Universität Hamburg
FB Informatik
Datenbanken und Informationssysteme

STUDIENARBEIT

Integration von Information Retrieval Funktionalität in eine offene, persistente Programmierumgebung

April 1996

Ulrike Steffens
Rosenstraße 28
21465 Reinbek

Tel.: 040 / 7223856

Betreuer:
Prof. Dr. Joachim W. Schmidt
Gerald Schröder

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Verfahren zur automatisierten Gewinnung von Informationen (*Information Retrieval*) aus umfangreichen, elektronischen Textdatenbanken sind seit mehr als drei Jahrzehnten Gegenstand wissenschaftlicher Untersuchungen. Die Anforderungen dieses Forschungsgebietes unterscheiden sich grundlegend von den Gegebenheiten im Bereich herkömmlicher Datenbanken. Während im Datenbankbereich die Repräsentation der Daten und damit auch die Ergebnisse eindeutig sind, unterliegen sie im Information Retrieval der Unschärfe.
Der Grund für diese Unschärfe ist die Vielfalt der natürlichen Sprache, die den Textdatenbanken zugrundeliegt. Ein und derselbe Sachverhalt kann in unzähligen Facetten ausgedrückt werden. Kommt ein Wort in unterschiedlichen Texten vor, so kann es immer wieder eine andere Bedeutung tragen. Um trotz der semantischen Fülle der natürlichen Sprache im Information Retrieval verwertbare Ergebnisse erzielen zu können, wurde eine Vielzahl an Konzepten entwickelt, um natürlichsprachliche Inhalte formal möglichst unverfälscht darzustellen.
Mittlerweile beeinflussen andere, weiterführende Aspekte die Forschung auf dem Gebiet des Information Retrieval. Texte werden inzwischen fast ausschließlich am Rechner erstellt, so daß sie dem Information Retrieval in viel größerer Zahl zur Verfügung stehen als noch vor einigen Jahren. Hinzu kommt, daß inzwischen globale Weitverkehrsnetze bestehen, die den Austausch von Daten zwischen nahezu beliebigen Orten auf der Welt ermöglichen. Die textuelle Repräsentation ist hierbei nicht mehr die einzige Form, in der Inhalte gespeichert werden. Das Schlagwort "Multimedia" drückt aus, daß andere Datenrepräsentationen, wie beispielsweise Grafik, Video oder Audio, mehr und mehr an Bedeutung gewinnen.
Ein modernes Retrieval System kann somit aus einem Pool verschiedener Daten, die an den verschiedenen Stellen abgelegt sind, schöpfen. Hierdurch bleibt die Forschung auf dem Sektor des Information Retrieval nicht auf bloße linguistische Fragestellungen beschränkt. Um den veränderten Umständen gerecht zu werden, sind verteilte und hochgradig polymorphe Systeme zu realisieren.
Die vorliegende Arbeit stellt einen Schritt in diese Richtung dar. Neben der Nennung bekannter Konzepte des Information Retrieval beschreibt sie die Anbindung des probabilistischen Retrieval Systems INQUERY an die Systementwicklungsumgebung Tycoon. Durch die Eigenschaften Persistenz, Polymorphismus und Interoperabilität, die dem Tycoon System zu eigen sind, werden Grundlagen für eine moderne Art des Information Retrieval geschaffen.

2. Information Retrieval

Als Information Retrieval bezeichnet man einen Prozeß, in dem die Verbindung zwischen einer Person mit Informationsbedarf (*information need*) und einer Sammlung von Texten[1] hergestellt wird (vgl. Abbildung 2.1).

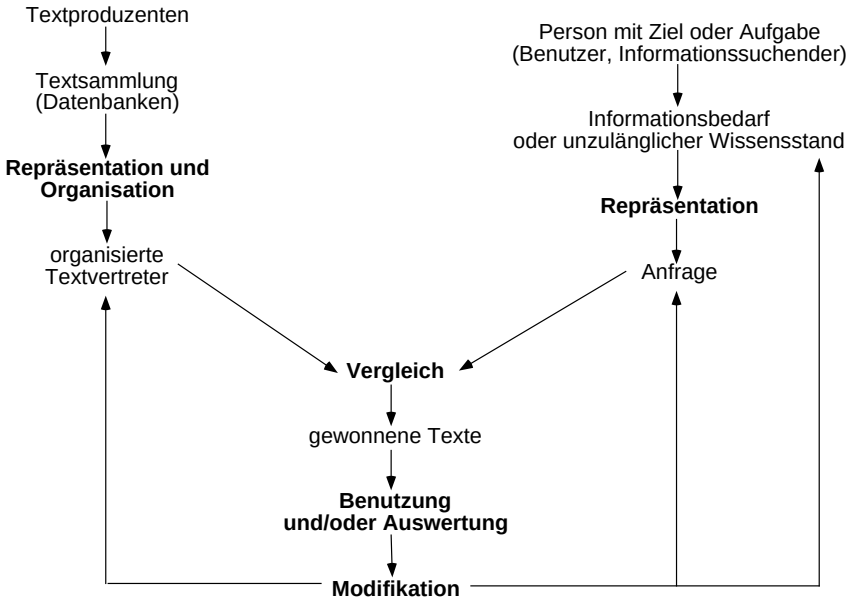

Abbildung 2.1: Ein allgemeines Modell des Information Retrieval [BC92]

Der Informationsbedarf der Person resultiert hierbei aus einem bestimmten, von ihr verfolgten Ziel und ist zeitlich begrenzt. Eine regelmäßige Weiterleitung von Informationen an Personen heißt Informationsfilterung (*information filtering*) [BC92] und ist vom Information Retrieval zu unterscheiden.
Sowohl die Verwaltung von Büchern, Artikeln und ähnlichem in Bibliotheken, in denen die Texte in Papierform vorliegen, als auch das effektive Suchen in diesen Bibliotheken sind mit

[1] Hier und im folgenden wird der Ausdruck Text oder Objekt verwendet. Der Großteil der behandelten Konzepte ist aber durchaus auch für Multimedia-Objekte wie Grafik, Video oder Audio gültig.

Problemen behaftet (vgl. [SM83]). Die Forderung nach ständiger Aktualisierung und einem breiten Angebot beispielsweise führt zu großen Datenmengen. Da in einer Bibliothek nur begrenzt Raum zur Verfügung steht, müssen regelmäßig Werke ausgemustert werden. Dabei ist die Entscheidung, welche Texte für die Benutzer einer Bibliothek keine Relevanz mehr besitzen, nicht trivial. Weiterhin hat die Suche über das Katalogsystem der Bibliothek oft eine große Zahl von Werken zum Ergebnis, die vom ersten Anschein her den Informationsbedarf des Anfragenden decken könnten. Dieser muß zunächst alle Texte sichten, bevor er entscheiden kann, welche davon tatsächlich seinem Zweck dienen. Eine automatisierte Vorabbewertung der Texte, die auf das Problem des Suchenden abgestimmt ist, wäre für die Begrenzung der Ergebnismenge hilfreich. Erfordernisse dieser Art sind in traditionellen Bibliotheken kaum zu erfüllen und machen das Information Retrieval zu einem Einsatzgebiet für Softwaresysteme [2].

Damit Information Retrieval Systeme ihre Aufgabe erfüllen können, müssen für die natürlichsprachlichen Daten sowohl auf der Seite des Informationssuchenden (des Benutzers) als auch auf der Seite der Textsammlung maschinengerechte Repräsentationen[3] gefunden werden. Der Benutzer formuliert seinen Informationsbedarf in einer syntaktischen Form, die das Information Retrieval System verarbeiten kann, der Anfrage (*query*). Die Dokumente der Textsammlung werden durch Strukturen beschrieben, die Textvertreter (*text surrogates*) genannt werden. Ein typischer Textvertreter ist beispielsweise eine Menge von im Text vorkommenden Termen. Den Vorgang, in dem die ursprünglichen Texte durch Vertreter ergänzt werden, bezeichnet man als Indizierung (*indexing*). Die Textvertreter, die sich durch die Indizierung ergeben, werden in Datenbanken organisiert, auf die das Information Retrieval System zugreift.

In der Repräsentation von Texten und Informationsbedarf sowie in deren Abgleich liegen die besonderen Anforderungen des Information Retrieval. Die Abbildung natürlicher Sprache auf vom Rechner interpretierbare Darstellungen läßt sich nicht ohne einen Verlust an Semantik durchführen. Hierbei fallen die folgenden zwei Aspekte besonders ins Gewicht:

- Die Anfragen, die an ein Information Retrieval System gestellt werden, sind *vage Anfragen*. Das heißt, ihre Bedeutung ist a priori nicht eindeutig definiert.

  Die Ursache hierfür ist zum einen, daß der Benutzer nicht immer im Vorwege weiß, nach welchen Begriffen er sucht, und diese daher auch nicht benennen kann. Als Beispiel diene ein Studierender, der nach einem mathematischen Verfahren sucht, um den unbekannten Wert einer Funktion zwischen zwei bekannten Werten zu ermitteln. Wüßte er, daß das gewünschte Verfahren *"interpolation"*[4] genannt wird, so könnte er seine Anfrage präziser formulieren als ohne dieses Wissen.

  Ein anderer Grund für die Vagheit von Anfragen ist, daß der Benutzer häufig nicht

[2] Im weiteren Verlauf der vorliegenden Arbeit bezieht sich der Begriff des Information Retrieval grundsätzlich auf rechnergestützte Verfahren.

[3] Maschinengerechte Repräsentationen werden im Folgenden der Einfachheit halber als Repräsentationen bezeichnet.

[4] In dieser Arbeit werden durchgängig Beispiele in englischer Sprache repräsentiert. Die behandelten Konzepte, Modelle und Anforderungen lassen sich jedoch auf andere Sprachen übertragen.

in der Lage ist, den logischen Kontext der von ihm gesuchten Information genau zu erkennen und darzulegen. Für detaillierte Formulierungen, wie z.B. *"I am looking for political information about World War II, that is especially concerned with the role of Great Britain, that does not contain any military or strategic details and that also considers the pre-war time."*, bedarf es konkreter Vorstellungen über die gesuchten Texte und darüber hinaus auch über die anderen Texte der Textsammlung, die es auszugrenzen gilt.

- Der Vagheit auf der Anfrageseite steht die *Unsicherheit* auf der Textseite gegenüber. Die Semantik eines natürlichsprachlichen Textes läßt sich nur unvollständig auf eine Repräsentation abbilden [Fuh93]. Häufig geht während der Indizierung der Kontext, den ein Dokument für Begriffe darstellt, verloren. Der Begriff *"operating system"* beispielsweise kann in unterschiedlichem Zusammenhang in Texten über Betriebssysteme selbst, über Hardware und auch über Software auftauchen.

