `Number`-Token mit der Zeichenkette „3“, ein `Punctuation`-Token mit der Zeichenkette „,“ und ein weiteres `Number`-Token mit der Zeichenkette „2“ zerlegt.

- **Symbol:** Eine beliebige Anzahl aufeinanderfolgender Symbole (z.B. &, §). Speziell gekennzeichnet werden Währungssymbole, indem das Attribut `symbolkind` auf `currency` gesetzt wird. Sonstige Symbole erhalten kein Attribut.

- **Punctuation:** Jedes einzelne Satzzeichen ist ein getrenntes Token. Öffnende bzw. schließende Satzzeichen werden durch das Hinzufügen des Attributs `position` und dessen Setzen auf `startpunct` bzw. `endpunct` gekennzeichnet.

Zudem werden Leerräume (engl.: white spaces) mit dem speziellen Token `SpaceToken` gekennzeichnet. Dabei wird unterschieden, ob es sich um ein Leerzeichen (`kind=space`) oder ein Kontrollzeichen (`kind=control`) wie ein Tabulatorzeichen oder einen Zeilenumbruch handelt.

An die Zerlegung des Textes in Token kann direkt ein Analyseschritt angeschlossen werden, in dem anhand von Regeln überprüft wird, ob Token zusammengefasst werden können und so zum Beispiel eine Zahl mit ihren Dezimalstellen als eine Einheit dargestellt wird. Der von dem NLP-Framework GATE bereit gestellte `English Tokeniser` wandelt auf diese Weise Negativ-Konstrukte, wie „don't" von drei Token („don","" und „t") in zwei Token („do" und „n't) um und ermöglicht so in den nachfolgenden Arbeitsschritten das Identifizieren des Verbs und die Feststellung einer vorliegenden Negation.

4.2 Linguistische Analyse

Basierend auf die erfolgte Zerlegung in Token lässt sich der Text einer linguistischen Analyse unterziehen, in der die in ihm verwendeten *Wortformen* (engl.: *part of speech*, PoS) (Kapitel 4.2.2) bestimmt werden. Da die bei diesem Arbeitsschritt eingesetzten Algorithmen im hohem Maße an die Eigenarten einer bestimmten Sprache angepasst sind, ist es jedoch zunächst erforderlich, die zur Formulierung des Textes verwendete Sprache zu ermitteln (Kapitel 4.2.1).

4.2.1 Bestimmung der verwendeten Sprache

NLP-Frameworks wie GATE erfordern aufgrund der Sprachabhängigkeit der linguistischen Analyse die Vorgabe der im Text verwendete Sprache durch den Benutzer. Es existieren jedoch Softwaretools, mit denen sich die Sprache eines Textes automatisch bestimmen läßt. Ein Beispiel ist das Programm *TextCat*, das den in [CT94] vorgestellten Algorithmus implementiert und als Bibliothek für verschiedene Programmiersprachen zur Verfügung steht. Durch die Analyse einer Menge an Textdokumenten (dem *Trainingskorpus*), für die die jeweilige Sprache bekannt ist, identifiziert *TextCat* signifikante Merkmale der jeweiligen Sprache, wie Worthäufigkeiten, und ist anschließend in der Lage, die in zuvor unbekannten Texten verwendete Sprache anzugeben.

Type	Set	Start	End	Features
Token		0	3	{category=ART, kind=word, lemma=d, length=3, orth=upperInitial, string=Das}
SpaceToken		3	4	{kind=space, length=1, string= }
Token		4	10	{category=NN, kind=word, lemma=Schiff, length=6, orth=upperInitial, string=Schiff}
SpaceToken		10	11	{kind=space, length=1, string= }
Token		11	14	{category=VAFIN, kind=word, lemma=haben, length=3, orth=lowercase, string=hat}
SpaceToken		14	15	{kind=space, length=1, string= }
Token		15	18	{category=ART, kind=word, lemma=ein, length=3, orth=lowercase, string=ein}
SpaceToken		18	19	{kind=space, length=1, string= }
Token		19	24	{category=NN, kind=word, lemma=Segel, length=5, orth=upperInitial, string=Segel}
Token		24	25	{category=$., kind=punctuation, lemma=., length=1, string=.}

10 Annotations (0 selected)

Das Schiff hat ein Segel.

Abb. 4.2.1: Beispiel für die Annotationen des *TreeTaggers*

4.2.2 Bestimmen von Wortformen

Bei der *Wortformenanalyse* (engl.: *morphological analysis*) werden zunächst Lexikon gestützt zu den Zeichenketten der markierten *Wort*-Token alle potentiellen Wortformen bestimmt und als Annotationen mit dem Token verknüpft[2]. Für die verschiedenen Wortformen verwendeten Bezeichner sind in sogenannten *Tagsets* definiert, wie dem *STTS Tagset* [STS+95] für die deutsche Sprache oder dem *Penn-Treebank-Tagset* [San91] für das Englische. Anschließend wird über in kaskadierende und gewichtete endliche Automaten umgesetzte Regeln die genaue Wortform bestimmt [CMB+06]. Ein Beispiel für eine Regel: Wenn vor einem Token, das als mögliches Verb und als mögliches Nomen gekennzeichnet wurde, ein Artikel steht, dann ist das Token ein Nomen und die Kennzeichnung als mögliches Verb wird entfernt.

Die Software *TreeTagger* [CMB+06] liefert zudem zu jedem Wort noch dessen *Lemma* (die Grundform; bei einem Verb ist dies der Infinitiv) (siehe Abb. 4.2.1).

4.3 Extraktion von Informationen

Basierend auf den Annotationen der linguistischen Analyse kann die eigentliche Informationsextraktion durchgeführt werden. Das Ergebnis ist die Darstellung der textuell kodierten Aussagen in der formellen Syntax einer Beschreibungslogik und deren Hinzufügen zu der verwendeten Wissensbasis (siehe Abb. 4.3.1).

2 Dieser Vorgang wird in der Fachliteratur häufig als *Part-of-Speech-Tagging* oder kurz *PoS-Tagging* bezeichnet („to tag“ = „markieren“/“etikettieren“).

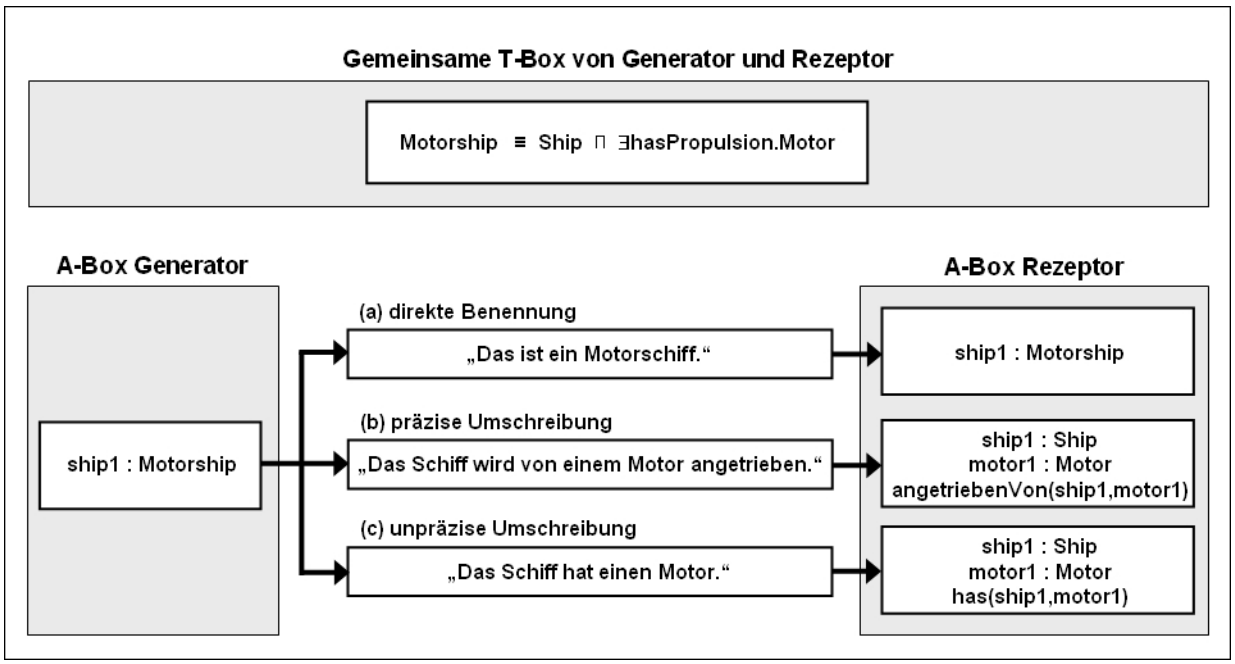

Abb. 4.3.1: Beispiel für T-Box und A-Box von Generator und Rezeptor

4.3.1 Extraktion von Konzepten

Da die in der Ontologie erfassten Konzepte Mengen von gleichartigen Objekten darstellen und Objekte linguistisch mit Nomen bezeichnet werden, können die Lemmata der durch die linguistische Analyse identifizierten Nomen als potentielle Deskriptoren von in der Ontologie definierten Konzepten betrachtet werden. Um die im Text benannten Konzepte zu identifizieren, muss also überprüft werden, welche der im Text enthaltenen Deskriptorenkandidaten $\mathsf{d_T} \in D_T$ in der Menge der Konzeptdeskriptoren D_C enthalten sind, und es muss eine Vorschrift m_C definiert werden, die die Konzeptdeskriptoren D_C auf die Menge der Konzepte C abbildet (siehe Abb. 4.3.2):

$$m_C : D_C \mapsto C$$

Um die Übereinstimmung eines Nomen-Lemmas mit dem Deskriptor eines Konzepts zu überprüfen, kann ein sogenannter *Gazetteer* eingesetzt werden [KPO+05]. Als Gazetteer wurden ursprünglich geographische Namensverzeichnisse bezeichnet, in denen beispielsweise Städtenamen länderweise in separaten Listen erfasst wurden. Erfasst man anstatt von Ortsnamen die Deskriptoren jeweils eines Konzepts in einer Liste und vermerkt in einem Index, welche Liste mit welchem Konzept assoziiert ist, so kann von der Liste, in der eine Übereinstimmung mit dem Lemma gefunden wurde, auf das im Text repräsentierte Konzept geschlossen werden. Daraufhin kann in der Wissensbasis ein Individuum des erkannten Konzepts angelegt und mit dem entsprechenden Textfragment assoziiert werden[3].

3 Die Assoziation eines Textfragments mit den aufgrund seiner inhaltlichen Analyse erstellten Individuen

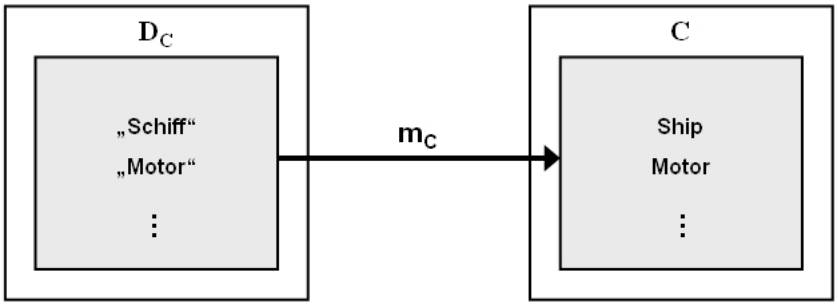

Abb. 4.3.2: Abbildung von Deskriptoren $\mathsf{d_c} \in D_C$ auf Konzepte $\mathsf{C} \in C$

Die Erstellung der Gazetteer-Listen kann durch elektronische Wörterbücher, wie *WordNet* [KS99] (Englisch) oder *GermaNet* [KL02] (Deutsch), unterstützt werden. Diese liefern zu jedem ihnen bekannten Sinn eines Wortes eine Menge an synonymen Begriffen (ein sogenanntes *SynSet*). Auf diese Weise muss der Domänenexperte bei der Erstellung der Ontologie nur noch wenige Konzeptdeskriptoren vorgeben (jedoch minimal einen pro zu unterstützender Sprache) und die Gazetteer-Listen können semi-automatisch unter Einbezug der in den Wörterbüchern hinterlegten Synonyme generiert werden. Eine völlige Automation ist jedoch nicht möglich, da der Domänenexperte vorgeben muss, welchen Wortsinn der von ihm gewählte Deskriptor besitzt (beispielsweise kann der Begriff „Bank“ kontextabhängig die Bedeutung eines Finanzinstituts oder die einer Sitzgelegenheit einnehmen).

Gerade die mögliche Mehrdeutigkeit von Deskriptoren stellt ein Problem für der Suche nach Übereinstimmung von Lemma und Deskriptor dar. Methoden zum Auflösen solcher Mehrdeutigkeiten werden in [DEG+03, AR95, RSM94, MNV02] vorgestellt. Ihnen gemeinsam ist, dass sie ein Wort im Zusammenhang mit den es umgebenden Worten betrachten und aus diesem Kontext den wahrscheinlichen Sinn des Wortes bestimmen.

4.3.2 Extraktion von Relationen

Wesentlich komplexer, als das Erkennen von im Text erwähnten Konzepten, ist die Extraktion der intendierten Beziehung zwischen diesen. Aus der Annahme, dass Worte nicht zufällig zusammen auftreten, sondern der Generator mit ihrer Zusammenstellung eine bestimmte Aussage beabsichtigt [HQW06], läßt sich zunächst ableiten, dass zwei zusammen erwähnte Konzepte generell in Beziehung stehen.

wird in Kapitel 5.2 beschrieben.

In einem naiven Ansatz ließe sich also eine allgemeinste Relation associatedWith einführen, mit der alle Individuen verknüpft werden, die für die in einem Textfragment auftretenden Konzepte erstellt wurden. Damit findet jedoch eine erhebliche Abstraktion statt, die realen Gegebenheiten selten gerecht wird. Stehen zwei Konzepte in mehreren Beziehungen zueinander, so kann zwischen diesen Relationen nicht unterschieden werden und falsche Schlussfolgerungen sind möglich. Das folgende Beispiel verdeutlicht dies: Ist ein stählernes Schiff mit Hilfe der spezielleren Relation $\mathsf{isMadeOf} \sqsubseteq \mathsf{associatedWith}$ definiert als

$$\mathsf{StealShip} \quad \equiv \quad \mathsf{Ship} \sqcap \exists \mathsf{isMadeOf}.\mathsf{Steal}$$

und ergeben sich aus der Analyse des Textfragments „*Das Schiff ist aus Stahl.*“ die Assertionen

$$\begin{aligned} \mathsf{ship}_1 \quad &: \quad \mathsf{Ship} \\ \mathsf{steal}_1 \quad &: \quad \mathsf{Steal} \\ \mathsf{associatedWith}(\mathsf{ship}_1, \mathsf{steal}_1) & \end{aligned}$$

so kann die intuitive Schlussfolgerung, dass ship_1 ein StealShip ist, nicht abgeleitet werden. Steht hingegen zur Definition von StealShip nur die allgemeine Relation associatedWith zur Verfügung, so dass

$$\mathsf{StealShip} \quad \equiv \quad \mathsf{Ship} \sqcap \exists \mathsf{associatedWith}.\mathsf{Steal}$$

so ist die Schlussfolgerung

$$\Longrightarrow \quad \mathsf{ship}_1 \quad : \quad \mathsf{StealShip}$$

möglich. Dieselben Assertionen ergeben sich jedoch auch aus der textuellen Aussage „*Das Schiff transportiert Stahl.*“ und würden hier ebenfalls die zwar formal korrekte, nun jedoch inhaltlich falsche Schlussfolgerung ergeben, dass dieses Schiff ein StealShip sei.

Dies zeigt, dass es unerlässlich ist, zur Konzept-Definition ausreichend spezielle Relationen zu verwenden. Um bei der Textanalyse deduktive Schlussfolgerungen ziehen zu können, ist es dann wiederum erforderlich, dass diese speziellen Relationen auch aus dem Text extrahiert werden.

Es wird also nicht die in Abb. 4.3.3 skizzierte primitive Abbildungsvorschrift m_{R1} gesucht, sondern eine komplexere Vorschrift m_{R2}, die Relationen-Deskriptoren $\mathsf{d_R} \in D_R$ in Relationen $\mathsf{R} \in R$ überführt:

$$m_{R2} : D_R \mapsto R$$

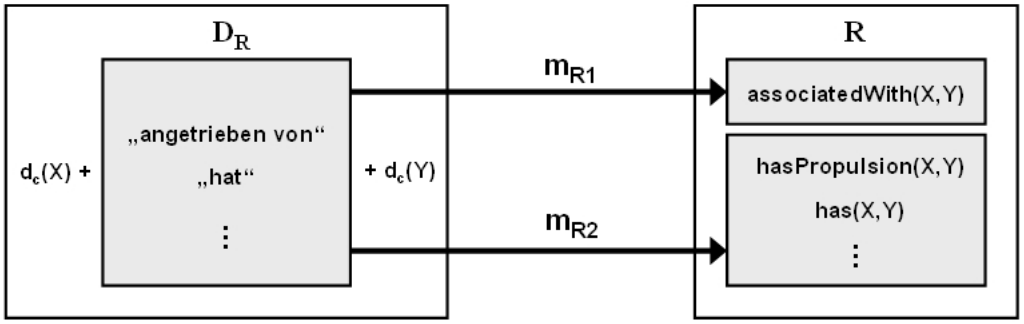

Abb. 4.3.3: Abbildung von Deskriptoren $\mathsf{d_R} \in D_R$ in Relationen $\mathsf{R} \in R$

Verben als Relationsdeskriptoren

Stellt ein Konzept das Subjekt eines Satzes dar und ein weiteres das Objekt, so ist die durch den Satz implizierte Beziehung zwischen ihnen durch das Prädikat gegeben [Dud98]. Wie Eingang des Kapitels erwähnt gibt es heute jedoch noch keine Systeme, die ein vollständiges grammatikalisches Verständnis für die menschliche Sprache besitzen und so kann die Beziehung nicht mit allen möglichen Nuancen erschlossen werden. Es ist jedoch möglich, zumindest einen wesentlichen Teil des Prädikats zu erschließen - das Verb. Durch die lexikalische Analyse liegt die Information vor, welches Token ein Verb enthält und zudem ist das Lemma dieses Verbs bekannt. Analog zu der Identifikation von Konzepten, lassen sich Relationen also näherungsweise durch Verben beschreiben. Werden diejenigen Verben, die eine Relation repräsentieren, als deren Deskriptoren in Gazetteer-Listen erfasst, so ist analog zu dem Abgleich von Konzeptdeskriptoren ein Abgleich von Relationsdeskriptoren möglich[4]. Damit werden zwar nicht alle möglichen Beziehungen erfasst, die mit der menschlichen Sprache ausgedrückt werden können, es kann jedoch ein großer Anteil von Relationen identifiziert werden.

Spezialisierung von Relationen durch Abduktion

Relationsdeskriptoren können eingesetzt werden, um die Verknüpfung zweier innerhalb eines Satzes auftretende Konzeptdeskriptoren in Beziehung zu setzen. Treten zwei Konzeptdeskriptoren jedoch in unterschiedlichen Sätzen oder gar in unterschiedlichen HOTs auf, so lassen sich keine spezielleren Relationen erkennen und einzig eine allgemeine associatedWith Beziehung kann zwischen ihnen gesetzt werden.

Man kann jedoch das Erkennen einer Konstellation von in Beziehung stehender Konzepte als die Beobachtung eines Zustandes und die Suche nach einer spezielleren Relation als die Suche nach einer Erklärung für diese Situation auffassen.

4 Der Abgleich von Deskriptoren wird im Englischen als *Descriptor Matching* bezeichnet.

Diese Erkenntnis brachte die Idee, Regeln zu definieren, mit deren Hilfe eine speziellere Relation abduktiv geschlossen werden kann. Definiert man beispielsweise die Regel

$$\begin{aligned} \mathsf{associatedWith}(\mathsf{X},\mathsf{Y}) \quad \leftarrow \quad & \mathsf{Ship}(\mathsf{X}),\ \mathsf{Sail}(\mathsf{Y}),\ \mathsf{Transport}(\mathsf{Z}), \\ & \mathsf{associatedWith}(\mathsf{X},\mathsf{Z}),\ \mathsf{associatedWith}(\mathsf{Y},\mathsf{Z}), \\ & \mathsf{transports}(\mathsf{X},\mathsf{Y}) \end{aligned}$$

und extrahiert aus dem Text „*Das Schiff wird für den Transport von Segeln verwendet.*“ die Assertionen

$$\begin{aligned} \mathsf{ship}_1 &: \mathsf{Ship} \\ \mathsf{sail}_1 &: \mathsf{Sail} \\ \mathsf{transport}_1 &: \mathsf{Transport} \\ \mathsf{associatedWith}(\mathsf{ship}_1, \mathsf{transport}_1) & \\ \mathsf{associatedWith}(\mathsf{sail}_1, \mathsf{transport}_1) & \end{aligned}$$

so lässt es sich abduktiv schließen, dass ebenfalls $\mathsf{transports}(\mathsf{ship}_1, \mathsf{sail}_1)$ gelten muss. Durch das Setzen eines Guards auf alle Assertionen der Prämisse ausser der Assertion $\mathsf{transports}(\mathsf{X},\mathsf{Y})$ wird verhindert, dass durch den Abduktionsprozess bei dem Text „*Das Schiff hat ein Segel.*“ aus den erkannten Assertionen

$$\begin{aligned} \mathsf{ship}_2 &: \mathsf{Ship} \\ \mathsf{sail}_2 &: \mathsf{Sail} \\ \mathsf{associatedWith}(\mathsf{ship}_2, \mathsf{transport}_2) & \end{aligned}$$

fälschlicherweise ein

$$\Delta := \{\mathsf{transport}_2 : \mathsf{Transport},\ \mathsf{associatedWith}(\mathsf{ship}_2, \mathsf{transport}_2),\ \mathsf{transports}(\mathsf{ship}_2, \mathsf{sail}_2)\}$$

abgeleitet wird.