Um im Information Retrieval dennoch befriedigende Anfrageergebnisse zu erreichen, wurden zahlreiche allgemeine Modelle entwickelt, die sich mit einer angemessenen Repräsentation natürlicher Sprache und mit der Auswertung solcher Repräsentationen beschäftigen (vgl. Kapitel 3).
Viele dieser Modelle machen von konkreten sprachlichen Konzepten Gebrauch, die das Information Retrieval vereinfachen. Ein solches Konzept ist beispielsweise die Stoppwortelimitierung (*stopping*), bei der Stoppwörter (*stopwords*), die für den Inhalt der Dokumente nicht wesentlich sind, von der Indizierung ausgeschlossen werden. Ein weiteres Konzept ist die Stammformreduktion der Wörter (*stemming*). Hierdurch werden Derivationen[5] und Flexionen[6] unter ihrem Wortstamm zusammengefaßt und ihre einzelne Indizierung unterbunden. Somit wird in der Textsammlung grundsätzlich nach dem Wortstamm gesucht, was die Ergebnismenge vergrößert. Die Veränderung eines exemplarischen Anfragetextes durch Stoppwortelimitierung und Stammformreduktion kann anhand der Abbildung 2.2 nachvollzogen werden.
Durch die Begriffserkennung (*concept recognition*) werden bestimmte Begriffsklassen in den Dokumenten erkannt. Solche Begriffsklassen sind z.B. Firmennamen, Zahlen, Orte, Städte und andere Konzepte, die für den Benutzer von Bedeutung sind. Der Begriffserkenner sucht nach bestimmten Mustern im Text, die auf die von ihm zu erkennende Begriffsklasse schließen lassen. So sucht ein Begriffserkenner für Firmennamen beispielsweise nach Wörtern, die von Bezeichnungen begleitet werden, die auf eine Unternehmungsform hinweisen, wie *"Co"*, *"Inc"* oder *"Ltd"*. Ein Begriffserkenner für Personennamen könnte ein ähnliches Schema verfolgen, indem er nach Wörtern in Verbindung mit Berufs- oder Ehrentiteln sucht. Ebensogut könnte ein solcher Begriffserkenner aber auch auf einer Datenbank mit bekannten Namen arbeiten. Hat ein Begriffserkenner ein bestimmtes Muster identifiziert, so wird an der entsprechenden Stelle im Dokument das Auftauchen der jeweiligen Begriffsklasse vermerkt. Zudem ist ein Begriffserkenner in der Lage, verschiedene Repräsentationen eines

[5] **Derivation**: Bildung neuer Wörter aus einem Ursprungswort
[6] **Flexion**: Deklination oder Konjugation eines Wortes

Document will describe **marketing strategies carried** out by **U.S. companies** for their **agricultural chemicals**, **report predictions** for **market share** of such **chemicals**, or **report market statistics** for the **chemicals**. **pesticide**, **herbicide**, **fungicide**, **insecticide**, **fertilizer**, **predicted sales**, **market share**, **stimulate demand**, **price cut**, **volume** of **sales**	marketing strategy carry usa company agriculture chemical report prediction market share chemical report market statistic chemical pesticide herbicide fungicide insecticide fertilizer predict sale market share stimulate demand price cut volume sale

Abbildung 2.2: Ein Anfragetext vor und nach der Stoppwortelimitierung und der Stammformreduktion [BCC94]

einzelnen Begriffes als einen solchen zu erfassen. So müssen z.B. die Begriffe *"1 million"*, *"1000000"* und *"1.000.000"* als die gleiche Zahl erkannt werden. Die Rolle von Begriffserkennern verdeutlicht die Abbildung 2.3.

John Davenport, 52 years old, was appointed chief executive officer of this international telecommunications concern's **U.S.** subsidiary, **Cable & Wireless North America Inc. Mr. Davenport**, who succeeds **John Zrno**, is currently general manager of the group's operations in **Bermuda**	John Davenport #PERSON, 52 years old, appointed chief executive officer international telecommunications concern U.S. #USA subsidiary, Cable Wireless North America Inc. #COMPANY Mr. Davenport #PERSON, succeeds John Zrno #PERSON, currently general manager group's operations Bermuda #FOREIGNCOUNTRY.

Abbildung 2.3: Ein Dokumententext vor und nach der Begriffserkennung [BCC94]

Besonders bei der Suche nach Eigennamen ist das Konzept der phonetischen Suche hilfreich. Hierbei wird die korrekte Schreibweise eines Wortes vom System außer acht gelassen, indem es auch nach ähnlich klingenden Wörtern sucht. Eine Anfrage nach *Allan Turing* oder *Allen Turing* wäre daher ebenso erfolgreich wie die korrekte Anfrage nach *Alan Turing*.
Zusätzlich zur einfachen Indizierung nehmen einige Information Retrieval Systeme auch eine Feldindizierung vor. Felder sind speziell gekennzeichnete Textabschnitte in einem Dokument. Denkbare Felder wären beispielsweise *"title"*, *"abstract"*, *"text"* oder ähnliches. Die Vorgehensweise bei der Feldindizierung ist identisch zu der der einfachen Indizierung. Die jeweils zu indizierenden Felder werden wie Gesamtdokumente behandelt und die übrigen Felder werden nicht beachtet.

Bei Benutzung der beschriebenen oder weiterer Konzepte ist es wichtig, daß diese sowohl bei der Indizierung der Textsammlung als auch bei der Bearbeitung der Anfragen Anwendung finden. Werden die Anfragen auf andere Weise repräsentiert als die Texte, so besteht die Gefahr, daß nach Begriffen gesucht wird, die in der Textsammlung in dieser Form nicht vorkommen. Der entsprechende Information Retrieval Prozeß wird somit erfolglos abgeschlossen, obwohl sich Dokumente, die Relevanz für die Anfrage besitzen, in der Textsammlung befinden.
Liegen Informationsbedarf und Textsammlung in zueinander passenden, systemadäquaten Repräsentationen vor, so kann der eigentliche Information Retrieval Vorgang stattfinden. Durch eine Auswertung der organisierten Textvertreter hinsichtlich der Anfrage ermittelt das System eine Auswahl an Texten, die den Informationsbedarf des Benutzers decken könnten. Da der tatsächliche Informationswert der gewonnenen Texte aufgrund der semantischen Unsicherheit im Information Retrieval schwankt, kann jedoch nicht davon ausgegangen werden, daß die Ergebnistexte dem Informationsbedarf mit Sicherheit genügen werden.
Möglicherweise beinhalten die gewonnenen Objekte für den Benutzer ausreichendes Wissen. Er wird dann den Information Retrieval Prozeß abschließen und das System verlassen. Die Ergebnistexte können jedoch auch für die Relevanzrückkopplung (*relevance feedback*) verwendet werden. Das bedeutet, daß der Benutzer die Möglichkeit erhält, das Ergebnis des ersten Information Retrieval Vorganges zu bewerten. Das System ist in der Lage, aus dieser Evaluation heraus die Anfrage so zu verändern, daß sie dem eigentlichen Informationsbedarf des Benutzers besser entspricht als zuvor. Abgesehen von dieser anfragebezogenen Nutzung von Relevanzrückkopplungsdaten gibt es weitere Ansätze, wie z.B. die dokumentenbezogene Nutzung, bei der diese Daten für eine Verbesserung der Dokumentenrepräsentationen herangezogen werden [Fuh92].

3. Modelle des Information Retrieval

Modelle des Information Retrieval beschreiben unterschiedliche Vorgehensweisen bei der Repräsentation von Informationsbedarf und Dokumenten sowie bei der Auswertung dieser Repräsentationen.

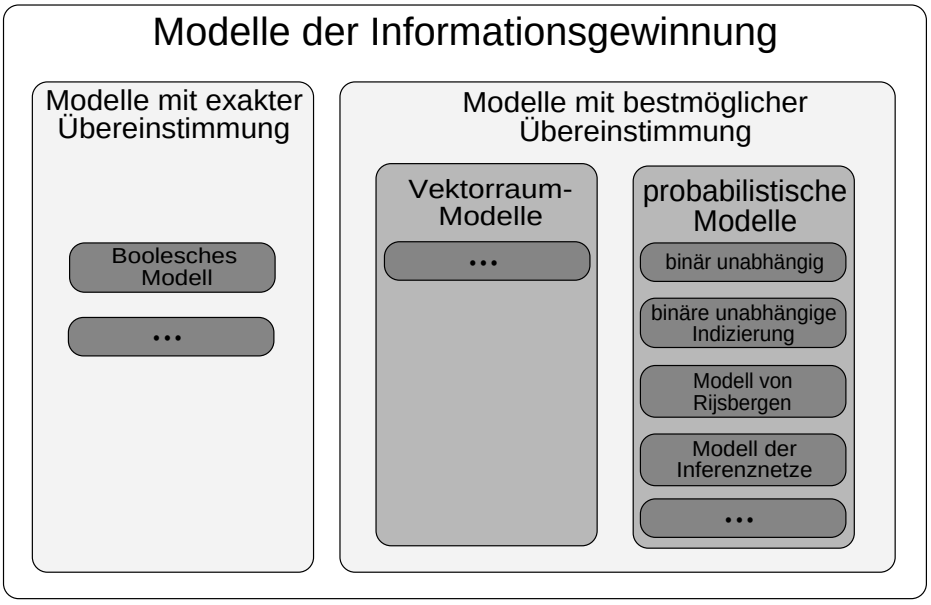

Abbildung 3.1: Eine Übersicht über die Modelle des Information Retrieval

Grob wird dabei nach zwei Klassen von Modellen unterschieden. Die erste Klasse wird von Modellen mit exakter Übereinstimmung (*exact-match models*) gebildet, die zweite von Modellen mit bestmöglicher Übereinstimmung (*best-match models*), die sich wiederum noch genauer unterteilen lassen (vgl. Abbildung 3.1). Während bei Modellen mit exakter Übereinstimmung für jedes Dokument entschieden wird, ob es für die Anfrage relevant ist und damit zur Ergebnismenge gehört, wird bei Modellen mit bestmöglicher Übereinstimmung

für die Dokumente jeweils der Grad angegeben, in dem sie der Anfrage entsprechen.

3.1 Modelle mit exakter Übereinstimmung

Modelle mit exakter Übereinstimmung führen bei der Auswertung der Anfrage Funktionen aus, die die Dokumentensammlung in zwei Teile partitionieren. In der einen Menge befinden sich Dokumente, die für die Anfrage relevant sind, in der anderen solche, die nicht relevant sind. Die Ergebnistexte eines Information Retrieval Prozesses mit exakter Übereinstimmung können somit nicht in eine Rangfolge gebracht werden. Kein Text ist relevanter oder weniger relevant als der andere.[1] Modelle mit exakter Übereinstimmung sind mathematisch einfach und schnell in der Ausführung. Sie bilden deshalb die Basis vieler kommerzieller Information Retrieval Systeme [TC92].
Eines der bekanntesten Modelle mit exakter Übereinstimmung ist das Boolesche Modell, das im folgenden näher beschrieben wird.

Das Boolesche Modell

In vielen Modellen des Information Retrieval ist es möglich, Anfragen mit Hilfe von Booleschen Operatoren zu formulieren. Diese Anfragen können jedoch auf verschiedene Arten verarbeitet werden, wie z.B. mit Inferenznetzen (siehe Abschnitt 3.2.2) oder in einem Vektorraum (siehe Abschnitt 3.2.1). Das Boolesche Modell mit exakter Übereinstimmung ist jedoch nicht auf die Anfrageformulierung bezogen. Es beschreibt vielmehr die Boolesche Interpretation der Anfragen, also die Benutzung Boolescher Logik bei der Auswertung der Textrepräsentationen hinsichtlich der Anfrage.
Als Grundlage des Booleschen Information Retrieval dient eine Menge von binärwertigen Variablen. Sie entsprechen Eigenschaften, die sich Dokumenten zuordnen lassen. Solche Eigenschaften sind im einfachsten Fall Terme, die in Dokumenten zu finden sind. Kompliziertere Eigenschaften sind in Texten enthaltene Phrasen oder Personennamen, die durch ihren Aufbau identifiziert werden können (vgl. Kapitel 2), oder auch manuell zugewiesene Deskriptoren (z.B. "wissenschaftlicher Text").
Für jedes Dokument der Textsammlung ist von vornherein festgelegt, welche der Eigenschaften es erfüllt. Die diesen Eigenschaften entsprechenden Variablen werden mit dem Wert "wahr" versehen, alle anderen Variablen erhalten für das vorliegende Dokument den Wert "falsch".
Die Anfrage an ein System mit Booleschem Information Retrieval ist ein Boolescher Ausdruck, in dem einige Eigenschaftsvariablen durch Boolesche Operatoren verbunden sind.
Bei der Auswertung der Texte hinsichtlich der Anfrage wird jedes Dokument, dessen Eigenschaftsvariablen so belegt sind, daß sie im Booleschen Ausdruck der Anfrage eingesetzt den Wert "wahr" ergeben, in die Ergebnismenge des Information Retrieval Prozesses übernommen. Alle anderen Dokumente sind nicht in der Ergebnismenge enthalten.

[1] Obwohl die Texte nicht nach Relevanz geordnet werden können, ist es dennoch möglich, sie nach anderen Kriterien zu sortieren, wie z.B. zeitlich nach Erscheinungsdaten oder alphabetisch nach Verfassernamen.