Spezialisierung von Relationen durch die Auswertung von Regeln

Da bei der Relations-Spezialisierung jeweils nur eine einzelne Assertion vorgenommen werden soll, ist es ebenfalls denkbar, eine einfache DL-sichere Regel auswerten zu lassen, bei der bei Erfüllung der Prämisse die als Konklusion angegebene Assertion vorgenommen wird:

$$\begin{aligned} \mathsf{transports}(\mathsf{X},\mathsf{Y}) \quad \leftarrow \quad & \mathsf{Ship}(\mathsf{X}),\ \mathsf{Sail}(\mathsf{Y}),\ \mathsf{Transport}(\mathsf{Z}), \\ & \mathsf{associatedWith}(\mathsf{X},\mathsf{Y}),\ \mathsf{associatedWith}(\mathsf{X},\mathsf{Z}),\ \mathsf{associatedWith}(\mathsf{Y},\mathsf{Z}) \end{aligned}$$

5 System zur Klassifikation von Textfragmenten

Basierend auf den in den Kapiteln 2 bis 4 erarbeiteten Grundlagen wurde ein System zur automatischen wissensbasierten Klassifikation hierarchisch angeordneten Textfragmente entwickelt: *HOTClassifier*. In diesem Kapitel wird zunächst erläutert nach welchem Ansatz Domänenwissen für die Verwendung in HOTClassifier konzeptualisiert wird (Abschnitt 5.1) und wie die Klassifikation von Textfragmenten erfolgt (Abschnitt 5.2), bevor die Architektur und die Implementation des im Zuge dieser Arbeit erstellten Prototyps vorgestellt wird (Abschnitt 5.3).

5.1 Repräsentation von Domänenwissen

Für die inhaltliche Klassifikation von Textfragmenten ist es erforderlich, thematische Klassen festzulegen, denen die Textfragmente zugeordnet werden können. Dieses erfolgt durch die Erstellung einer Ontologie, deren Konzepte die vom Klassifikationssystem verwendeten thematischen Klassen darstellen. Um ihre gemeinsame Funktion als inhaltliche Kategorien zum Ausdruck zu bringen, werden alle thematischen Konzepte als Spezialisierungen des generellen Konzepts Topic erstellt, für das die allgemeine symmetrische[1] Rolle associatedWith definiert ist (siehe Abb. 5.1.1).

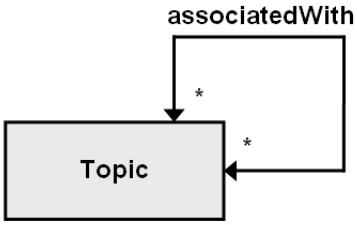

Abb. 5.1.1: Konzept *Topic*, das generellste Konzept aller Themen-Konzepte

1 Besteht eine symmetrische Relation R zwischen einem Individuum i und einem Individuum j, so ist j ebenfalls via R mit i verbunden.

Die Sub-Konzepte von Topic modellieren physische oder logische Objekte, die in dem von dem zu klassifizierenden Textfragmenten behandelten Wissensgebiet auftreten. Die zwischen ihnen bestehenden Beziehungen werden durch Sub-Rollen der allgemeinen Rolle associatedWith abgebildet (diese müssen nicht zwangläufig symmetrisch sein).

Bei der Konzeptualisierung der Domäne sollte vordergründig erörtert werden, was die Besonderheit eines Konzepts auszeichnet und es von anderen, insbesondere seinem Elternkonzept und seinen Geschwistern, unterscheidet. Ziel ist es, eine oder auch mehrere von einander unabhängige notwendige und hinreichende Bedingungen festzulegen, deren Erfüllung einen Schluss auf die Zugehörigkeit eines Individuums zu diesem Konzept ermöglicht. In den allermeisten Fällen lässt sich ein Konzept dabei durch die Konjunktion von einem allgemeineren Konzept und einer Anzahl erforderlicher Rollenrestriktionen darstellen, wie in dem bereits eingeführten Beispiel eines Segelschiffs geschehen:

$$\mathsf{SailingShip} \equiv \mathsf{Ship} \sqcap \exists \mathsf{hasPart}.\mathsf{Sail}$$

Die definierten Bedingungen sollten dabei aus einem minimal erforderlichen Satz an Rollenrestriktionen bestehen, so dass ein Schließen auf die Konzeptzugehörigkeit eines Individuums mit einer minimalen Anzahl an als A-Box Assertionen formulierten Informationen möglich ist.

Das Festlegen hinreichender Bedingungen ist jedoch nicht immer problemlos möglich. Betrachten wir einmal das folgende Beispiel: Für ein Motorschiff ist es sicherlich eine notwendige Bedingung, dass dieses einen Motor besitzt:

$$\mathsf{MotorShip} \sqsubseteq \mathsf{Ship} \sqcap \exists \mathsf{hasPart}.\mathsf{Motor}$$

Allerdings können auch Segelschiffe einen Motor besitzen, den sie als Hilfsantrieb verwenden. Das einfache Vorhandensein eines Motors ist also nicht hinreichend für den Schluss, dass es sich bei einem betrachteten Schiff um ein Motorschiff handeln muss. Die Definition, dass ein Motorschiff ein Schiff ist, das einen Motor, jedoch kein Segel besitzt

$$\mathsf{MotorShip} \equiv \mathsf{Ship} \sqcap \exists \mathsf{hasPart}.\mathsf{Motor} \sqcap \neg (\exists \mathsf{hasPart}.\mathsf{Sail})$$

ist zur Klassifikation von Textfragmenten ungeeignet, da (wie in Kapitel 3.2.1 erläutert) die *Open World Assumption* des Reasoners in diesem Fall kein deduktives Ableiten einer Konzeptzugehörigkeit gestattet.

Eventuell ist es jedoch möglich, die in der notwendigen und hinreichenden Bedingung verwendete Rollenrestriktion so zu spezialisieren, dass die Entscheidbarkeit gegeben ist. Im Beispiel des Motorschiffs könnte man einerseits festlegen, dass die Rolle hasPart zu einer bestimmten Art von Antriebsmaschine bestehen muss, von der bekannt ist, das sie nur als Hauptantrieb verwendet wird und ihr Einsatz als Hilfsmaschine ausgeschlossen ist:

$$\mathsf{MotorShip} \equiv \mathsf{Ship} \sqcap \exists \mathsf{hasPart}.\mathsf{MainEngine}$$

Andererseits könnte man die Relation selbst spezialisieren und zum Beispiel aussagen, dass das Motorschiff nicht nur einen Motor besitzen muss (hasPart), sondern dass dieser auch als Antrieb verwendet wird (hasPropulsion):

$$\mathsf{MotorShip} \equiv \mathsf{Ship} \sqcap \exists \mathsf{hasPropulsion.Engine}$$

Eine Überprüfung des korrekten Aufbaus der T-Box kann erfolgen, indem man diese mit Hilfe eines Reasoners klassifiziert und überprüft, ob keine Inkonsistenzen auftreten. Als weiteren Schritt kann man für jedes definierte Konzept ein Individuum anlegen, dass exakt die vorgegebenen notwendigen und hinreichenden Bedingungen erfüllt. Für dieses darf bei korrektem Aufbau der T-Box dann nur der beabsichtigte direkte Typ abgeleitet werden. Wird das Individuum einer weiteren Klasse zugewiesen, so sind die bestehenden Restriktionen nicht ausreichend.

Neben notwendigen und hinreichenden Bedingungen werden für Konzepte *Label Properties* definiert, die Deskriptoren des Konzepts und deren Sprache angeben. Diese werden später von `HOTClassifier` ausgewertet und dienen als Grundlage zum Aufbau von Deskriptorenlisten (siehe Abschnitt 5.3).

Die Erstellung Beschreibungslogik sicherer Regeln als Grundlage für eine abduktive Ableitung von Erklärungen für beobachtete Zustände wurde im gegenwärtigen Konzept noch nicht berücksichtigt, da die zum Zeitpunkt dieser Arbeit aktuellen Versionen von Inferenzmaschinen den Abduktionsprozess noch nicht unterstützen.

5.2 Klassifikation hierarchisch angeordneter Textfragmente

Die eigentliche Klassifikation eines Textfragments erfolgt durch seine Assoziation mit den Konzeptnamen, die für die während seiner Verarbeitung generierten A-Box-Individuen abgeleitet werden können. Die Verknüpfung von Textfragmenten und Konzeptnamen lässt sich beispielsweise mit Hilfe von Tabellen in einem relationalen Datenbankmanagementsystem (RDBMS) erreichen. Ein anderer Ansatz ist es, die Fähigkeiten des verwendeten Reasoners zur Informationswiedergewinnung (engl.: *information retrieval*) auszunutzen:

Reasoner unterstützen mehr oder weniger mächtige Abfragesprachen, mit deren Hilfe sich in der Ontologie und in der Wissenbasis enthaltene Informationen erfragen lassen (z.B. nRQL bei RacerPro [HMW05], SPARQL bei Pellet [PS006]). Werden vom Klassifikationssystem erfasste Textfragmente als Individuen eines geeigneten Konzepts repräsentiert, so können diese über in diesen Abfragesprachen formulierte Suchkriterien von der Inferenzmaschine ausgewählt und zurück geliefert werden.

Der Ansatz, Dokumente als Individuen in einer A-Box abzubilden, wurde bereits in mehreren Projekten [Kun05, VFC05, MMV02] formuliert. Die im Zuge dieser Projekte erstellten Konzeptualisierungen von Textdokumenten sind jedoch nicht ausreichend für den Kontext dieser Arbeit, da in ihnen Dokumente isoliert betrachteten werden und anders als HOTs keine inhaltlichen Abhängigkeiten und strukturellen Beziehungen zueinander aufweisen.

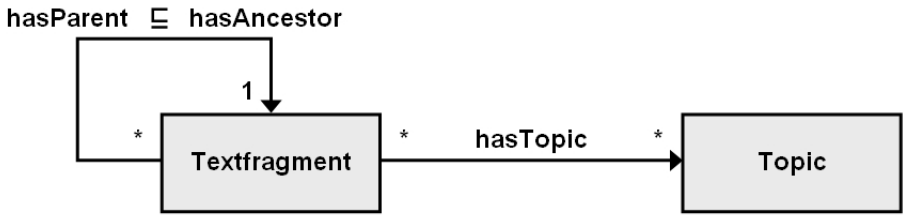

Abb. 5.2.1: Relationen zwischen den Konzepten Textfragment und Topic

Der naive Ansatz, Textfragmente durch eine Relation hasTopic mit den bei ihrer inhaltlichen Analyse erzeugten Individuen von Sub-Konzepten des Konzepts Topic zu verknüpfen und ihre hierarchische Anordnung durch eine Relation hasParent zu modellieren, die jeweils zwischen dem A-Box Individuum eines Textfragments und dem Individuum seines Elternelements gesetzt wird (siehe Abb. 5.2.1), ist nicht ausreichend: Zwar wird durch die Einführung einer transitiven[2] Super-Relation hasAncestor erreicht, dass ein Textfragment implizit mit allen ihn hierarchisch übergeordneten Textfragmenten verknüpft ist, der Umstand, dass der Inhalt eines Textfragments im Zusammenhang der Inhalte dieser ihm hierarchisch übergeordneten Textfragmente zu betrachten ist, ist jedoch nicht erfasst. Wird mit einer Anfrage nach allen Textfragmenten gesucht, die eine bestimmte Kombination von Themen behandeln (die hasTopic Relationen zu Individuen bestimmter Konzepte besitzen), so werden nur diejenigen Textfragmente zurück geliefert, die direkt die genannten Themen beinhalten. Relevante Textfragmente, die nur eine Untermenge der angefragten Themen direkt behandeln, jedoch übergeordnete Textfragmente besitzen, die den restlichen Themen zugeordnet wurden, sind nicht im Resultat der Anfrage enthalten. Sollen auch diese erfasst werden, so ist dies explizit in der Anfrage zu formulieren. Werden beispielsweise alle Textfragmente gesucht, die die Themen Ship, Engine und Leakage zum Inhalt haben, so kann dies durch die folgenden drei nRQL-Abfragen ausgedrückt werden (die Variable x[3] beinhaltet die ermittelten Textfragmente):

2 *Transitivität von Rollen*: Sind die Individuen i und j durch eine transitive Rolle R verknüpft, und besteht die Rolle R ebenfalls zwischen j und dem Individuum k, so kann abgeleitet werden, dass i via R ebenfalls mit k in Beziehung steht.

3 RacerPro bindet Individuen normalerweise an exakt eine Variable (*injektive Variable*). Durch ein einer Variable vorangestelltes $ wird erlaubt, dass an diese Variable gebundene Individuen gleichzeitig an andere Variablen gebunden werden dürfen (*gewöhnliche Variable*).

- Suche nach direkt den Themen zugeordneten Textfragmenten:

```
(retrieve (?x)
   (and
      (?x |Ship| |hasTopic|)
      (?x |Engine| |hasTopic|)
      (?x |Leakage| |hasTopic|)))
```

- Suche nach Textfragmenten, die zwei Themen direkt zugeordnet sind und einem Textfragment untergeordnet sind, das das dritte Thema beinhaltet:

```
(retrieve ($?x)
   (or
      (and ($?x |Ship| |hasTopic|) ($?x |Engine| |hasTopic|)
         ($?y |Leakage| |hasTopic|) ($?x $?y |hasAncestor|))
      (and ($?x |Ship| |hasTopic|) ($?y |Engine| |hasTopic|)
         ($?x |Leakage| |hasTopic|) ($?x $?y |hasAncestor|))
      (and ($?y |Ship| |hasTopic|) ($?x |Engine| |hasTopic|)
         ($?x |Leakage| |hasTopic|) ($?x $?y |hasAncestor|))))
```

- Suche nach Textfragmenten, die einem Thema direkt zugeordnet sind und Textfragmenten untergeordnet sind, die die anderen beiden Themen beinhalten:

```
(retrieve ($?x)
   (or
      (and ($?x |Ship| |hasTopic|) ($?y |Engine| |hasTopic|)
         ($?z |Leakage| |hasTopic|)
         ($?x $?y |hasAncestor|) ($?x $?z |hasAncestor|))
      (and ($?y |Ship| |hasTopic|) ($?x |Engine| |hasTopic|)
         ($?z |Leakage| |hasTopic|)
         ($?x $?y |hasAncestor|) ($?x $?z |hasAncestor|))
      (and ($?y |Ship| |hasTopic|) ($?z |Engine| |hasTopic|)
         ($?x |Leakage| |hasTopic|)
         ($?x $?y |hasAncestor|) ($?x $?z |hasAncestor|))))
```

Diese drei Abfragen lassen sich wie nachfolgend dargestellt zu einer einzigen Abfrage kombinieren:

```
(retrieve ($?x)
   (or
      (and ($?x |Ship| |hasTopic|) ($?x |Engine| |hasTopic|)
         ($?x |Leakage| |hasTopic|))
      (and ($?x |Ship| |hasTopic|) ($?x |Engine| |hasTopic|)
         ($?y |Leakage| |hasTopic|) ($?x $?y |hasAncestor|))
      (and ($?x |Ship| |hasTopic|) ($?y |Engine| |hasTopic|)
         ($?x |Leakage| |hasTopic|) ($?x $?y |hasAncestor|))
      (and ($?y |Ship| |hasTopic|) ($?x |Engine| |hasTopic|)
         ($?x |Leakage| |hasTopic|) ($?x $?y |hasAncestor|))
      (and ($?x |Ship| |hasTopic|) ($?y |Engine| |hasTopic|)
         ($?z |Leakage| |hasTopic|)
         ($?x $?y |hasAncestor|)($?x $?z |hasAncestor|))
      (and ($?y |Ship| |hasTopic|) ($?x |Engine| |hasTopic|)
         ($?z |Leakage| |hasTopic|)
         ($?x $?y |hasAncestor|)($?x $?z |hasAncestor|))
      (and ($?y |Ship| |hasTopic|) ($?z |Engine| |hasTopic|)
         ($?x |Leakage| |hasTopic|)
         ($?x $?y |hasAncestor|)($?x $?z |hasAncestor|))
   )
)
```

Allgemein sind bei dieser Modellierung für die Abfrage aller Textfragmente, die eine Kombination von n Themen beinhalten, $n \cdot (n-1)+1$ disjunkte Konjunktionsterme mit insgesamt $n + n \cdot \sum_{i=1}^{n-1} (n+1)$ Rollen-Abfragetermen (engl.: *role query atoms*) erforderlich.

Durch die Einführung eines Konzepts Content als Super-Konzept der schon bekannten Konzepte Textfragment und Topic und der transitiven Relation hasContent als Super-Relation zu hasTopic lässt sich jedoch der Gesamtkontext eines Textfragments erfassen und die Komplexität der inhaltlichen Suche wesentlich reduzieren, ohne dass eine höhere Anzahl an Assertionen in der Wissensbasis vorgenommen werden muss (siehe Abb. 5.2.2).

Die hasContent-Relation zwischen Textfragmenten besitzt als Domain das untergeordnete Textfragment und als Range das übergeordnete Textfragment und setzt das untergeordnete Textfragment nicht nur mit allen in der Hierarchie höher angeordneten Textfragmenten, sondern auch mit allen Individuen in Beziehung, die eine hasTopic-Rolle zu diesen einnehmen. Die Menge der über die Relation hasContent verknüpften Individuen des Konzepts Topic stellt somit den inhaltlichen Kontext eines Textfragments dar.

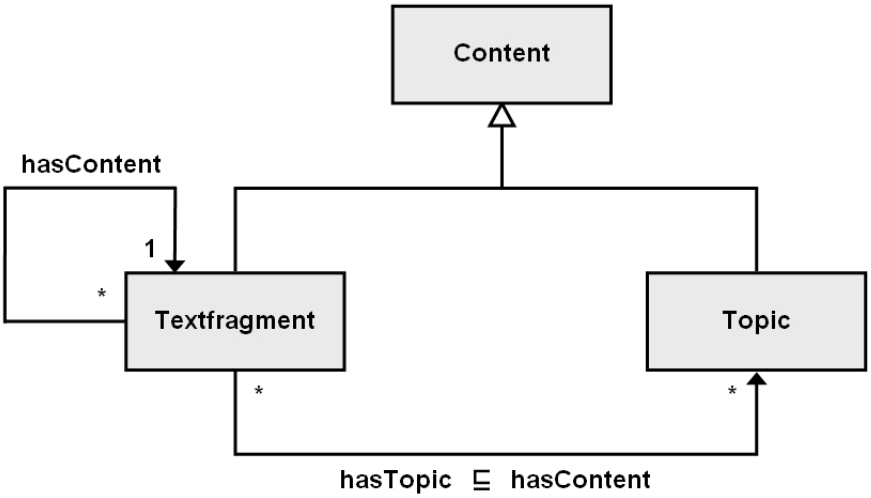

Abb. 5.2.2: Einführung des generellen Konzepts *Content*

Ein Beispiel für die bei der Verarbeitung zweier HOTs erzeugten Assertionen ist in Abb. 5.2.3 dargestellt. Eine Beispiel-Ontologie, die diesen Modellierungsansatz verwendet, befindet sich in Anhang A.

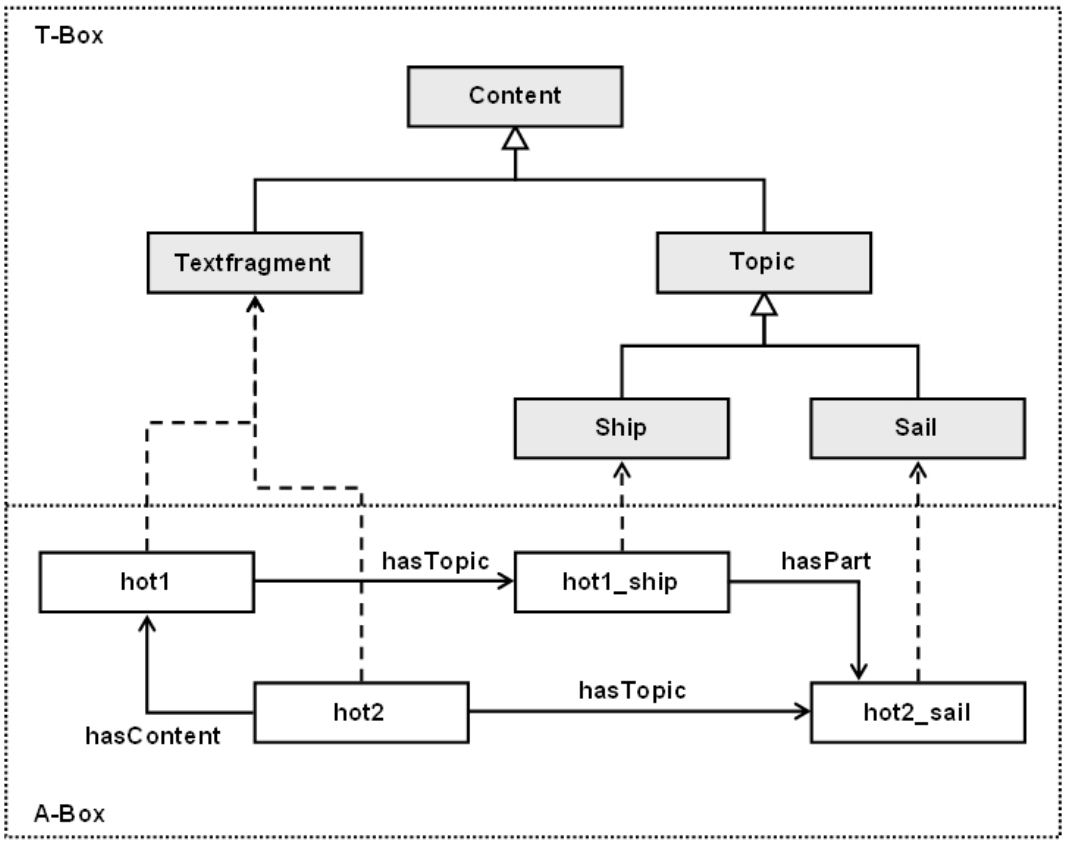

Abb. 5.2.3: Beispiel für die bei der Verarbeitung zweier HOTs erzeugten Assertionen

Bei diesem Modellierungsansatz sind für die Abfrage aller Textfragmente, die eine Kombination von n Themen beinhalten, nur noch n disjunkte Konjunktionsterme mit insgesamt $n \cdot 2$ Rollen-Abfragetermen erforderlich:

```
(retrieve (?x)
   (or
      (and (?x |Ship| |hasTopic|) (?x |Engine| |hasContent|)
         (?x |Leakage| |hasContent|))
      (and (?x |Ship| |hasContent|) (?x |Engine| |hasTopic|)
         (?x |Leakage| |hasContent|))
      (and (?x |Ship| |hasContent|) (?x |Engine| |hasContent|)
         (?x |Leakage| |hasTopic|))))
```

Die reduzierte Komplexität wirkt sich deutlich auf die von der Inferenzmaschine zur Beantwortung der Abfrage benötigte Zeitdauer aus: auf dem Testsystem (siehe Anhang B) konnte die Abfrage mindestens 65% schneller von RacerPro beantwortet werden (siehe Tab. 5.2.1).