3.2 Modelle mit bestmöglicher Übereinstimmung

Bei Modellen mit exakter Übereinstimmung kann für die gewonnenen Dokumente keine Relevanzrangfolge festgelegt werden (vgl. Abschnitt 3.1). Die Anfragen an ein Information Retrieval System sind jedoch vage (vgl. Kapitel 2). Dadurch decken einige Dokumente den Informationsbedarf mit größerer Wahrscheinlichkeit als andere. Gibt ein Information Retrieval System eine Ergebnismenge aus, deren Elemente nach ihrer angenommenen Relevanz geordnet sind, so trägt dieses zur Effektivität und Benutzbarkeit des Systems bei und entspricht eher der intuitiven Auffassung von Information Retrieval. Modelle mit bestmöglicher Übereinstimmung führen daher die Relevanzrangfolge für gewonnene Texte ein. Zwei Gruppen der Modelle mit bestmöglicher Übereinstimmung, die Vektorraum-Modelle (*vector space models*) und die probabilistischen Modelle, werden im folgenden vorgestellt. Sie unterscheiden sich sowohl in der Art der Repräsentation natürlicher Sprache als auch in den Methoden zur Auswertung von Anfragen.

3.2.1 Vektorraum-Modelle

In Vektorraum-Modellen werden Dokumente und Anfragen als Vektoren in einem k-dimensionalen Raum dargestellt. Jede der k Dimensionen entspricht dabei einer möglichen Eigenschaft eines Dokuments (vgl. Abbildung 3.2).

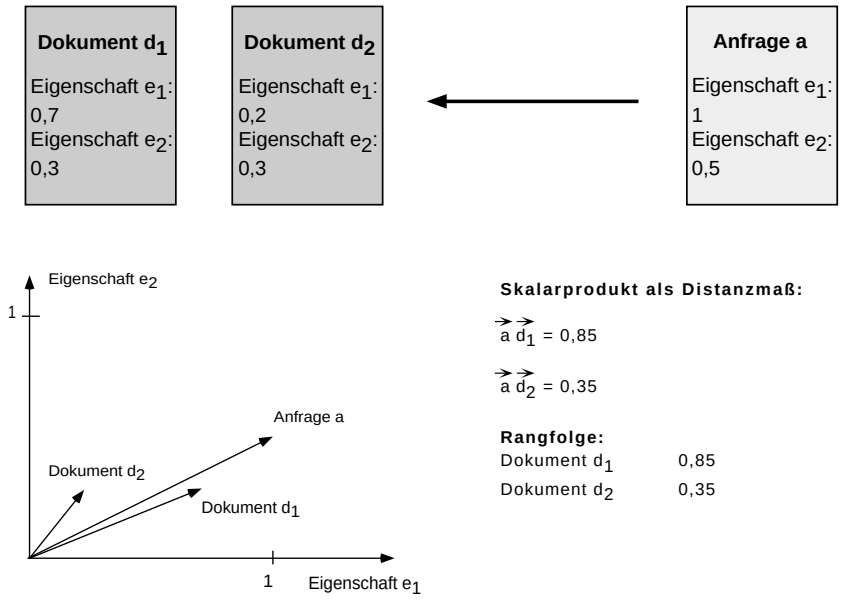

Abbildung 3.2: Das Information Retrieval im Vektorraum

Die Vektorkomponenten können binärwertig sein. Das Dokument besitzt dann entweder

diese Eigenschaften oder nicht. Im allgemeinen sind die Vektorkomponenten jedoch nicht mit Wahrheitswerten belegt sondern mit Gewichtungen in einem festen Intervall, meist im Intervall [0..1]. Hiermit wird ausgedrückt, daß die Eigenschaften zu einem gewissen Grad erfüllt sind. Besitzt eine Vektorkomponente den Wert 0, so erfüllt das zugehörige Dokument bzw. die Anfrage die entsprechende Eigenschaft nicht.
Bei den Eigenschaften handelt es sich häufig um Terme. Die Gewichtung eines Terms in einem Eigenschaftsvektor bezeichnet die Vorkommenshäufigkeit des Terms im Dokument im Vergleich zur Vorkommenshäufigkeit des Terms in der gesamten Dokumentensammlung.
Die Auswertungsfunktion für Dokumente hinsichtlich der Anfrage ist im Vektorraum-Modell ein Distanzmaß, das auf die Dokumenten- und Anfragevektoren angewendet wird. Eine Betrachtung über die Eignung des Cosinus des Winkels zwischen zwei Vektoren und des Skalarprodukts zweier Vektoren als Distanzmaß findet sich in Anhang A.

3.2.2 Probabilistische Modelle

Im Information Retrieval ist weder die Semantik der Anfrage noch die der Textsammlung eindeutig. Es gibt keine vollkommene Sicherheit dafür, daß ein Ergebnis eine Anfrage befriedigt. Modelle mit exakter Übereinstimmung verlagern dieses Problem auf den Benutzer, indem sie für alle gewonnenen Dokumente die Behauptung aufstellen, diese seien relevant. Es liegt beim Benutzer, die Ergebnismenge zu sichten und zu entscheiden, welche der Dokumente seinen Informationsbedarf tatsächlich decken.
Information Retrieval Systeme, die probabilistischen Modellen unterliegen, entlasten den Benutzer, indem sie abschätzen, wie wahrscheinlich es ist, daß ein bestimmtes Dokument d_m für eine Anfrage q_j relevant ist. Ein solches System berechnet also die Wahrscheinlichkeit $P(R|q_j, d_m)$ der Relevanz unter der Voraussetzung, daß die Anfrage q_j und das Dokument d_m betrachtet werden. Diesen Wahrscheinlichkeiten entsprechend stellt das System die gewonnenen Dokumente in eine Rangfolge.
Für die Aufstellung solcher Rangfolgen liefert das Prinzip probabilistischer Rangfolgen (*Probability Ranking Principle, PRP*) die theoretische Grundlage. Es legt fest, wie ein optimales probabilistisches Information Retrieval aussehen soll. Der Begriff des "optimalen Information Retrieval" wird dabei von dem Begriff des "perfekten Information Retrieval" abgegrenzt [Fuh92]. Das perfekte Information Retrieval läge genau dann vor, wenn in der Ergebnisrangfolge alle tatsächlich relevanten Dokumente einen Platz vor den tatsächlich irrevelevanten Dokumenten einnähmen. Das Ergebnis des Retrieval Prozesses wird hier also an den Dokumenten selbst gemessen. Da dieses Ziel jedoch aufgrund der semantischen Unsicherheit im Information Retrieval (vgl. Kapitel 2) nicht erreichbar ist, greift man auf die schwächere Definition des optimalen Information Retrieval zurück. Optimales Information Retrieval liegt dann vor, wenn die gewonnenen Dokumente hinsichtlich der aktuellen Anfrage absteigend nach Relevanzwahrscheinlichkeiten geordnet sind. Hier sind somit nicht mehr die Dokumente selbst, sondern ihre Repräsentationen ausschlaggebend für die Ergebnisrangfolge.
Es gibt eine Vielzahl unterschiedlicher probabilistischer Modelle, die relevanzbezogen oder deduktiv arbeiten.

- Relevanzbezogene probabilistische Modelle des Information Retrieval stellen Hypothesen auf, unter welchen Voraussetzungen ein Dokument für eine Anfrage relevant ist. Aufgrund dieser Annahmen entstehen Rechenmodelle, mit denen die Wahrscheinlichkeit der Relevanz $P(R|q_j, d_m)$ unter der Voraussetzung, daß die Anfrage q_j und das Dokument d_m betrachtet werden, berechnet werden kann. Die Grundlage dieser Rechenmodelle ist die Relevanzrückkopplung (vgl. Kapitel 2), deren Ergebnisse die Parameter zukünftiger Berechnungen darstellen.

  Relevanzbezogene Modelle weisen jedoch Schwachpunkte auf. Sie sind stark von einer Datensammlung abhängig, denn die für sie angenommenen Parameter und damit das Wissen, das einmal in einer Sammlung erworben wurde, sind nicht auf andere Sammlungen übertragbar und müssen somit für jede Sammlung neu ermittelt werden [Fuh92]. Die Hypothesen in relevanzbezogenen Modellen sind simpel und nicht erweiterbar. In den beiden im folgenden vorgestellten Modellen, dem Modell des binären unabhängigen Information Retrieval und dem Modell der binären unabhängigen Indizierung, wird lediglich der Zusammenhang zwischen Termen und Anfragen untersucht. Die wirkliche Relevanz von Objekten für Anfragen hängt jedoch von weiteren Faktoren ab [Fuh92]. So gibt es beispielsweise für viele Terme Synonyme, die ebenfalls das Dokument oder die Anfrage repräsentieren. Hier ist der Einsatz von Thesauri sinnvoll. Einige Terme ändern ihre Bedeutung, wenn sie im Text zusammen mit anderen Termen gruppiert sind. Eine Vielzahl an weiteren sprachlichen Konzepten ist denkbar und kann in den Hypothesen der relevanzbezogenen Modelle nicht oder nur partiell berücksichtigt werden, da die einzubindenden Regeln die Grenzen des Rechenmodells überschreiten würden.

- Die deduktiven probabilistischen Modelle des Information Retrieval legen lediglich fest, auf welche Art während des Retrievalprozesses aus vorliegenden sprachlichen Konzepten Rückschlüsse auf die Relevanzwahrscheinlichkeit der Dokumente gezogen werden können. Ob es sich bei diesen Konzepten um einfache Vorkommenshäufigkeit von Termen, um den Einsatz von Thesauri oder andere linguistische Überlegungen handelt, ist hierbei nicht wichtig. Ein Information Retrieval System, das einem deduktiven probabilistischen Modell folgt, kann aufgrund beliebig beschaffener und beliebig vieler Konzepte Rückschlüsse ziehen. Diese Modelle sind somit unabhängig und erweiterbar. Das Modell von Rijsbergen und das Modell der Inferenznetze, die in diesem Kapitel vorgestellt werden, arbeiten deduktiv.

3.2.2.1 Das Modell des binären unabhängigen Information Retrieval

Den Dokumentenrepräsentationen im Modell des binären unabhängigen Information Retrieval (*binary independence retrieval model, BIR*) liegt die Menge $T = t_1, ..., t_n$ der in der Sammlung vorhandenen Terme zugrunde. Sei d_m^T die Menge der Terme, die in einem Dokument erscheinen, so läßt sich das Dokument als ein Vektor $\chi = (\chi_1, ..., \chi_n)$, mit $\chi_i = 1$ g.d.w. $t_i \in d_m^T$ und $\chi_i = 0$ sonst, darstellen. Das BIR-Modell folgt der Ballungshypothese (*cluster hypothesis*), die besagt, daß Terme in relevanten Dokumenten anders verteilt sind

als in irrelevanten Dokumenten. Nach dieser Hypothese müssen nur noch Dokumente mit unterschiedlichem Vektor χ betrachtet werden, da alle Dokumente, die der jeweilige Vektor beschreibt, entweder relevant oder irrelevant sind. Die Relevanz wird somit für Gruppen von Dokumenten ermittelt. Hier wird nicht die Wahrscheinlichkeit $P(R|q_j, d_m)$ der Relevanz unter der Annahme, daß die Anfrage q_j und das Dokument d_m betrachtet werden, berechnet. Stattdessen wird die Wahrscheinlichkeit $P(R|q_j, \chi)$ der Relevanz unter der Annahme, daß die Anfrage q_j und die Dokumente, die durch den Vektor χ repräsentiert sind, betrachtet werden, kalkuliert. Dabei wird die Anfrage $q_j \subset T$ ebenfalls als eine Menge von Termen q_j^T repräsentiert. Mit Hilfe von Chancen (*odds*) $O(R|q_j, \chi)$, wobei gilt $O(y) = P(y)/P(\bar{y})$, und unter Anwendung von Bayes' Theorem $P(a|b) = P(b|a)P(a)/P(b)$ werden die Relevanzwahrscheinlichkeiten $P(R|q_j, \chi)$ ermittelt. Dabei wird mit Relevanzbeurteilungen für die entsprechenden Dokumente gearbeitet, die aus Relevanzrückkopplungen zurückliegender Information Retrieval Prozesse hervorgegangen sind. Eine genaue Herleitung für $P(R|q_j, \chi)$ sowie ein Beispiel für den Einsatz der Ergebnisse aus den Relevanzrückkopplungen findet sich in [Fuh92].