Abfrageart	Modell *ohne* Konzept Content	Modell *mit* Konzept Content
kombinierte Abfrage	77151 ms = 1 min 17 s 151 ms	341 ms
separierte Abfragen	$1 \cdot 20 + 3 \cdot 20 + 3 \cdot 30 = 170$ ms	$3 \cdot 20 = 60$ ms

Tab. 5.2.1: Antwortzeiten von RacerPro zu der Beispielabfrage

5.3 HOTClassifier

Das entwickelte Konzept zur wissensbasierten inhaltlichen Klassifikation hierarchisch angeordneter Textfragmente wurde durch die Implementation des Softwaresystems `HOTClassifier` prototypisch realisiert. Aufgrund der für das Projekt vorgegebenen zeitlichen Beschränkung wurden in der gegenwärtigen ersten Version folgende Vereinfachungen vorgenommen:

- Textfragmente werden nicht als Individuen den Konzepts `Textfragment` in der Wissensbasis repräsentiert. Stattdessen werden die direkten Typen der bei der inhaltlichen Analyse kreierten Individuen in einem Datenfeld des Textfragments erfasst.
- Eine Spezialisierung der durch den Inhalt eines Textfragments implizierten Relationen findet noch nicht statt. Es wird bislang einzig mit der allgemeinsten Relation associatedWith gearbeitet.

Dem Lauf eines zu klassifizierenden HOTs folgend, wird das System nachfolgend vorgestellt: Zur Erfassung und Verwaltung der Textfragmente wird das Open Source Content-Management-System (CMS) *Plone* eingesetzt (siehe Abb. 5.3.1), das auf der Basis des Application Servers *Zope* läuft und seine Inhalte auf einem *PostgreSQL* Datenbank-Server ablegt. In der Programmiersprache *Python* wurde ein neuer Inhaltstyp `Textelement` zur Repräsentation von HOTs erstellt (siehe Anhang C). Beim erstmaligen Speichern eines neu angelegten HOTs wird dessen textueller Inhalt zusammen mit seiner von Plone erzeugten eindeutigen Identifikationsnummer (engl.: unique identifier, UID) und der UID seines Elternelementes per Transport Control Protocol (TCP) an den `HOTClassifier`-Server übermittelt. Dieser instanziiert das Textfragment in Vorbereitung auf die nachfolgenden Arbeitsschritte zunächst als ein Java-Objekt der Klasse `HOT`. Durch die serielle Übergabe das `HOT`-Objekts an in geeigneter Weise konfigurierte Module des NLP-Frameworks GATE (siehe Abb. 5.3.2) werden

- atomare Texteinheiten detektiert (Modul `Tokeniser`),
- Wortformen ermittelt (Modul `PoSTagger`),
- alle Token herausgefiltert, die keine Nomen sind und daher keine Konzeptdeskriptoren sein können (Modul `TokenFilter`) und
- die verbliebenen Nomen-Token als Deskriptorkandidaten mit den Konzeptdeskriptoren abgeglichen (Modul `Gazetteer`).

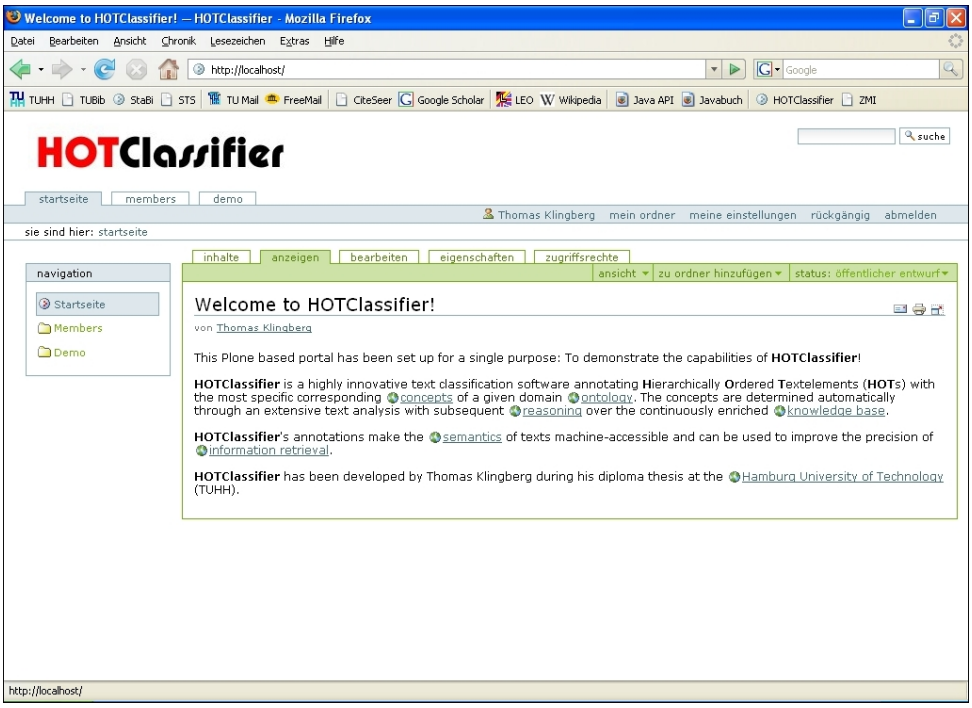

Abb. 5.3.1: Startseite des `HOTClassifier`-Portals

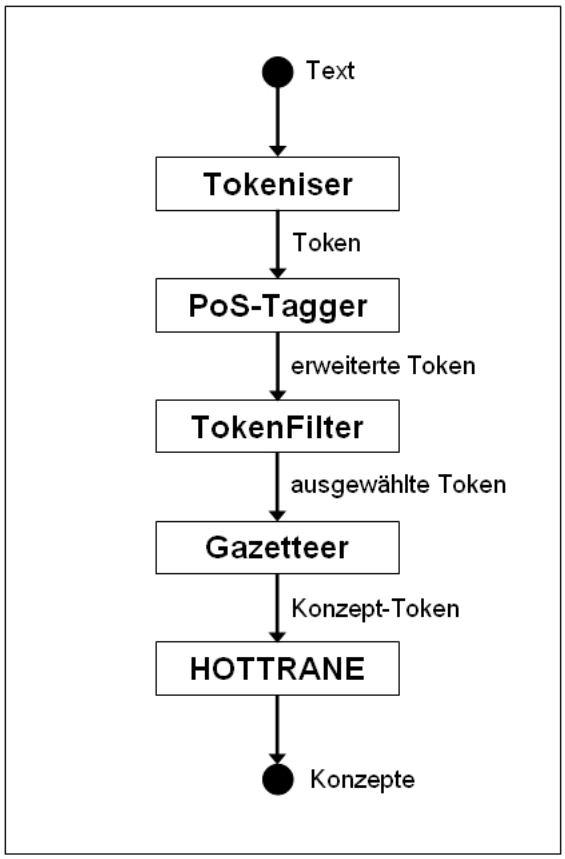

Abb. 5.3.2: Klassifikation von Textfragmenten mit `HOTClassifier`

Die Erstellung der benötigten Gazetteer-Listen wird durch das entwickelte Java-Programm `OWL2LST` erheblich vereinfacht: `OWL2LST` liest nacheinander alle Konzepte der in einer OWL-Datei hinterlegten Ontologie aus, ermittelt ihre in Label Properties annotierten Deskriptoren und schreibt diese in die entsprechende Liste. Für englische Deskriptoren werden zusätzlich alle in WordNet vorhandenen SynSets ermittelt und dem Benutzer zur Auswahl angezeigt. Hat der Benutzer das SynSet ausgewählt, das dem von ihm beabsichtigten Sinn des Deskriptors entspricht, so werden alle in diesem enthaltenen Synonyme ebenfalls in die entsprechende Gazetteer-Liste aufgenommen.

Die vom Gazetteer ermittelten Konzeptnamen werden anschließend dem als *GATE*-Komponente erstellten Modul `HOTTRANE`[4] übergeben. `HOTTRANE` formuliert daraufhin nRQL-Anweisungen zur Erstellung entsprechender Individuen und (associatedWith) Relationen und sendet diese an *RacerPro*. Um anschließend nicht für jedes angelegte Individuum einzeln die eventuell abgeleiteten Konzeptzugehörigkeiten mit einer eigenen Abfrage ermitteln zu müssen, wird der von *RacerPro* angebotene `publish-subscribe`-Mechanismus [HMW05] verwendet: Hierzu wird beim Systemstart für jedes einzelne in der Ontologie enthaltene Konzept eine benannte Abfrage bei *RacerPro* registriert. Da `HOTTRANE` nach jeder Anweisung zur Erstellung eines Individuums mit dem `publish-`

4 Der Name des Moduls `HOTTRANE` steht als Abkürzung für *Terminological Reasoning About Named Entities in Hierarchically Ordered Textelements.*

Befehl dessen Veröffentlichung veranlasst, liefert *RacerPro* veränderte Konzeptzugehörigkeiten als direkte Antwort auf die ihm übermittelten nRQL-Anweisungen.

Zum Abschluss des Klassifikationsvorgangs leitet `HOTClassifier` die Namen der Konzepte, die für die Individuen des Textfragments abgeleitet werden konnten, an *Plone* weiter. Sollten sich Änderungen für die dem aktuell betrachteten Textfragment übergeordneten HOTs ergeben haben, so werden diese ebenfalls an Plone übermittelt. *Plone* trägt daraufhin die übermittelten Konzeptnamen als Schlagwörter für das klassifizierte Textfragment ein. Über die von *Plone* bereitgestellte Suchfunktion lassen sich klassifizierte Textelemente dann durch die Suche nach Konzeptnamen auffinden (siehe Abb. 5.3.3).