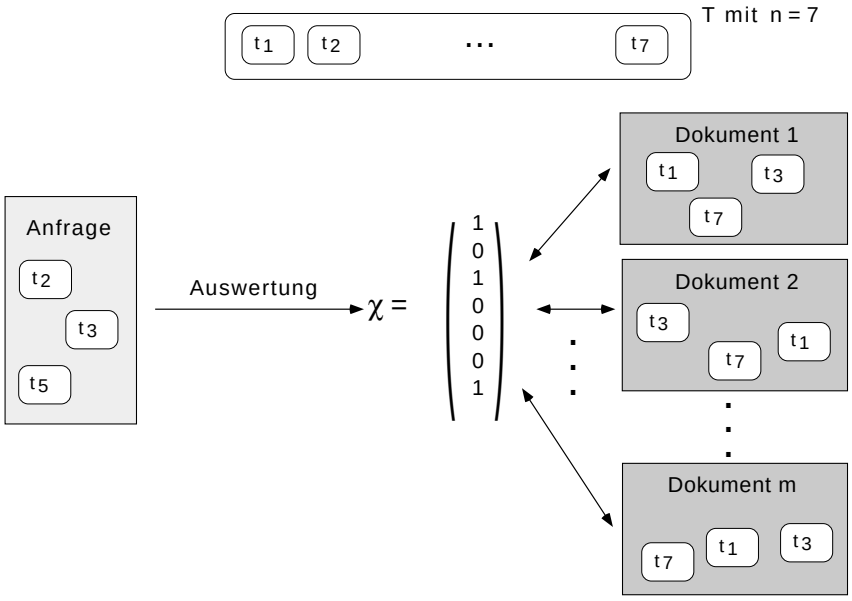

Abbildung 3.3: Das Modell des binären unabhängigen Information Retrieval

3.2.2.2 Das Modell der binären unabhängigen Indizierung

Das BIR-Modell betrachtet eine einzelne Anfrage bezüglich einer Menge von Dokumenten. Das Modell der binären unabhängigen Indizierung (*binary independence indexing model, BII*) hingegen betrachtet ein einzelnes Dokument hinsichtlich einer Menge von Anfragen

(vgl. Abbildung 3.4). Wie im BIR-Modell wird vorausgesetzt, daß die Menge $T = t_1, ..., t_n$ die Menge der in der Textsammlung erscheinenden Terme bezeichnet. Eine Anfrage q_j ist im BII-Modell als eine Menge von Termen $q_j^T \subset T$ repräsentiert. Daraus folgt, daß zwei unterschiedliche Anfragen, die mit derselben Menge von Termen formuliert sind, eine identische Ergebnismenge mit derselben Dokumentenrangfolge erzielen. Daher ist im BII-Modell der binäre Vektor $\varphi = (\varphi_1, ..., \varphi_n)$, mit $\varphi_i = 1$ g.d.w. $t_i \in q_j^T$ und $\varphi_i = 0$ sonst, eine Repräsentation für alle Anfragen q_j, die aus den Termen der Menge q_j^T bestehen. Für diese Vektoren läßt sich die Wahrscheinlichkeit $P(R|\varphi, d_m)$ der Relevanz unter der Voraussetzung, daß die Anfragen, die durch den Vektor φ repräsentiert sind, und das Dokument d_m betrachtet werden, berechnen. Diese entspricht durch die gegebenen Bedingungen der Wahrscheinlicht $P(R|q_j, d_m)$ der Relevanz unter der Voraussetzung, daß die Anfrage q_j und das Dokument d_m betrachtet werden. Bei der Berechnung werden Relevanzurteile für spezielle Dokumenten-Term-Paare aus Relevanzrückkopplungen zurückliegender Information Retrieval Prozesse herangezogen. Die Ermittlung von $P(R|\varphi, d_m)$ wird in [Fuh92] genauer beschrieben. Der Vorteil des BII-Modells ist, daß die Repräsentation der Dokumente frei gewählt werden kann. Dennoch ist dieses Modell in der Praxis kaum anwendbar, denn die erforderliche Relevanzinformation über Dokumenten-Term-Paare ist zu speziell, als daß sie durch Relevanzrückkopplung in ausreichendem Maße zur Verfügung stünde.

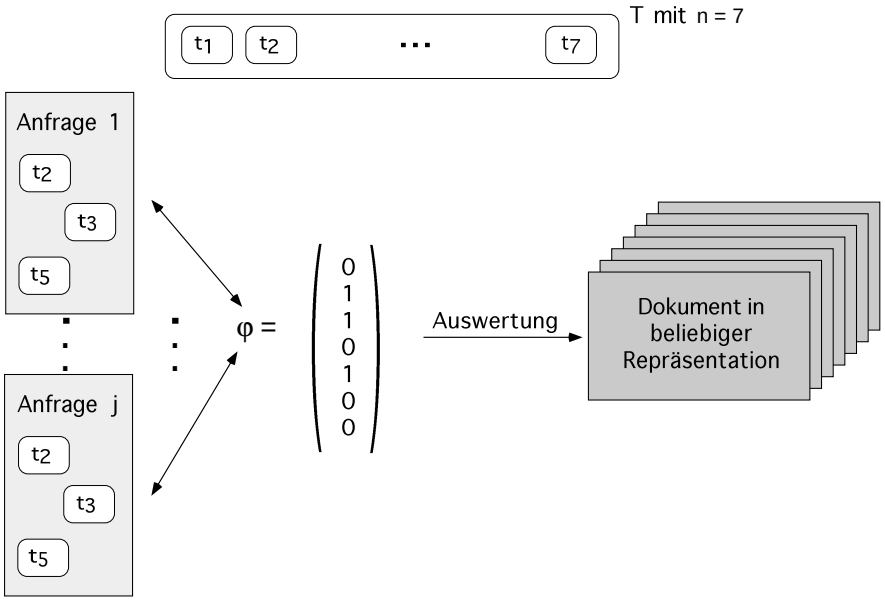

Abbildung 3.4: Das Modell der binären unabhängigen Indizierung

3.2.2.3 Das Modell von Rijsbergen

Das Modell von Rijsbergen verallgemeinert das Prinzip deduktiver Datenbanken im Hinblick auf Information Retrieval. In deduktiven Datenbanken werden sowohl die Daten als auch die Anfragen durch logische Formeln repräsentiert. Die Auswertung einer Anfrage ist der Versuch, die logische Aussage, die diese Anfrage darstellt, durch die logischen Formeln der Datenbank zu beweisen. Auf das Information Retrieval bezogen bedeutet dieses, daß ein Dokument d_m ein Ergebnis für eine Anfrage q_j ist, wenn q_j durch d_m bewiesen werden kann. Es ist zu zeigen, daß $q_j \leftarrow d_m$ wahr ist. Dabei ist die Benutzung zusätzlicher Formeln gestattet. Sei z.B. d_1 ein Text über Quadrate und q_1 eine Anfrage für Texte über Rechtecke, so kann mit Hilfe der Formel $Quadrate \rightarrow Rechtecke$ ausgedrückt werden, daß d_1 eine Relevanz für q_1 besitzt [Fuh92].
Der deduktive Ansatz allein reicht jedoch für das Information Retrieval nicht aus. In herkömmlichen deduktiven Datenbanken sind alle Formeln wahr. In Textdatenbanken hingegen ist es möglich, daß sich die Formeln, die die Dokumente repräsentieren, widersprechen. Rijsbergen löst dieses Problem, indem er das deduktive System weiter aufschlüsselt. Für jedes Dokument wird eine mögliche "Welt" W eingeführt. Diese entspricht einer Menge von Annahmen, die wahr sind. Demgemäß wird eine erweiterte Wahrheitsfunktion τ mit folgender Bedeutung definiert: $\tau(W, x) = 1$, falls x wahr ist in W, und $\tau(W, x) = 0$, falls x falsch ist in W. Durch diese erweiterte Logik werden Konflikte innerhalb der Dokumentensammlung vermieden.
Ein weiteres Problem ist, daß eine Logik, wie van Rijsbergen sie bis hierhin definiert, die Unsicherheit von Vorbedingungen außer acht läßt. Der Wahrheitswert von Formeln der Form $x \rightarrow y$ ist im Information Retrieval nicht immer bestimmbar. Um die Unsicherheit in das logische System von Rijsbergen zu integrieren, wird die Wahrscheinlichkeit $P(x \rightarrow y)$ geschätzt. Wie diese Schätzung durchgeführt wird, wird in [Fuh92] beschrieben. Im Modell von Rijsbergen wird der Begriff der Relevanz nicht betrachtet. Aus dem Ergebnis von unbestimmten Inferenzprozessen einen Wert für die Relevanzwahrscheinlichkeit zu ermitteln, ist bisher nicht möglich. Man kann jedoch anhand der Werte von $P(q_j \leftarrow d_m)$ eine Dokumentenrangfolge bezüglich einer bestimmten Anfrage festlegen [Fuh92].

3.2.2.4 Das Modell der Inferenznetze

Grundlage für das deduktive Modell der Inferenznetze sind Bayes Netze. Bayes Netze sind gerichtete, azyklische Graphen, deren Knoten Annahmen mit konstanter oder variabler Wahrscheinlichkeit darstellen und deren Kanten Abhängigkeiten zwischen den Annahmen aufzeigen [Cha91]. Bewirkt oder beinhaltet eine Annahme p eine weitere Annahme q, so drückt sich dieses im Netz durch eine gerichtete Kante von p nach q aus. Besitzt ein Knoten n Väter, so gibt es für diese, da sie Annahmen darstellen, 2^n mögliche Belegungen, denn jede einzelne Annahme kann entweder wahr oder falsch sein. Für jede dieser Belegungen muß im Sohnknoten bekannt sein, mit welcher Wahrscheinlichkeit die Annahme des Sohnes unter den gegebenen Voraussetzungen gilt oder nicht gilt. Diese Wahrscheinlichkeiten $P(q|p_1, ..., p_n)$ und $P(\bar{q}|p_1, ..., p_n)$ befinden sich für jeden Knoten in einer dazugehörigen 2×2^n-Matrix, der

Verbindungsmatrix. Liegen an den Wurzeln des Netzes anfängliche Wahrheitswerte vor, so kann entlang der Kanten für alle übrigen Knoten die Wahrscheinlichkeit, daß die durch sie dargestellte Annahme zutrifft, ermittelt werden [BC92].

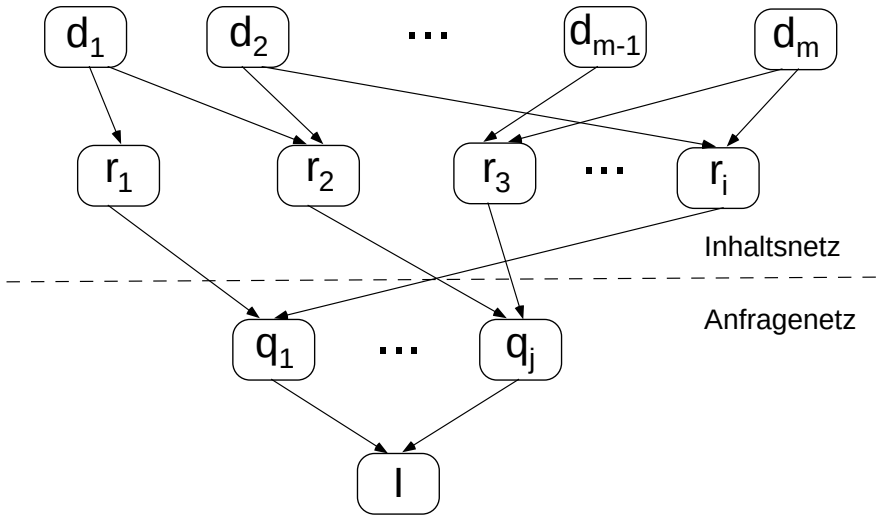

Abbildung 3.5: Ein Inferenznetz des Information Retrieval

Ein Inferenznetz des Information Retrieval (vgl. Abb. 3.5) ist in ein Inhaltsnetz (*document network*) und ein Anfragenetz (*query network*) unterteilt.
Das Inhaltsnetz wird einmal für die Datensammlung generiert und nicht verändert, sofern die Datensammlung gleich bleibt. Es besteht aus den Dokumentenknoten d_m und aus den Konzeptknoten r_i, die verschiedene Repräsentationskonzepte für Dokumente darstellen. Konzeptknoten entstehen somit während der Indizierung. Wird ein Dokument d_m durch ein Konzept r_i repräsentiert, so existiert im Inhaltsnetz eine gerichtete Kante vom Knoten d_m zum Knoten r_i. Dem Konzeptknoten wird eine Wahrscheinlichkeit $P(r_i|d_m)$ für alle seine Elternknoten d_m zugewiesen. Je nachdem welches Konzept ein Konzeptknoten darstellt, kann dieser zu einem weiteren Inferenznetz verfeinert werden.
Das Anfragenetz enthält einen einzelnen Knoten I entsprechend dem Ereignis, daß der Informationsbedarf eines Benutzers gedeckt werden kann. Auf diesen Knoten deuten gerichtete Kanten von Knoten, die Anfragekonzepten q_j entsprechen. Diese Anfragekonzepte repräsentieren den Informationsbedarf in Anfragen unterschiedlicher Form, wie z.B. als Booleschen Ausdruck. Die Verfeinerung von Knoten, die Anfragekonzepte repräsentieren, ist möglich.
Beim eigentlichen Prozeß des Information Retrieval in einem Inferenznetz wird das Anfragenetz, im Gegensatz zum Inhaltsnetz, in Interaktion mit dem Benutzer für jede Anfrage neu erstellt. Beide Netze werden miteinander verbunden. Man betrachtet nun nacheinander alle Dokumentenknoten und ermittelt über die dazwischenliegenden Konzeptknoten die Wahrscheinlichkeit $P(I|d_m)$. Somit kann eine Dokumentenrangfolge bezüglich des vorliegenden

Informationsbedarfs aufgestellt werden.
Das folgende Beispiel dient der Veranschaulichung eines Retrieval Prozesses im Inferenzmodell. Ein Benutzer stellt die Anfrage *"information AND retrieval AND NOT(filtering)"*. Sein Informationsbedarf läßt sich durch ein Anfragenetz, dessen Konzeptknoten aufgrund des Booleschen Charakters der Anfrage verfeinert wurden, darstellen (vgl. Abbildung 3.6).

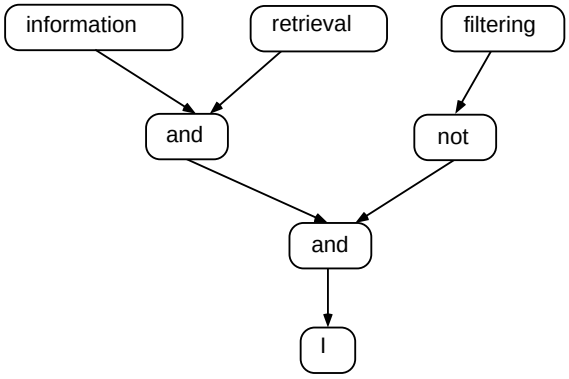

Abbildung 3.6: Ein Anfragenetz für die Anfrage *"information AND retrieval AND NOT(filtering)"*

In der Textsammlung sind unter anderem die Dokumente d_1 und d_2 enthalten. Sie sind in diesem Beispiel einfach durch die Terme repräsentiert, die in ihnen vorkommen. So beinhaltet d_1 die Terme *"information"*, *"retrieval"* und *"filtering"* und d_2 den Term *"retrieval"*. Der betrachtete Teil des Inhaltsnetzes der Textsammlung ist in Abbildung 3.7 dargestellt.

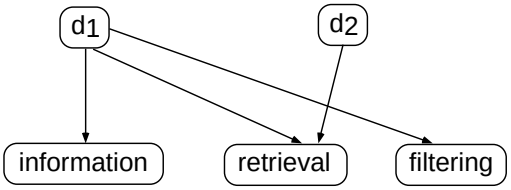

Abbildung 3.7: Ein Inhaltsnetz mit Dokumenten, die die Terme *"information"*, *"retrieval"* und *"filtering"* enthalten

Verbindet man das Anfrage- und das Inhaltsnetz über ihre jeweiligen identischen Konzeptknoten, so erhält man das Inferenznetz in Abbildung 3.8.

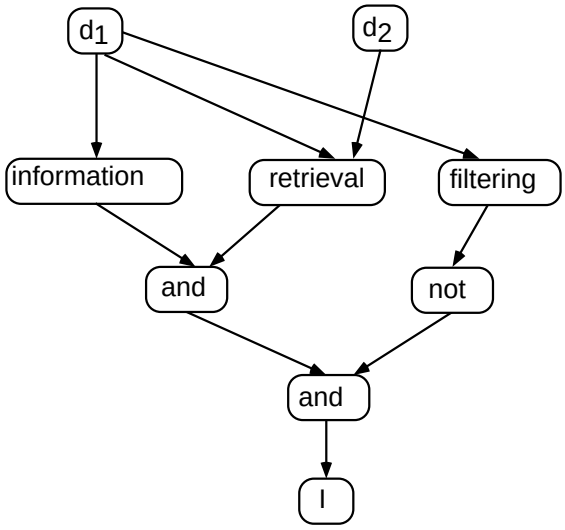

Abbildung 3.8: Ein Inferenznetz für die Anfrage *"information AND retrieval AND NOT(filtering)"*

Mit Hilfe dieses Inferenznetzes läßt sich jedes einzelne Dokument hinsichtlich der Anfrage auswerten, indem man jeweils einem Dokumentenknoten den Wert 1 zuweist und allen anderen den Wert 0. Für das Beispiel gilt also entweder der Fall

$$d_1 = 1 \wedge d_2 = 0$$

oder der Fall

$$d_1 = 0 \wedge d_2 = 1$$

Für beide Fälle wird über das Inferenznetz die Wahrscheinlichkeit $P(I|d_m)$ abgeleitet, daß der vorliegende Informationsbedarf von dem jeweiligen Dokument erfüllt wird.
Die Wahrscheinlichkeiten an den Knoten des Netzes sind dabei von unterschiedlichen Faktoren bestimmt. Die Wahrscheinlichkeit $P(r_i|d_m)$, daß ein Konzeptknoten des Inhaltsnetzes wahr ist, wenn ein mit ihm verbundener Dokumentenknoten betrachtet wird, läßt sich beispielsweise über die relative Vorkommenshäufigkeit des Konzeptes im Dokument bestimmten.

$$P(r_i|d_m) = \frac{Anzahl\ von\ r_i\ in\ d_m}{Anzahl\ von\ r_i\ in\ der\ Textsammlung}$$

Für dieses Beispiel seien die folgenden Vorkommenshäufigkeiten angenommen:

kommt vor in	information	retrieval	filtering
d_1	18	9	4
d_2	0	21	0
gesamt	427	182	79

Daraus lassen sich für die jeweiligen Wahrscheinlichkeiten an den Konzeptknoten des Inhaltsnetzes folgende Werte ableiten:

$$P(information \mid d_1) = \frac{18}{427} \approx 0,0422$$

$$P(retrieval \mid d_1) = \frac{9}{182} \approx 0,0495$$

$$P(filtering \mid d_1) = \frac{4}{79} \approx 0,0506$$

$$P(information \mid d_2) = \frac{0}{127} = 0$$

$$P(retrieval \mid d_2) = \frac{21}{182} \approx 0,1154$$

$$P(filtering \mid d_2) = \frac{0}{79} = 0$$

Die Wahrscheinlichkeiten für die restlichen Knoten des Netzes entsprechen ihrer Booleschen Semantik, die durch folgende Formeln realisiert ist:

$$P(not \mid n_1) = 1 - p_1$$

$$P(and \mid n_1, n_2, \ldots, n_n) = p_1 \cdot p_2 \cdot \ldots \cdot p_n$$

Die Wahrscheinlichkeit eines Knotens, der das Boolesche *"not"* ausdrückt, unter der Voraussetzung, daß der Vaterknoten n_1 betrachtet wird ist also die Differenz aus 1 und der Wahrscheinlichkeit p_1 des Vaterknotens. Die Wahrscheinlichkeit eines Knotens, der das Boolesche *"and"* ausdrückt, ist das Produkt der Wahrscheinlichkeiten p_i aller seiner Vaterknoten n_i. In Kenntnis dieser Formeln kann über die Knoten des Inferenznetzes die Wahrscheinlichkeit, daß das Dokument d_m den Informationsbedarf I deckt, für die beiden Dokumente d_1 und d_2 errechnet werden:

$$P(I|d_1) = (P(information \mid d_1) \cdot P(retrieval \mid d_1)) \cdot (1 - P(filtering \mid d_1)) =$$

$$\frac{18 \cdot 9}{427 \cdot 182} \cdot (1 - \frac{4}{79}) \approx 0,0020$$

$$P(I|d_2) = (P(information \mid d_2) \cdot P(retrieval \mid d_2)) \cdot (1 - P(filtering \mid d_2)) =$$

$$\frac{0 \cdot 9}{127 \cdot 182} \cdot (1 - 0) = 0$$

Somit wird der Informationsbedarf mit größerer Wahrscheinlichkeit vom Dokument d_1 gedeckt als vom Dokument d_2. Dokument d_1 würde daher in der Ergebnisrangfolge für die Anfrage *"information AND retrieval AND NOT(filtering)"* vor Dokument d_2 stehen
Das Modell der Inferenznetze erlaubt die Integration unterschiedlicher Konzepte. So lassen sich andere Modelle des Information Retrieval, wie z.B. das Vektorraum-Modell (vgl. Abschnitt 3.2.1) oder Modelle mit exakter Übereinstimmung (vgl. Abschnitt 3.1) in das Modell der Inferenznetze überführen [TC92]. Der Netzansatz gestattet es, verschiedene Wege der Textrepräsentation, wie z.B. über Thesauri oder Vorkommenshäufigkeit von Termen, in einem Modell zu kombinieren. Eine Verwendung unterschiedlicher Anfrageformulierungen, wie zum Beispiel natürlichsprachlich oder mit Hilfe strukturierter Anfragesprachen, für ein und denselben Informationsbedarf ist möglich.

4. Bewertung von Information Retrieval Systemen

Es gibt eine Vielzahl von Modellen des Information Retrieval (vgl. Abschnitt 3). Somit werden adäquate Kriterien benötigt, um die verschiedenen Systeme, die diese Modelle implementieren, vergleichen zu können. Zusätzlich ermöglichen derartige Bewertungskriterien, für ein einzelnes System abzuschätzen, unter welchen Bedingungen es der Anfrage am besten entsprechen kann.
Der Information Retrieval Prozeß beinhaltet eine Reihe von Schwierigkeiten. Diese Schwierigkeiten sind durch äußere Faktoren wie mehrdeutige Semantik natürlicher Sprache und Unterschiede in den Anforderungen verschiedener Benutzer bedingt, wobei die Abhängigkeiten nicht immer durchschaubar sind (vgl. Abschnitt 2). Daher reichen hier herkömmliche Bewertungsmethoden für Softwaresysteme allein nicht aus. Zusätzliche Bewertungskriterien, die auf die spezielle Situation des Information Retrieval abgestimmt sind, sind notwendig. Aus diesem Grunde unterscheidet man im Information Retrieval zwischen Systemtests, die sich auf die Leistung (*efficiency*) beziehen, und solchen, die mit der Wirksamkeit (*effectiveness*) eines Systems befaßt sind. Hierbei fallen gebräuchliche Bewertungskriterien wie Antwortzeiten und Kosten in die Kategorie Leistung. Die Bewertungskriterien für die Wirksamkeit eines Systems sollen im folgenden eingehender betrachtet werden. Dabei werden zunächst die Maße des Rücklaufs (*recall*) und der Genauigkeit (*precision*) beschrieben. Anschließend wird die Nützlichkeit (*usefulness*) erläutert.