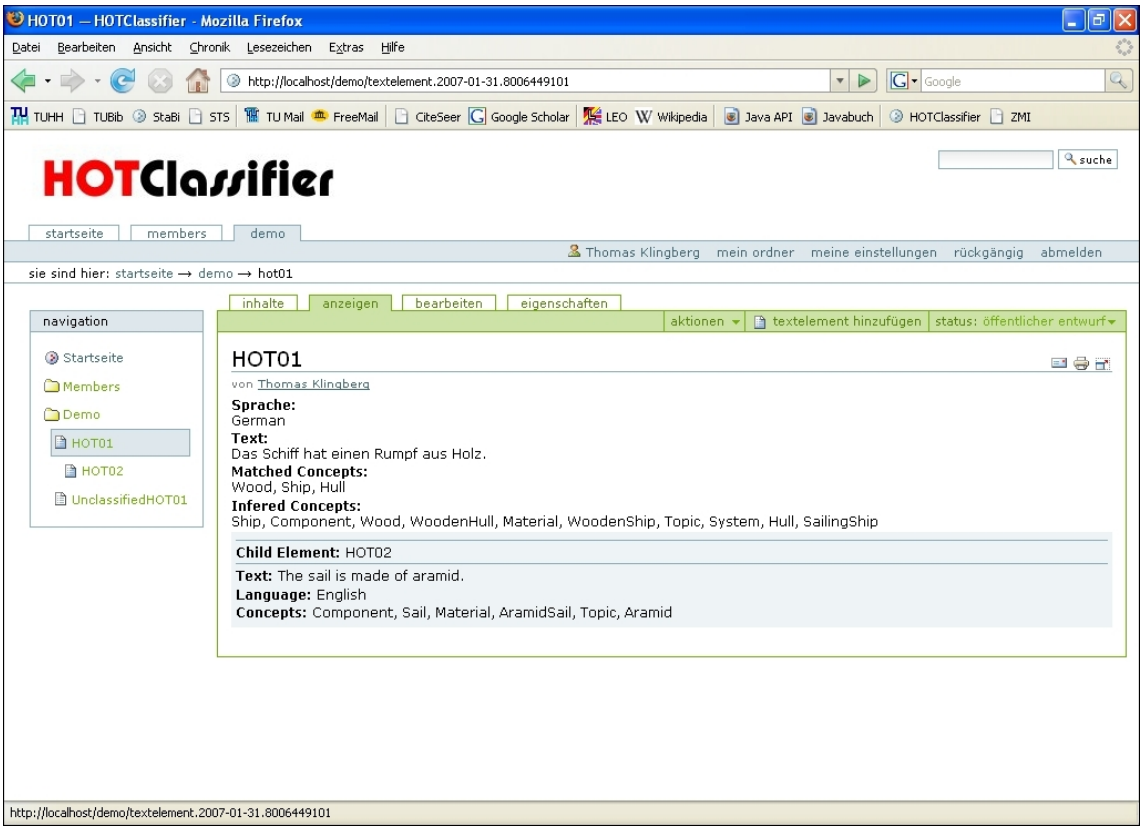

Abb. 5.3.3: Zwei klassifizierte Textfragmente im `HOTClassifier`-Portal

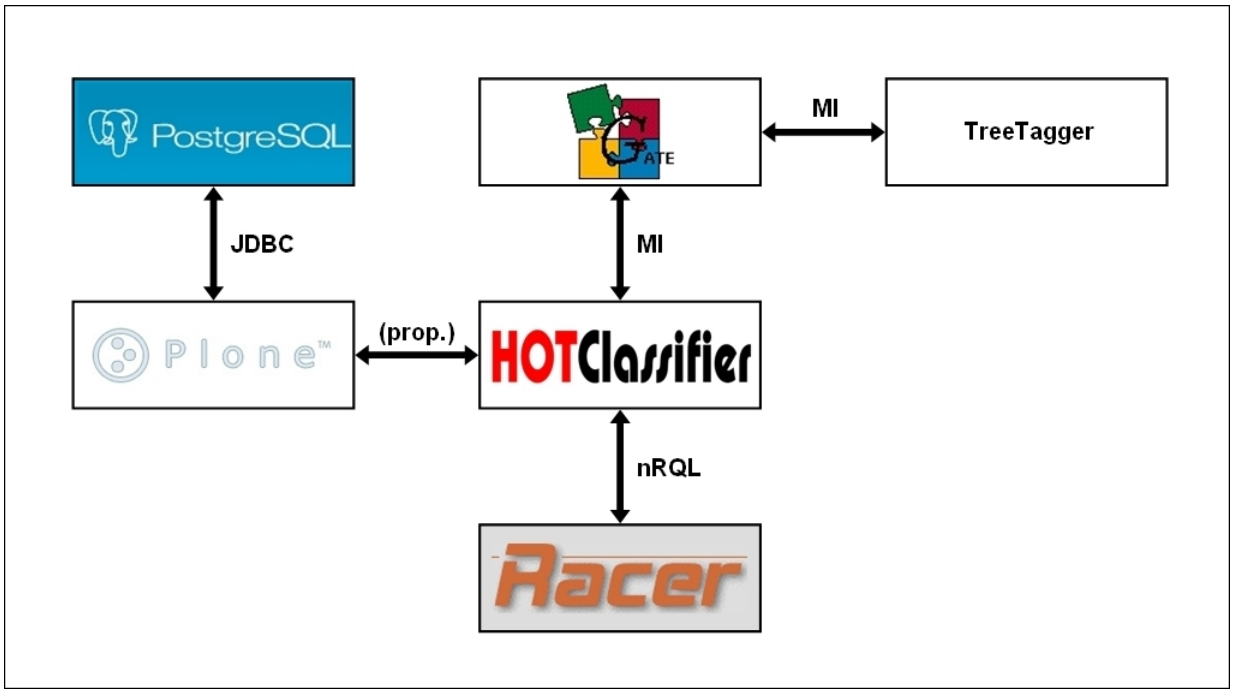

Abb. 5.3.4: Anbindung zwischen den Systemkomponenten

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Mit der vorliegenden Arbeit wurde ein Konzept für ein System zur wissensbasierten Textklassifikation von hierarchisch angeordneten und inhaltlich abhängigen Textfragmenten erstellt. Dieses umfasst die maschinenverständliche und maschineninterpretierbare Konzeptualisierung von Domänenwissen ebenso, wie die Abbildung von Textfragmenten und die Ausnutzung der von der verwendeten Inferenzmaschine *RacerPro* bereitgestellten Retrievalfähigkeiten. Das System ist für die vollautomatische Extraktion von Deskriptoren für Konzepte und Relationen ausgelegt und klassifiziert Textfragmente durch ihre Assoziation mit den ihrem Inhalt entsprechend angelegten Individuen von Themen-Konzepten. Darüber hinaus wurde ein Lösungsansatz für die Spezialisierung von aus Texten extrahierten Relationen durch den Einsatz von abduktivem Schließen entwickelt.

Die praktische Anwendbarkeit des Konzepts wurde durch die Implementation des Prototypen `HOTClassifier` erfolgreich nachgewiesen - wenngleich dieser aufgrund des limitierten Zeitrahmens noch nicht den vollen Umfang des Konzepts umsetzt. Das anvisierte Ziel, einen vertikalen Durchstich durch alle beteiligten Systemkomponenten zu schaffen, wurde jedoch vollständig erfüllt. Durch geeignete Modellierung von Aussagen in Beschreibungslogik und dem Einsatz von deduktivem Schliessen, gelingt es bereits jetzt, die inhaltliche Klassifikation eines Textfragments zu Spezialisieren. Das System wurde zudem für den multilingualen Einsatz ausgelegt und unterstützt zum Zeitpunkt des Abschlusses dieser Diplomarbeit die Klassifikation sowohl englischsprachiger, als auch deutschsprachiger Texte.

6.2 Probleme

Zwei Hemmnisse für den sofortigen Einsatz des Systems lassen sich identifizieren:

Zum einen kann die Verwendung von Verben als Deskriptoren für Relationen die im Text kommunizierten Zusammenhänge nur eingeschränkt extrahieren. In diesem Punkt besteht weiterer Forschungsbedarf, wenngleich in naher Zukunft durch den von *RacerPro* bereitgestellten Abduktions-Service die Möglichkeit einer stärkeren Spezialisierung der erkannten Beziehungen bestehen wird.

Das zweite Hemmnis ist, dass der Aufbau einer geeigneten Ontologie bereits für eine stark eingegrenzte Domäne sehr zeitaufwändig ist. Hier scheint der Untersuchung des Einsatzes semi-automatischer Verfahren sinnvoll zu sein. Der Aufwand für die Erfassung von Domänenwissen ist jedoch gerechtfertigt, zumal dieses auch in anderen Zusammenhängen und Anwendungen wiederverwendet werden kann.

6.3 Ausblick

Mit der Implementation von `HOTClassifier` wurde eine Plattform geschaffen, die sehr gut als Grundlage für weitere Entwicklungen genutzt werden kann.

So ließe sich auf ihrer Basis ein Dienst entwickeln, der natürlichsprachliche Anfragen in Beschreibungslogik abbildet und auf diesem Weg menschlichen Benutzern die in der Wissensbasis vorhandenen Information zugänglich macht. Abfragen ließen sich durch einen solchen Dienst entlang der Konzepthierachie spezialisieren (einschränken) oder generalisieren (ausdehnen).

Ein interessanter Aspekt wäre auch die Untersuchung, wie sich im Text enthaltene numerische Angaben extrahieren und bestimmten Konzepten zuordnen lassen. Damit wäre es möglich zu überprüfen, ob sich die Eigenschaften eines Individuum, wie beispielsweise eines Bauteils eines Technischen Systems, innerhalb eines festgelegten Rahmens befinden.

Neben Textfragmenten ließen sich auch andere Medien als Konzepte abbilden und unter einem generellen Konzept InformationUnit vereinigen. So könnte erreicht werden, dass z.B. Bilder, die in einem Textfragment enthalten sind, als dessen Kindelemente betrachtet werden. Durch geeignete Extraktionsverfahren könnten in diesen enthaltene Information ebenfalls der Wissensbasis hinzugefügt werden und zum Gesamtkontext Beiträge leisten.

Literaturverzeichnis

[ACD+06] Avrithis, Yannis ; Castano, Silvana ; Dalakleidi, Kalliopi [u. a.]: BOEMIE - Methodology and Architecture for Multimedia Ontology Evolution. 2006. – Forschungsbericht. http://www.boemie.org

[AH04] Antoniou, Grigoris ; van Harmelen, Frank: *A Semantic Web Primer*. Cambridge, MA, USA : The MIT Press, 2004

[AR95] Agirre, Eneko ; Rigau, German: A Proposal for Word Sense Disambiguation using Conceptual Distance. In: *Proceedings of the First International Conference on Recent Advances in Natural Language Processing (RANLP)*. Tzigov Chark, Bulgarien, 1995

[BOS03] Buitelaar, Paul ; Olejnik, Daniel ; Sintek, Michael: OntoLT: A Protégé Plug-In for Ontology Extraction from Text. In: *Proceedings of the International Semantic Web Conference (ISWC)*. Sanibel Island, FL, USA, 2003

[BYRN99] Baeza-Yates, Ricardo ; Ribeiro-Neto, Berthier: *Modern Information Retrieval*. New York, NY, USA : ACM Press, 1999

[CMB+02] Cunningham, Hamish ; Maynard, Diana ; Bontcheva, Kalina [u. a.]: GATE: An Architecture for Development of Robust HLT Applications. In: *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL)*. Philadelphia, PA, USA, 2002

[CMB+06] Cunningham, Hamish ; Maynard, Diana ; Bontcheva, Kalina [u. a.]: *Developing Language Processing Components with GATE Version 3 (A User Guide)*. Natural Language Processing Research Group, Department of Computer Science, University of Sheffield, England. 2006. – http://gate.ac.uk/sale/tao/tao.pdf (2006-12-28)

[CT94] Cavnar, William B. ; Trenkle, John M.: N-Gram-Based Text Categorization. In: *Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval (SDAIR)*. Las Vegas, NV, USA, 1994

[DEG+03] Dill, Stephen ; Eiron, Nadav ; Gibson, David [u. a.]: A Case for Automated Large Scale Semantic Annotation. In: *Elsevier's Journal of Web Semantics* 1 (2003), Nr. 1

[DFH03] Davies, John (Hrsg.) ; Fensel, Dieter (Hrsg.) ; van Harmelen, Frank (Hrsg.): *Towards the Semantic Web - Ontology-driven Knowledge Management*. Chichester, England : John Wiley & Sons, 2003

[Dud98] Dudenredaktion (Hrsg.): *Duden - Grammatik der deutschen Gegenwartssprache*. Mannheim : Dudenverlag, 1998

[EKS06] Elsenbroich, Corinna ; Kutz, Oliver ; Sattler, Ulrike: A Case for Abductive Reasoning over Ontologies. In: *Proceedings of OWL: Experiences and Directions Workshop (OWLED)*. Athens, GA, USA, 2006

[Fuh06] Fuhr, Norbert: *Information Retrieval - Skriptum zur Vorlesung*. Fakultät für Ingenieurwissenschaften, Universität Duisburg. 2006. – `http://www.is.informatik.uni-duisburg.de/courses/ir_ss06/folien/irskall.pdf` (2006-12-28)

[GPFLC04] Gómez-Pérez, Asunción ; Fernández-López, Mariano ; Corcho, Oscar: *Ontological Engineering*. London, England : Springer Verlag, 2004

[GSV04] Gabel, Thomas ; Sure, York ; Voelker, Johanna: *KAON - An Overview*. Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB), Universität Karlsruhe. 2004. – `http://kaon.semanticweb.org/main_kaonOverview.pdf` (2006-12-28)