4.1 Rücklauf und Genauigkeit

Als Rücklauf eines Information Retrieval Systems bezeichnet man den Anteil des relevanten gewonnenen Materials am gesamten relevanten Material der Datensammlung:

$$Rücklauf = \frac{Anzahl\ der\ relevanten\ gewonnenen\ Objekte}{Gesamtanzahl\ der\ relevanten\ Objekte\ in\ der\ Datensammlung}$$

Als Genauigkeit eines Information Retrieval Systems bezeichnet man den Anteil des relevanten gewonnenen Materials am gesamten gewonnenen Material:

$$Genauigkeit = \frac{Anzahl\ der\ relevanten\ gewonnenen\ Objekte}{Gesamtanzahl\ der\ gewonnenen\ Objekte}$$

Anders ausgedrückt bezeichnet der Rücklauf die Fähigkeit eines Systems, beim Information Retrieval nützliche Objekte zu finden, während die Genauigkeit die Fähigkeit beschreibt, nutzloses Material auszuschließen.
Die jeweiligen Werte von Rücklauf und Genauigkeit können nicht unabhängig voneinander betrachtet werden. Eine Anfrage kann auf der einen Seite eine große Anzahl an Objekten als Ergebnismenge besitzen. Es wird dann jedoch mit großer Wahrscheinlichkeit nicht jedes gewonnene Objekt Relevanz für die Anfrage besitzen. Auf der anderen Seite kann sichergestellt werden, daß gewonnene Objekte mit großer Sicherheit für die Anfrage relevant sind. Dann ist jedoch mit einer kleineren Ergebnismenge zu rechnen. Schon durch die Art der Indizierung einer Textsammlung ist es möglich, ein System derartig zu steuern, daß es entweder sehr gute Rücklauf- oder sehr gute Genauigkeitswerte liefert. Es wird hier zwischen erschöpfender (*exhaustive*) und spezifischer (*specific*) Indizierung unterschieden [SM83]. Ein effektiv benutzbares Information Retrieval System läßt jedoch sowohl Anfragen zu, die so weit gefaßt sind, daß sie einen hohen Rücklauf haben, als auch solche, die aufgrund ihrer Detailliertheit eine hohe Genauigkeit nach sich ziehen.
Die Ermittlung von Rücklauf und Genauigkeit bringt sowohl praktische als auch Interpretationsprobleme mit sich. So ist etwa der Begriff der Relevanz nicht eindeutig definierbar. Prinzipiell hängt die Relevanz eines Textes stark von dem Wissensstand des Benutzers zu einem bestimmten Zeitpunkt ab. Liegt ein gewonnenes Dokument als erstes der Ergebnismenge vor, so mag es für den Benutzer relevant sein, doch schon wenn es nicht als erstes vom Benutzer angesehen wird, kann es diese Relevanz verlieren, weil der Benutzer die Information, die dieses Dokument beinhaltet, bereits aus einem zuvor betrachteten Dokument erhalten haben könnte. In [SM83] wird für diese an einen Zeitpunkt gebundene Relevanz der Begriff der Sachdienlichkeit (*pertinence*) eingeführt. Der Begriff Relevanz ist hingegen zeitunabhängig zu betrachten. Selbst wenn man den Begriff der Relevanz als den Grad interpretiert, zu dem ein Ergebnis eine Anfrage befriedigt, läßt sich dieser Wert einem Dokument bezüglich einer Anfrage nicht eindeutig zuweisen. Das probabilistische Modell (vgl. Abschnitt 3) erleichtert diese Zuweisung, indem es nicht wie Modelle mit exakter Übereinstimmung die Entscheidung fordert, ob ein Dokument relevant ist oder nicht, sondern angibt, wie wahrscheinlich es ist, daß das Dokument der Anfrage genügt. In der rein praktischen Ermittlung von Genauigkeit und Rücklauf unterscheiden sich Modelle mit exakter und Modelle mit bestmöglicher Übereinstimmung entsprechend. Während beim Modell mit

exakter Übereinstimmung nur jeweils ein einziger Wert für Genauigkeit und Rücklauf berechnet werden kann, ist es beim Modell mit bestmöglicher Übereinstimmung sinnvoll, der Rangfolge der gewonnenen Objekte folgend eine Reihe solcher Werte zu betrachten. In [SM83] wird speziell die Vorgehensweise bei der Bewertung von Modellen mit bestmöglicher Übereinstimmung eingehender beschrieben.
Ein praktisches Problem unabhängig von dem zugrundeliegenden Modell ergibt sich aus der Definition des Rücklaufs. Um den entsprechenden Wert berechnen zu können, muß die Anzahl aller relevanten Texte in der Datensammlung bestimmt werden. Bei kleinen Datensammlungen kann eventuell die Relevanz jedes einzelnen Textes manuell überprüft werden. Bei größeren Sammlungen jedoch ist dieses Verfahren nicht praktikabel. Hier werden zwei Vorgehensweisen verfolgt. Zum einen läßt sich die Gesamtanzahl der relevanten Dokumente mit Hilfe von Stichproben aus der Datensammlung bestimmen. Zum anderen kann man eine Anfrage mit verschiedenen Methoden des Information Retrieval auswerten in der Annahme, daß dadurch schließlich alle relevanten Dokumente der Sammlung gefunden und gezählt werden können.

4.2 Nützlichkeit als Alternative zu Rücklauf und Genauigkeit

Rücklauf- und Genauigkeitswerte werden häufig betrachtet, wenn Information Retrieval Systeme auf Standard-Datensammlungen, für die bereits im Vorwege optimale Anfrageergebnisse definiert wurden, getestet werden. Ihre Handhabung wird kompliziert, wenn die Datenmengen zugrundegelegt werden, die sich beispielsweise in Weitverkehrsnetzen befinden. Die Anzahl der Daten dort überschreitet die der Daten in Testsammlungen bei weitem, und es handelt sich dabei um dynamische Datenmengen. Zusätzlich besteht das Problem, daß die Relevanz von Texten bei der Ermittlung von Rücklauf und Genauigkeit nur statisch betrachtet wird. Der momentane Wissensstand eines oder gar mehrerer Benutzer geht nicht mit ein. Um derartigen Schwachstellen zu begegnen, ist das Maß der Nützlichkeit entwickelt worden. Bei der Ermittlung dieses Wertes weist ein Benutzer den vom System gewonnenen Objekten eine relative Relevanz zu, indem er sie vergleicht. Er bildet Paare (o, o') und legt dadurch fest, daß für ihn o' mehr Relevanz besitzt als o. In den Information Retrieval Systemen werden ebenfalls Relationen festgelegt, die sich aus der systeminternen Rangfolge der gewonnenen Objekte ergeben. Unter Verwendung der Benutzer- und der Systemrelationen lassen sich zwei Systeme stochastisch miteinander vergleichen. Die genaue Vorgehensweise hierzu findet sich in [FMS91]. Die ausschließliche Berücksichtigung gewonnener Dokumente führt dazu, daß der Aufwandd bei der Bewertung eines Systems gering bleibt. Das relative Maß bedeutet für menschliche Entscheidungsträger, die die Relevanz eines Objektes zu beurteilen haben, eine Vereinfachung. Es ist einfacher, zwei Objekte zu vergleichen als ein einzelnes unabhängig von anderen zu beurteilen.

5. INQUERY: Ein probabilistisches Information Retrieval System

Um dem wachsenden Interesse Rechnung zu tragen, das dem Information Retrieval aufgrund immer größerer verfügbarer Datenmengen und variierender Formen der Information entgegengebracht wird, wurde im Information Retrieval Laboratory der Universität von Massachusetts das System INQUERY entwickelt. Dieses dient der Forschung auf dem Gebiet des Information Retrieval mit verschiedenen Arten von Textrepräsentationen, Suchmodellen, Lernverfahren und Schnittstellen [CCH91].

INQUERY folgt dem probabilistischen Modell der Inferenznetze (vgl. Abschnitt 3.5), was den Vorteil hat, daß verschiedene Indizierungs- und Anfragekonzepte in diesem System verknüpft werden können.

5.1 Architektur von INQUERY

Die Aufgaben eines Information Retrieval Systems sind die rechneradäquate Darstellung der Dokumente der Textsammlung und des Informationsbedarfs des Benutzers sowie die Auswertung der Anfrage hinsichtlich der Textrepräsentationen. Die Architektur von INQUERY erfüllt diese Aufgaben folgendermaßen (vgl. Abb. 5.1).

Eine Textsammlung durchläuft zunächst ein Parser Subsystem, das die Dokumente für das Information Retrieval aufbereitet und dabei Transaktionen (*transactions*) erstellt. Diese enthalten Informationen über in den Dokumenten enthaltene Terme, die für die Generierung einer invertierten Datei (*inverted file*) verwendet werden. Die invertierte Datei ist die Grundlage, auf der das Retrieval Subsystem die Anfragen verarbeitet, die es nach einer dem Parsing ähnlichen Verarbeitung von der Benutzerschnittstelle erhält. Als Ergebnis liefert das Retrieval Subsystem die von ihm ermittelte Dokumentenrangfolge an die Benutzerschnittstelle zurück. Der Benutzer kann mit Hilfe einer Ansichtsdatenbank (*viewing database*), die ebenfalls vom Parser Subsystem bereitgestellt wird, die ihm relevant erscheinenden Dokumente einsehen.

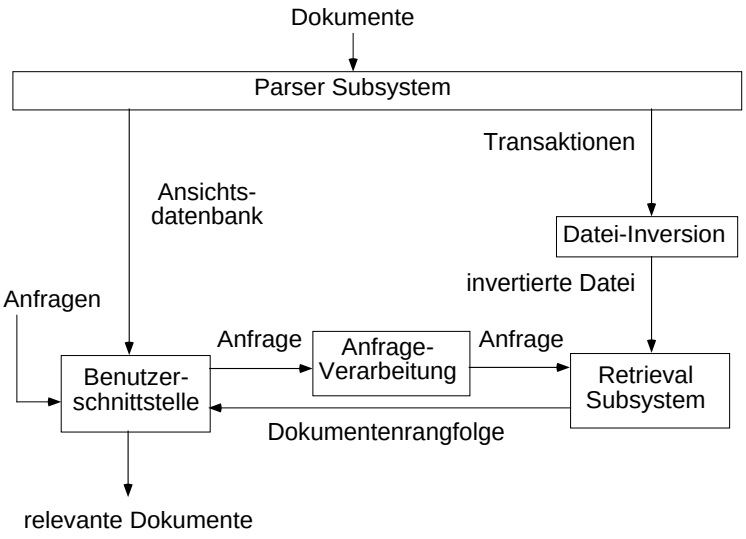

Abbildung 5.1: Die Architektur des Information Retrieval Systems INQUERY [CCH91]

5.1.1 Parser Subsystem

Um eine Textsammlung für das Information Retrieval nutzbar zu machen, müssen zunächst die enthaltenen Dokumente auf entsprechende Repräsentationen abgebildet werden (vgl. Kapitel 2). Dieser Vorgang wird als Parsing bezeichnet. Da INQUERY dem Modell der Inferenznetze folgt (vgl. Abschnitt 3.2.2), kann das Parsing auch als Aufbau der Kanten zwischen den Dokumenten- und den Konzeptknoten des Inhaltsnetzes angesehen werden [CCH91]. Das Parsing ist in vier Phasen aufgeteilt: Die Anordnungsanalyse (*layout analysis*), die lexikalische Analyse, die syntaktische Analyse und die Begriffserkennung (*concept identification*). Während dieser Phasen werden Transaktionen generiert, die jeweils einen Term, das dazugehörige Dokument sowie die Stellen im Dokument, an denen der Term vorkommt, beinhalten [BCC94]. Die Programmodule der einzelnen Parsingphasen führen die Transaktionen einem Transaktionsmanager zu, der sie für die spätere Datei-Inversion (*file inversion*) abspeichert [CCH91].

5.1.1.1 Anordnungsanalyse

Während der Anordnungsanalyse werden Anfang und Ende der Dokumente erkannt. Der Text wird von Informationen unterschieden, die für den Zweck des Information Retrieval irrelevant sind, wie z.B. von Formatierungsinformationen, Katalogisierungsinformationen oder ähnlichem. Optional können während der Anordnungsanalyse die Begrenzungen besonderer Felder (vgl. Kapitel 2), wie z.B. Titel, Haupttext usw., erkannt werden [BCC94]. Die Programme zur Anordnungsanalyse in INQUERY arbeiten standardmäßig mit einer Untermenge der Trennzeichen (*tags*) der *Standardized General Mark-up Language (SGML)*.