[Hal04a] Halder, Alexander: *Entwurf und Nutzung von Ontologien zur Produktteileverwaltung am Beispiel des Geschäftsfeldes PKW der DaimlerChrysler AG*, Abteilung „Künstliche Intelligenz", Fakultät für Informatik, Universität Ulm, Diplomarbeit, 2004. – `http://www.informatik.uni-ulm.de/ki/Edu/Diplomarbeiten/halder-dipl.pdf` (2006-12-28)

[Hal04b] Hall, Mark: *Semi-Automatische Ontologieerstellung mittels TextToOnto*. Institut für Sprachwissenschaft und Computerlinguistik, Universität Klagenfurt, Österreich, Seminar-Ausarbeitung. 2004. – `http://www.edu.uni-klu.ac.at/~mhall/data//courses/se_cl/se_cl_semi_automatic_texttoonto.pdf` (2006-12-28)

[HKR+04] Horridge, Matthew ; Knublauch, Holger ; Rector, Alan [u. a.]: A Practical Guide to Building OWL Ontologies using the Protégé-OWL Plugin and CO-ODE Tools. In: *Proceedings of the 3rd International Semantic Web Conference (ISWC)*. Hiroshima, Japan, 2004

[HMW05] Haarslev, Volker ; Möller, Ralf ; Wessel, Michael: *RacerPro User's Guide Version 1.9.* Racer Systems GmbH & Co. KG. 2005. – `http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf` (2006-12-28)

[HQW06] Heyer, Gerhard ; Quasthoff, Uwe ; Wittig, Thomas: *Text Mining: Wissensrohstoff Text.* Bochum : W3L, 2006

[HST00] Horrocks, Ian ; Sattler, Ulrike ; Tobies, Stephan: Reasoning with Individuals for the Description Logic SHIQ. In: *Proceedings of the 17th International Conference on Automated Deduction (CADE).* Pittsburgh, PA, USA, 2000

[KL02] Kunze, Claudia ; Lemnitzer, Lothar: GermaNet - Representation, Visualization, Application. In: *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC).* Las Palmas de Gran Canaria, Spanien, 2002

[Koc04] Koch, Stefan: *Knowledge Discovery durch Text Mining*, Fakultät für Informations- und Kommunikationswissenschaft, Fachhochschule Köln, Diplomarbeit, 2004. – `http://stefkoch.de/diplom/knowledge-discovery-durch-text-mining.pdf` (2006-05-12)

[KPO+05] Kiryakov, Atanas ; Popov, Borislav ; Ognyanoff, Damyan [u. a.]: Semantic Annotation, Indexing, and Retrieval. In: *Elsevier's Journal of Web Semantics* 2 (2005), Nr. 1

[KS99] Kiryakov, Atanas K. ; Simov, Kiril I.: Ontologically Supported Semantic Matching. In: *Proceedings of the 12th Nordic Conference of Computational Linguistics (NoDaLiDa).* Trondheim, Norwegen, 1999

[Kun05] Kunz, Christoph D.: *Ein integrierter Ansatz zur wissensbasierten Informationsrecherche*, Fakultät für Maschinenbau, Universität Stuttgart, Diss., 2005. – `http://elib.uni-stuttgart.de/opus/volltexte/2006/2556/pdf/Diss_Kunz_hs.pdf` (2006-12-28)

[MF03] Mayfield, James ; Finin, Tim: Information Retrieval on the Semantic Web: Integrating Inference and Retrieval. In: *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Workshop on the Semantic Web.* Toronto, Kanada, 2003

[MGH04] McGuinness, Deborah L. (Hrsg.) ; van Harmelen, Frank (Hrsg.): *OWL Web Ontology Language Overview*. World Wide Web Consortium (W3C). 2004. – `http://www.w3.org/TR/owl-features/` (2006-12-28)

[MMV02] Motik, Boris ; Maedche, Alexander ; Volz, Raphael: A Conceptual Modeling Approach for Semantics-Driven Enterprise Applications. In: Meersman, Robert (Hrsg.) [u. a.]: *On the Move to Meaningful Internet Systems 2002 - CoopIS, DOA, and ODBASE*. London, England : Springer Verlag, 2002

[MNV02] Missikoff, Michele ; Navigli, Roberto ; Velardi, Paola: The Usable Ontology: An Environment for Building and Assessing a Domain Ontology. In: *Proceedings of the International Semantic Web Conference (ISWC)*. Sardinien, Italien, 2002

[MSS+03] Maedche, Alexander ; Staab, Steffen ; Stojanovic, Nenad [u. a.]: SEmantic portAL - The SEAL Approach. In: Fensel, Dieter (Hrsg.) [u. a.]: *Spinning the Semantic Web*. Cambridge, MA, USA : The MIT Press, 2003

[MSS04] Motik, Boris ; Sattler, Ulrike ; Studer, Rudi: Query Answering for OWL-DL with Rules. In: *Proceedings of the 3rd International Semantic Web Conference (ISWC)*. Hiroshima, Japan, 2004

[NBB+97] Neumann, Günter ; Backofen, Rolf ; Baur, Judith [u. a.]: An Information Extraction Core System for Real World German Text Processing. In: *Proceedings of the 5th Conference on Applied Natural Language Processing*. Washington, D.C., USA, 1997

[NM01] Noy, Natalya F. ; McGuiness, Deborah L.: Ontology Development 101: A Guide to Creating Your First Ontology / Knowledge Systems Laboratory, Stanford University, CA, USA. 2001. – Forschungsbericht. `http://protege.stanford.edu/publications/ontology_development/ontology101.pdf` (2006-12-28)

[NP02] Neumann, Günter ; Piskorski, Jakub: A Shallow Text Processing Core Engine. In: *Computational Intelligence* 18 (2002), Nr. 3

[PKP+02] Petasis, Georgios ; Karkaletsis, Vangelis ; Paliouras, Georgios [u. a.]: Ellogon: A New Text Engineering Platform. In: *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC)*. Las Palmas, Spanien, 2002

[PS006] Prud'hommeaux, Eric (Hrsg.) ; Seaborne, Andy (Hrsg.): *SPARQL Query Language for RDF*. World Wide Web Consortium (W3C). 2006. – `http://www.w3.org/TR/rdf-sparql-query/` (2006-12-28)

[PS04] Parsia, Bijan ; Sirin, Evren: Pellet: An OWL DL Reasoner. In: *Proceedings of the 3rd International Semantic Web Conference (ISWC)*. Hiroshima, Japan, 2004

[RSM94] Richardson, Ray ; Smeaton, Alan F. ; Murphy, John: Using WordNet as a Knowledge Base for Measuring Semantic Similarity between Words. In: *Proceedings of the 7th Annual Irish Conference on Artificial Intelligence and Cognitive Science (AICS)*. Dublin, Irland, 1994

[San91] Santorini, Beatrice: *Part-of-Speech Tagging Guidelines for the Penn Treebank Project*. 1991. – `www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/Penn-Treebank-Tagset.ps` (2006-12-28)

[SS04] Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Berlin : Springer Verlag, 2004

[STS+95] Schiller, Anne ; Teufel, Simone ; Stöckert, Christine [u. a.]: *Vorläufige Guidelines für das Tagging deutscher Textcorpora mit STTS*. Institut für maschinelle Sprachverarbeitung, Universität Stuttgart. 1995. – `http://www.ims.uni-stuttgart.de/ftp/pub/corpora/stts_guide.ps.gz` (2006-12-28)

[TH06] Tsarkov, Dmitry ; Horrocks, Ian: FaCT++ Description Logic Reasoner: System Description. In: *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR)*. Seattle, WA, USA, 2006

[VFC05] Vallet, David ; Fernández, Mariam ; Castells, Pablo: An Ontology-Based Information Retrieval Model. In: *Proceedings of the 2nd European Semantic Web Conference (ESWC)*. Heraklion, Griechenland, 2005

[Yer03] Yergeau, François: UTF-8, a Transformation Format of ISO 10646. In: *RFC 3629* (2003). – `http://www.tzi.de/~cabo/pdfrfc/rfc3629.txt.pdf` (2006-12-28)

A Beispiel für Ontologie und Wissensbasis

Auf den folgenden Seiten ist eine Ontologie in der Beschreibungssprache OWL dargestellt, die dem in dieser Arbeit entwickelten Modellierungsansatz verwendet (siehe Kapitel 5). Neben der Ontologie enthält die OWL-Datei eine minimale Wissensbasis in Form der Assertionen aus Beispiel 5.2.3.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://sts.tu-harburg.de/ship-demo.owl#"
  xml:base="http://sts.tu-harburg.de/ship-demo.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Component">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Topic"/>
    </rdfs:subClassOf>
    <rdfs:label xml:lang="de">Bauteil</rdfs:label>
    <rdfs:label xml:lang="en">component</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="Hydrogen">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Fuel"/>
    </rdfs:subClassOf>
    <rdfs:label xml:lang="en">hydrogen</rdfs:label>
    <rdfs:label xml:lang="de">Wasserstoff</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="Engine">
```

```
  <rdfs:label xml:lang="en">engine</rdfs:label>
  <rdfs:label xml:lang="de">Antrieb</rdfs:label>
  <rdfs:label xml:lang="de">Motor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Component"/>
</owl:Class>
<owl:Class rdf:ID="AramidSail">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:ID="Sail"/>
        <owl:Restriction>
          <owl:someValuesFrom>
            <owl:Class rdf:ID="Aramid"/>
          </owl:someValuesFrom>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="isMadeOf"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:label xml:lang="en">aramid sail</rdfs:label>
  <rdfs:label xml:lang="de">Aramidsegel</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="ElectricMotor">
  <rdfs:subClassOf rdf:resource="#Engine"/>
  <rdfs:label xml:lang="en">electric motor</rdfs:label>
  <rdfs:label xml:lang="de">Elektromotor</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Hull">
  <rdfs:label xml:lang="de">Rumpf</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Component"/>
  <rdfs:label xml:lang="en">hull</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Textfragment"/>
<owl:Class rdf:ID="DieselEngine">
  <rdfs:label xml:lang="en">diesel engine</rdfs:label>
  <rdfs:label xml:lang="de">Dieselmotor</rdfs:label>
  <owl:equivalentClass>
```

```
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Engine"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasFuel"/>
          </owl:onProperty>
          <owl:someValuesFrom>
            <owl:Class rdf:ID="Diesel"/>
          </owl:someValuesFrom>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Fuel">
  <rdfs:label xml:lang="en">fuel</rdfs:label>
  <rdfs:label xml:lang="de">Brennstoff</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Topic"/>
</owl:Class>
<owl:Class rdf:ID="Ship">
  <rdfs:label xml:lang="en">ship</rdfs:label>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="System"/>
  </rdfs:subClassOf>
  <rdfs:label xml:lang="de">Schiff</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="SailingShip">
  <rdfs:label xml:lang="de">Segelschiff</rdfs:label>
  <rdfs:label xml:lang="en">sailing ship</rdfs:label>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Ship"/>
        <owl:Restriction>
          <owl:someValuesFrom>
            <owl:Class rdf:about="#Sail"/>
          </owl:someValuesFrom>
          <owl:onProperty>
```