Da die Module zur Anordnungsanalyse mit Hilfe des Parsergenerators *yacc* [ASU88] erstellt wurden, ist es unproblematisch, diese durch Module, die andere Dokumentenformate verarbeiten, zu ersetzen oder zu ergänzen.

5.1.1.2 Lexikalische Analyse

Die Module für die lexikalische Analyse erkennen Wortbegrenzungen und generieren Transaktionen für die Indizierung der Wörter. Stoppwörter (vgl. Kapitel 2) können optional erkannt werden. Zu diesem Zweck steht den Modulen der lexikalischen Analyse eine Stoppliste zur Verfügung, die vom Benutzer spezifiziert wird. Wird ein Stoppwort erkannt, so wird dieses nicht indiziert. Es verbleibt jedoch im Text, damit es in den weiteren Analysephasen zur Verfügung steht. Die Stammformreduktion (vgl. Kapitel 2) ist ebenfalls optional [BCC94]. Die Module, die die lexikalische Analyse in INQUERY durchführen, liegen im *lex*-Format vor. *lex* ist ein standardisiertes UNIX-Werkzeug, das aus regulären Grammatiken endliche Automaten generiert [LS79]. Dadurch wird die Erstellung von Programmen ermöglicht, die beliebige Formate von Textsammlungen lexikalisch analysieren können.

5.1.1.3 Syntaktische Analyse

Die syntaktische Analyse dient zur Überprüfung der Konsistenz der Textsammlung. Ist diese nicht gegeben, so sorgen die Programme der syntaktischen Analyse für Fehlererholung. In INQUERY stehen verschiedene Programme zur syntaktischen Analyse zur Verfügung, die mit *yacc* erstellt wurden und sich die *SGML*-Trennzeichen in der Textsammlung zunutze machen [CCH91].

5.1.1.4 Begriffserkennung

INQUERY ermöglicht es, die Textsammlung mit verschiedenen Begriffserkennern (vgl. Kapitel 2) zu bearbeiten. Hat ein Begriffserkenner ein bestimmtes Muster identifiziert, so fügt er dem Dokument einen Meta-Indexterm hinzu und generiert eine entsprechende Transaktion. Wird beispielsweise ein Firmenname erkannt, so wird für die entsprechende Stelle im Dokument und für den Metaterm #COMPANY eine Transaktion generiert. Die Begriffserkenner in INQUERY liegen im *lex*- oder *flex*-Format vor.

5.1.2 Generierung der invertierten Datei

Jede Transaktion, die während des Parsing generiert wird, stellt eine Kante zwischen einem Dokumenten- und einem Konzeptknoten des Inhaltsnetzes dar. Die Gesamtmenge dieser Transaktionen entspricht dem Inhaltsnetz der Textsammlung. Der Wert eines internen Knoten in einem Inferenznetz errechnet sich über eine Funktion der Werte seiner Vaterknoten. Die Geschwindigkeit, mit der während des Information Retrieval der Wert eines Knotens berechnet wird, ist daher abhängig davon, wie schnell auf Information über diesen Knoten und vor allen Dingen über die ihn verbindenden Kanten zugegriffen werden kann. INQUERY stellt diese Information für die Konzeptknoten des Inhaltsnetzes als invertierte Datei zur

Verfügung. Die während des Parsings erkannten Terme und Konzepte, wie z.B. die Meta-Indexterme der Begriffserkennung, bilden die Schlüsselwerte der invertierten Datei. Jedem Term oder Konzept sind seine Vorkommenshäufigkeit in der gesamten Textsammlung, die Anzahl der Dokumente, in denen er vorkommt, und die Transaktionen, in denen er erscheint, zugeordnet [CCH91]. Die invertierte Datei bildet eine wichtige Basis für die Verarbeitung und die Auswertung der Anfragen [CIIR95b].

5.1.3 Verarbeitung der Anfragen

Vor dem eigentlichen Anfrageprozeß muß zunächst spezifiziert werden, auf welche der vorhandenen INQUERY Datenbanken zugegriffen werden soll. Ein gleichzeitiger Zugriff auf mehrere Datenbanken ist in der vorliegenenden Version des Systems nicht möglich.
INQUERY läßt Anfragen in natürlicher Sprache, aber auch in einer strukturierten Anfragesprache (siehe Anhang B) zu. Anfragen in natürlicher Sprache werden zunächst von Textbearbeitungsmodulen (*text processing modules*) interpretiert, um zu gewährleisten, daß die lexikalische Analyse und die Begriffserkennung hier genauso ablaufen wie im Parser Subsystem. Die Resultate der Textbearbeitung werden in strukturierte Anfragen umgewandelt, indem auf die Terme der Anfrage der *#sum*-Operator angewendet wird. Anhang B und die Gleichungen 5.1 - 5.6 in Abschnitt 5.1.4 beschreiben *#sum* sowie die anderen Operatoren der strukturierten Anfragesprache von INQUERY. Durch die Operatoren der Anfragesprache ist es den Textbearbeitungsmodulen oder dem Benutzer möglich, zusätzlich zu den bloßen Termen strukturelle Information in die Anfrage einzubinden. Unter anderem kann so nach ganzen Phrasen, wie z.B. *"House of Representatives"*, oder nach mehreren Termen in gewissem Abstand zueinander gesucht werden [BCC94]. Die Operatoren der Anfragesprache werden in das Anfragenetz integriert (vgl. Abbildung 5.2).
Wie die Dokumente der Textsammlung können auch die Anfragetexte auf Stoppwörter untersucht und auf ihre Wortstämme reduziert werden (vgl. Abschnitt 5.1.1).
Die Vorgänge während der Anfragebearbeitung müssen zumindest denen im Parser Subsystem entsprechen, da sonst die Gefahr besteht, daß Information Retrieval Prozesse nicht mehr die gewünschten Ergebnisse liefern können (vgl. Kapitel 2). Zusätzlich zu den Modulen, die diese Anforderung erfüllen, enthält INQUERY weitere Prozessoren zur Bearbeitung des Anfragetextes. Einer davon erkennt Wortgruppen, die entweder durch Bindestriche verbunden oder durch Großschreibung gekennzeichnet sind. Bindestriche werden während des Parsings grundsätzlich entfernt. In den Dokumenten werden aus Ausdrücken, wie *Iran-Contra*, einzelne Terme, wie *Iran Contra*. Um dennoch sinnvoll nach dem Gesamtbegriff suchen zu können, wird bei der Anfrageverarbeitung der Bindestrich entfernt und die einzelnen Wörter werden mit dem Nachbarschaftsoperator *#1* verbunden. In der Anfrage wird aus *Iran-Contra* also *#1(Iran Contra)*. Dadurch wird erzwungen, daß die beiden Wörter im Dokument unmittelbar nebeneinander stehen müssen, um die Anfrage zu erfüllen. Entsprechendes geschieht mit Gruppen von Wörtern in Großschreibung. So wird *House of Representatives* in einer Anfrage zu *#2(House Representatives)* [BCC94].
Ein weiterer Textprozessor in INQUERY entfernt Ausdrücke, die sich auf den Information Retrieval Prozeß und nicht auf den Inhalt der gesuchten Dokumente beziehen, wie beispiels-

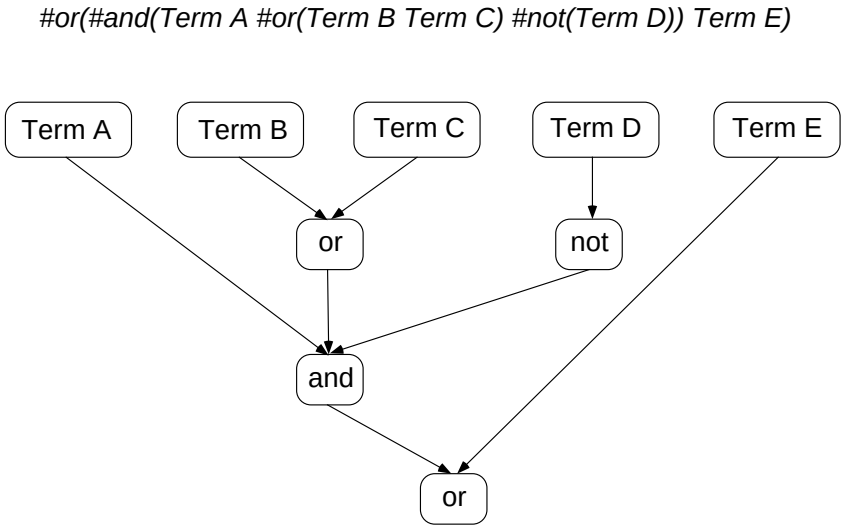

Abbildung 5.2: Ein Teil eines INQUERY Anfragenetzes [HKW94]

weise *"A relevant document must contain ..."* oder *"I would like to know something about..."*. Ob solche Ausdrücke entfernt werden können, hängt jedoch von dem Fachgebiet ab, für das Information Retrieval betrieben wird. Daher müssen die Regeln dieses Textprozessors an den Benutzungskontext angepaßt werden [BCC94].
Die Textprozessoren für die Anfrageverarbeitung in INQUERY liegen im *flex*-Format vor. Eine zusätzliche Bearbeitung der Anfragen mit weiteren Textprozessoren vor der Bearbeitung durch INQUERY ist möglich. Die Anfragekonzepte des Systems lassen sich hierdurch ergänzen [BCC94].

5.1.4 Retrieval Subsystem

Das Anfragenetz dient als Eingabe für das Retrieval Subsystem, das es mit dem Inhaltsnetz der Textsammlung verbindet. Im eigentlichen Information Retrieval Vorgang wird das vollständige Inferenznetz wie folgt ausgewertet.
Zunächst wird bestimmt, mit welcher Wahrscheinlichkeit ein Konzeptknoten Q des Anfragenetzes den Informationsbedarf erfüllt, unter der Annahme, daß alle Dokumente der Textsammlung betrachtet werden. Dieser Wert $bel(Q)$ wird auch als Erfüllungsgrad (*belief value*) des Knotens bezeichnet und ist vom Typ des Knotens abhängig. Er kann durch die folgenden Formeln unter Einsetzung der Bewertungen p_i und Gewichtungen w_i der jeweiligen n Vorgängerknoten ermittelt werden. Der Erfüllungsgrad eines Negationsknotens $bel_{not}(Q)$ ist dabei nur von einem Vaterknoten abhängig.

$$bel_{not}(Q) = 1 - p \tag{5.1}$$

$$bel_{or}(Q) = 1 - (1 - p_1) \cdot \ldots \cdot (1 - p_n) \tag{5.2}$$

$$bel_{and}(Q) = p_1 \cdot p_2 \cdot \ldots \cdot p_n \tag{5.3}$$

$$bel_{max}(Q) = max(p_1, p_2, \ \ldots \ , p_n) \tag{5.4}$$

$$bel_{sum}(Q) = \frac{p_1 + p_2 + \ldots + p_n}{n} \tag{5.5}$$

$$bel_{wsum}(Q) = \frac{(w_1 p_1 + w_2 p_2 + \ldots + w_n p_n) \cdot w_q}{w_1 + w_2 + \ldots + w_n} \tag{5.6}$$

Nun werden der Reihe nach alle Dokumentenknoten d_i des Inhaltsnetzes betrachtet. Die Bewertung eines jeden Knotens d_m wird über das Netz weitergegeben und der Knoten wird markiert. Dadurch wird die bedingte Wahrscheinlichkeit $P(I|d_m)$ des Ereignisses berechnet, daß das Dokument d_m den Informationsbedarf I des Benutzers deckt [HKW94]. Die Details der Auswertung mit Hilfe von Inferenznetzen beschreibt [TC91].

5.2 Schnittstellen von INQUERY

INQUERY kann im Stapelbetrieb ausgeführt werden. Auch eine Benutzerschnittstelle zur direkten Interaktion mit dem System ist vorhanden. Eine Schnittstelle für Anwendungsprogrammierung (*application programmer's interface, API*) unterstützt die Entwicklung eigener Benutzerschnittstellen sowie die Integration von INQUERY in eigene Anwendungen.