```
            <owl:ObjectProperty rdf:ID="hasPart"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Sail">
  <rdfs:label xml:lang="de">Segel</rdfs:label>
  <rdfs:label xml:lang="en">sail</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Component"/>
</owl:Class>
<owl:Class rdf:ID="Material">
  <rdfs:subClassOf rdf:resource="#Topic"/>
  <rdfs:label xml:lang="de">Material</rdfs:label>
  <rdfs:label xml:lang="en">material</rdfs:label>
</owl:Class>
<owl:Class rdf:about="#Diesel">
  <rdfs:label xml:lang="en">diesel fuel</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Fuel"/>
  <rdfs:label xml:lang="de">Diesel</rdfs:label>
</owl:Class>
<owl:Class rdf:about="#System">
  <rdfs:label xml:lang="en">system</rdfs:label>
  <rdfs:label xml:lang="de">System</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Topic"/>
</owl:Class>
<owl:Class rdf:ID="Wood">
  <rdfs:label xml:lang="de">Holz</rdfs:label>
  <rdfs:label xml:lang="en">wood</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Material"/>
</owl:Class>
<owl:Class rdf:ID="WoodenHull">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Hull"/>
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#Wood"/>
```

```
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#isMadeOf"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:label xml:lang="de">Holzrumpf</rdfs:label>
  <rdfs:label xml:lang="en">wooden hul</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Motorship">
  <rdfs:subClassOf rdf:resource="#Ship"/>
  <rdfs:label xml:lang="en">motorship</rdfs:label>
  <rdfs:label xml:lang="de">Motorschiff</rdfs:label>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasPart"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Engine"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Aramid">
  <rdfs:subClassOf rdf:resource="#Material"/>
  <rdfs:label xml:lang="de">Aramid</rdfs:label>
  <rdfs:label xml:lang="en">aramid</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="WoodenShip">
  <rdfs:label xml:lang="de">Holzschiff</rdfs:label>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Ship"/>
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#WoodenHull"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasPart"/>
          </owl:onProperty>
```

```
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#Wood"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#isMadeOf"/>
          </owl:onProperty>
        </owl:Restriction>
        <owl:Class rdf:about="#Ship"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:label xml:lang="en">wooden ship</rdfs:label>
</owl:Class>
<owl:ObjectProperty rdf:about="#hasFuel">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="isFuelOf"/>
  </owl:inverseOf>
  <rdfs:subPropertyOf>
    <owl:SymmetricProperty rdf:ID="associatedWith"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isMadeOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="isMaterialOf"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#Material"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Component"/>
        <owl:Class rdf:about="#System"/>
      </owl:unionOf>
    </owl:Class>
```

```
    </rdfs:domain>
    <rdfs:subPropertyOf>
      <owl:SymmetricProperty rdf:about="#associatedWith"/>
    </rdfs:subPropertyOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isContentOf">
    <owl:inverseOf>
      <owl:TransitiveProperty rdf:ID="hasContent"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#Textfragment"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Textfragment"/>
          <owl:Class rdf:about="#Topic"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isTopicOf">
    <rdfs:domain rdf:resource="#Topic"/>
    <rdfs:subPropertyOf rdf:resource="#isContentOf"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="hasTopic"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#Textfragment"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#isMaterialOf">
    <rdfs:domain rdf:resource="#Material"/>
    <owl:inverseOf rdf:resource="#isMadeOf"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Component"/>
          <owl:Class rdf:about="#System"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
```

```
    <rdfs:subPropertyOf>
      <owl:SymmetricProperty rdf:about="#associatedWith"/>
    </rdfs:subPropertyOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isPartOf">
    <rdfs:domain rdf:resource="#Component"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#hasPart"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#System"/>
    <rdfs:subPropertyOf>
      <owl:SymmetricProperty rdf:about="#associatedWith"/>
    </rdfs:subPropertyOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasTopic">
    <owl:inverseOf rdf:resource="#isTopicOf"/>
    <rdfs:range rdf:resource="#Topic"/>
    <rdfs:subPropertyOf>
      <owl:TransitiveProperty rdf:about="#hasContent"/>
    </rdfs:subPropertyOf>
    <rdfs:domain rdf:resource="#Textfragment"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#isFuelOf">
    <owl:inverseOf rdf:resource="#hasFuel"/>
    <rdfs:subPropertyOf>
      <owl:SymmetricProperty rdf:about="#associatedWith"/>
    </rdfs:subPropertyOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasPart">
    <rdfs:range rdf:resource="#Component"/>
    <rdfs:domain rdf:resource="#System"/>
    <owl:inverseOf rdf:resource="#isPartOf"/>
    <rdfs:subPropertyOf>
      <owl:SymmetricProperty rdf:about="#associatedWith"/>
    </rdfs:subPropertyOf>
  </owl:ObjectProperty>
  <owl:TransitiveProperty rdf:about="#hasContent">
    <rdfs:domain rdf:resource="#Textfragment"/>
    <owl:inverseOf rdf:resource="#isContentOf"/>
```

```
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Textfragment"/>
          <owl:Class rdf:about="#Topic"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:TransitiveProperty>
  <owl:SymmetricProperty rdf:about="#associatedWith">
    <owl:inverseOf rdf:resource="#associatedWith"/>
    <rdfs:range rdf:resource="#Topic"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Topic"/>
  </owl:SymmetricProperty>
  <Textfragment rdf:ID="hot2">
    <hasTopic>
      <Sail rdf:ID="hot2_sail">
        <isTopicOf rdf:resource="#hot2"/>
        <isPartOf>
          <Ship rdf:ID="hot1_ship">
            <isTopicOf>
              <Textfragment rdf:ID="hot1">
                <isContentOf rdf:resource="#hot2"/>
                <hasTopic rdf:resource="#hot1_ship"/>
              </Textfragment>
            </isTopicOf>
            <hasPart rdf:resource="#hot2_sail"/>
          </Ship>
        </isPartOf>
      </Sail>
    </hasTopic>
    <hasContent rdf:resource="#hot1"/>
  </Textfragment>
</rdf:RDF>
```

B Eigenschaften des Testsystems

- Prozessor: **Intel Pentium M 1,60 GHz**
- Arbeitsspeicher: **512 MB**

- Betriebssystem: **Microsoft Windows XP inkl. Service Pack 2**
- Java: **Java Development Kit 1.5.0-04**
- NLP Framework: **GATE 3.1**
- Reasoner: **RacerPro 1.9.0**
- CMS: **Plone 2.5 mit Zope 2.9 und Python 2.4.3**
- DBMS: **PostgreSQL 8.1.5-1**

C Inhaltstyp `Textelement`

Nachfolgend wird der für die Verwendung im Content-Management-System *Plone* in der Programmiersprache *Python* verfasste Inhaltstyp `Textelement` dargestellt:

```
#-----------------------------------------------------------------------------
# Class definition of content type 'TextElement'
#-----------------------------------------------------------------------------

# import required classes and defined project properties
from Products.Archetypes.atapi import registerType
from Products.Archetypes.atapi import OrderedBaseFolder, Schema
from Products.Archetypes.atapi import StringField
from Products.Archetypes.atapi import StringWidget, TextAreaWidget
from Products.Archetypes.atapi import PostgreSQLStorage
from Products.CMFCore.utils import getToolByName
from Acquisition import aq_parent
from socket import *

from Products.HOTClassifier.config import PROJECTNAME
from Products.HOTClassifier.config import SERVER_IP_ADDRESS, SERVER_PORT
from Products.HOTClassifier.config import CLIENT_IP_ADDRESS, CLIENT_PORT


# define class
class TextElement(OrderedBaseFolder):

    # set denotation
    archetype_name = "TextElement"
    meta_type = "TextElement"
    portal_type = "TextElement"

    # set icon
    content_icon = "textelement_icon.gif"
```

```
# define allowed subtypes
allowed_content_types = ("TextElement")


# define schema (based on Plone's OrderedBaseFolder schema)
schema = OrderedBaseFolder.schema.copy() + Schema((

    StringField("text",
        required = True,
        searchable = True,
        storage = PostgreSQLStorage(),
        widget = TextAreaWidget(
            label = "Text",
            description = ("Textual content of the TextElement"),
            modes = ("view", "edit")
        )
    ),

    StringField("language",
        required = False,
        searchable = True,
        storage = PostgreSQLStorage(),
        widget = StringWidget(
            label = "Language",
            description = ("Determined language"),
            modes = ("view")
        )
    ),

    StringField("matchedConcepts",
        required = False,
        searchable = False,
        storage = PostgreSQLStorage(),
        widget = StringWidget(
            label = "Matched Concepts",
            description = ("Concepts matched by Gazetteer"),
            modes = ("view")
        )
    ),
```

```
    StringField("inferedConcepts",
        required = False,
        searchable = True,
        storage = PostgreSQLStorage(),
        widget = StringWidget(
            label = "Infered Concepts",
            description = ("Concepts infered by RacerPro"),
            modes = ("view")
        )
    )
))


# perform classification right after instance creation
def at_post_create_script(self):

    # get portal catalog and uid catalog
    pc = getToolByName(self, "portal_catalog")
    uc = getToolByName(self, "uid_catalog")

    # open a TCP connection to the HOTClassifier server
    s = socket(AF_INET, SOCK_STREAM)
    s.bind((CLIENT_IP_ADDRESS, CLIENT_PORT))
    s.connect((SERVER_IP_ADDRESS, SERVER_PORT))

    # pass UID, parentUID and text to the HOTClassifier server
    s.send(self.UID() + "\n")
    s.send(aq_parent(self).UID() + "\n")
    s.send(self.getText() + "\n")

    # set fields according to received answers
    self.getField("language").set(self, self.receiveMessage(s))
    self.getField("matchedConcepts").set(self, self.receiveMessage(s))
    self.getField("inferedConcepts").set(self, self.receiveMessage(s))

    # reindex instance to include new contents
    pc.reindexObject(self)
```

```
        # receive number of other HOT updates
        numUpdates = int(self.receiveMessage(s))

        # perform necessary updates
        if (numUpdates > 0):
            for i in range(numUpdates):

                # receive messages with HOT UID and new infered concepts
                otherUID = self.receiveMessage(s)
                newConcepts = self.receiveMessage(s)

                # add new concepts to old ones and reindex instance
                other = uc(UID=otherUID)[0].getObject()
                oldConcepts = other.getInferedConcepts()
                if (oldConcepts != ""):
                    newConcepts = oldConcepts + ", " + newConcepts
                other.getField("inferedConcepts").set(other, newConcepts)
                pc.reindexObject(other)

        # close the TCP connection
        s.shutdown(SHUT_RDWR)
        s.close()


    # receive a message concluded by a line feed via the given socket
    def receiveMessage(self, socket):
        c = '' # the recently received character
        message = '' # the assembled message
        while (c != "\n"):
            message = message + c
            c = socket.recv(1)
        return message


# register class
registerType(TextElement, PROJECTNAME)
```

D Danksagung

Ich danke Herrn Prof. Dr. Ralf Möller und Herrn Prof. Dr. Joachim W. Schmidt, sowie ihrem gesamten Team vom Institut für Softwaresysteme der Technischen Universität Hamburg-Harburg, für die Ermöglichung dieser Diplomarbeit und die stets freundliche Unterstützung bei deren Durchführung. Insbesondere möchte ich Herrn Prof. Dr. Ralf Möller für die interessanten und inspirierenden Diskussionen danken, in denen er mit konstruktiver Kritik und Anregungen wichtige Impulse für das Gelingen dieser Arbeit gegeben hat.

Meinen Eltern Hans-Dieter und Edeltraut, meiner Schwester Sabine, meinen Großmüttern Helene und Cäzilie und meiner Freundin Graziella danke ich für ihre immerwährende und liebevolle Unterstützung.

E Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum	Unterschrift