5.2.1 INQUERY im Stapelbetrieb

INQUERYs Stapelprogramm erhält seine Argumente in Form von Dateinamen und booleschen Attributen aus der Kommandozeile. Hieraus erzeugt es eine Dokumentenrangliste, die von einem Auswertungsprogramm verarbeitet werden kann. Dieses Auswertungsprogramm erzeugt Rücklauf- und Genauigkeitstabellen, die über die Wirksamkeit des Systems Auskunft geben (vgl. Abschnitt 4.1). Die Stapelverarbeitung erleichtert die mehrfache Ausführung von Anfragen, so daß hier die Auswirkungen von Änderungen am System betrachtet werden können [CCH91].

5.2.2 Benutzerschnittstelle

Die Benutzerschnittstelle von INQUERY ermöglicht die direkte Eingabe sowohl von natürlichsprachlichen als auch von strukturierten Anfragen. Die Ergebnisse der Anfragen werden dem Benutzer in der vom System ermittelten Rangfolge angezeigt. Die Dokumente erscheinen dabei in Kurzform, können jedoch durch eine entsprechende Auswahl des Benutzers auch vollständig auf den Bildschirm gebracht werden. Der Benutzer hat die Möglichkeit, die für

seinen Informationsbedarf relevanten Dokumente zu markieren. Das System ist in der Lage, die ursprüngliche Anfrage aufgrund dieser Markierungen zu verändern. Es unterstützt somit das Konzept der Relevanzrückkopplung (vgl. Kapitel 2). Die Ergebnisse des Information Retrieval Prozesses können in einer Datei abgespeichert werden [CIIR95a].

5.2.3 Schnittstelle für Anwendungsprogrammierung

Die Schnittstelle für Anwendungsprogrammierung ermöglicht die Implementation einer eigenen Benutzerschnittstelle zum System INQUERY sowie deren Integration in eigene Anwendungen. Hierzu stellt sie einen Satz von Funktionen zur Verfügung, die in der Programmiersprache C [KR77] geschrieben wurden. Diese können von jedem C-Programm oder jedem anderen Programm, das C-Funktionen einbinden kann, aufgerufen werden.
Die Funktionen sind in zwei Bibliotheken zusammengefaßt. Die erste Bibliothek enthält die Funktionen für lokale Anwendungen, die zweite die Funktionen für Netzwerkanwendungen. Dadurch ist Information Retrieval auf lokalen, aber auch auf entfernten INQUERY-Datenbanken möglich. Hierbei ist jedoch nur eine der beiden Anwendungsarten zur Zeit zulässig.
Die Benutzung von C-eigenen Datentypen wird in INQUERY-Funktionen vermieden, um eine Portabilität des Systems zu erreichen. Die Annahmen, die INQUERY zum jeweiligen, im System definierten Datentyp trifft, sind in einer speziellen C-Header-Datei festgelegt.
Die Funktionen zur Anwendungsprogrammierung werden funktionalen Gruppen zugeordnet. Diese Gruppen umfassen [CIIR94]:

- den Zugang zu INQUERY-Datenbanken
- das Parsing von Anfragen
- die Auswertung von Anfragen
- die Relevanzrückkopplung
- den Zugang zu Dokumenten und Informationen über die Textsammlung
- die Indizierung
- Fehlerbehandlung
- sonstige Funktionen

Mit Hilfe der Funktionsgruppen für den Datenbankzugang, für das Parsing und die Auswertung von Anfragen sowie für den Zugang zu Dokumenten können einfache Anwendungen des Information Retrieval auf Standard-Textsammlungen realisiert werden. Diese lassen sich unter Benutzung der übrigen Funktionsgruppen erweitern.

6. WAIS: Ein Vektorraum Information Retrieval System

Als Beispiel für ein weiteres Information Retrieval System, das auf einem Modell mit bestmöglicher Übereinstimmung basiert, wird im folgenden das System WAIS[1] vorgestellt. Die Grundlage des Information Retrieval in WAIS ist das Vektorraum-Modell (vgl. Abschnitt 3.2.1).

Bei WAIS, das erstmals 1991 von der Firma Thinking Machines präsentiert wurde, handelt es sich um ein System, dessen Client-Server-Architektur eine verteilte Nutzung ermöglicht. Das zunächst als Public-Domain-Version freigegebene System wurde Mitte 1992 kommerzialisiert. Die freigebenen Versionen wurden parallel zum kommerziellen Produkt weiterentwickelt, so daß es mittlerweile eine Reihe von unterschiedlichen Systemen gibt, die WAIS zur Grundlage haben.

6.1 Architektur von WAIS

Das WAIS-System besteht aus einem Client und einem Server, die über ein systemeigenes Protokoll kommunizieren, sowie aus Dokumentendatenbanken, auf die der Server zugreift (vgl. Abbildung 6.1).

Der Client stellt eine Benutzerschnittstelle zur Eingabe von Anfragen zur Verfügung. Es gibt zeilenorientierte, bildschirmorientierte und fensterorientierte WAIS-Clients, die einen unterschiedlichen Leistungsumfang besitzen. Der Client nimmt über ein Protokoll Verbindung mit dem WAIS-Server auf. Dieses Protokoll ist eine Erweiterung des Z39.50-Standards, der ursprünglich für die Suche in konventionellen Bibliotheken entworfen wurde [NIS88]. Der Server bereitet die Anfrage für das Information Retrieval auf und sucht in einer oder mehreren vom Benutzer vorgegebenen Dokumentendatenbanken nach entsprechenden Dokumenten. Kurzbeschreibungen dieser Dokumente sendet er an den Client, der sie dem Benutzer anzeigt. Dieser kann Dokumente auswählen, die der Client vom Server anfordert und für den Benutzer ausgibt. Einige WAIS-Versionen ermöglichen die Speicherung von Anfragen und Teilen der Ergebnisse für eine erneute Benutzung oder zur Relevanzrückkopplung.

[1] WAIS: Wide Area Information Server

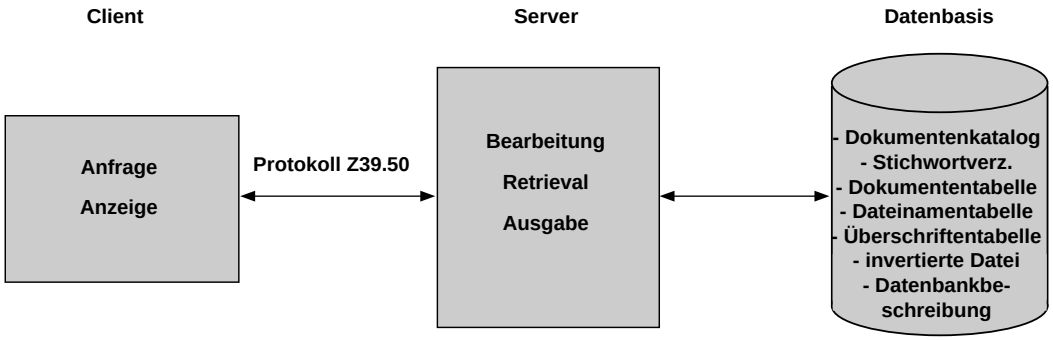

Abbildung 6.1: Die WAIS-Komponenten [SBGK94]

6.2 Datenbanken unter WAIS

Die Datenbasis unter WAIS wird mit Hilfe des Programms *waisindex* generiert. Dabei ist es möglich, rekursiv ganze Verzeichnisse und Unterverzeichnisse zu indizieren (vgl. Kapitel 2). Dokumente verschiedener Typen und Formate können aufgenommen werden. Hierbei bezieht sich das Format eines Dokuments darauf, wie das Dokument strukturiert ist, während der Typ impliziert, welche Software der Client später verwenden kann, um das Dokument anzuzeigen. Typische Dokumentenformate sind *text*, *bibtex* oder *mail_or_rmail*. Die Definition neuer Formate über eine Konfigurationsdatei ist möglich. Beispiele für Dokumententypen in WAIS sind *GIF*, *HTML* oder *PS* [Pfe95]. Zusätzlich können auch zusammengehörige Dokumente unterschiedlichen Typs als Multi-Typ-Dokumente indiziert werden. Ein Bild im Binärformat beispielsweise kann mit einem Text, der Erklärungen dazu enthält, assoziiert werden. Während sonst nur Begriffe aus der Überschrift des Bildes als Suchbegriffe in Betracht kommen, ist bei einem Multi-Typ-Dokument der erklärende Text ebenfalls suchrelevant [SBGK94].

Das Programm *waisindex* erzeugt bei der Indizierung verschiedene Dateien, wie z.B. einen lesbaren Katalog aller Dokumente, ein Stichwortverzeichnis, die invertierte Datei und eine lesbare Datenbankbeschreibung. Die Datenbankbeschreibung enthält wichtige Informationen bezüglich der Datenbasis und des dazugehörigen Servers. So ist hier die komplette Netzwerkadresse des Servers angegeben, die der WAIS-Client zum Verbindungsaufbau nutzen kann. Weiterhin ist in der Datenbankbeschreibung der Inhalt der Datenbasis dargelegt. Die Datenbankbeschreibungen werden zentral gesammelt und in einer eigenen Datenbank (*Directory-of-Servers*) verwaltet. Der Benutzer eines WAIS-Systems hat die Möglichkeit, Anfragen an dieses Datenbankverzeichnis zu stellen, dadurch Datenbanken zu finden, die seinem Suchgebiet entsprechen, und in einem nächsten Schritt in diesen speziellen Datenbanken zu suchen.

6.3 Konzepte des Information Retrieval in WAIS

Die Anfrageeingabe in WAIS erfolgte zunächst ausschließlich in natürlicher Sprache. Mittlerweile existieren jedoch Systemversionen, die zusätzlich die Eingabe boolescher Anfragen sowie die Trunkierung von Termen ermöglichen. Ein Verfahren zur phonetischen Suche (vgl. Kapitel 2) ist in WAIS implementiert.
Bei der Indizierung und der Anfragebearbeitung können eine Stoppwortelimitierung und eine Stammformreduktion (vgl. Kapitel 2) optional durchgeführt werden.
Das System bietet die Möglichkeit der Relevanzrückkopplung. Dabei wird dem Benutzer ermöglicht neben ganzen Dokumenten auch Dokumententeile als relevant zu markieren. Hierdurch kann die Relevanzrückkopplung auf den markierten Dokumententeil beschränkt werden. Als relevant markierte Texte werden jeweils nur auf der Datenbank zur Relevanzrückkopplung eingesetzt, aus der sie stammen. Für alle anderen Datenbanken, an die der Benutzer zum jeweiligen Zeitpunkt Anfragen stellt, sind die Rückkopplungsdaten nicht verwendbar.
Die Indizierung von ausgewählten Dokumentenfeldern und die eingeschränkte Suche auf diesen Feldern ist in einzelnen WAIS-Versionen realisiert.

7. Tycoon: Eine persistente, interoperable Systemumgebung

Im Rahmen des Projektes Tycoon[1] am Arbeitsbereich Datenbanken und Informationssysteme des Fachbereichs Informatik der Universität Hamburg wurde eine offene, heterogene Systementwicklungsumgebung geschaffen. Diese sollte zum einen eine persistente, polymorphe Programmiersprache mit einem Typsystem höherer Ordnung zur Verfügung stellen, um Programmierer bei der Erstellung neuer und der Einbindung bereits existierender generischer Dienste zu unterstützen. Zum anderen sollte dieser Sprache eine Architektur zugrunde liegen, die es möglich macht, bei der Programmierung von allen technischen Anforderungen außerhalb der Programmiersprache zu abstrahieren.

7.1 Architektur des Tycoon-Systems

Das Tycoon System läßt sich in verschiedene Schichten einteilen, die ihrer Aufgabe nach zu unterscheiden sind (vgl. Abbildung 7.1).

7.1.1 Tycoon Speicherprotokoll

Das Tycoon Speicherprotokoll (*Tycoon Store Protocol, TSP*) verbindet die verschiedenen Objektspeicher des Systems mit der Tycoon Maschine (*Tycoon Machine, TM*) [MSS95]. Dazu stellt es Funktionen, wie z.B. das Einrichten oder Löschen eines Speichers oder das Anlegen neuer Objekte im Speicher, bereit. Spezielle Merkmale der zugrunde liegenden Objektspeichersysteme bleiben hierbei verborgen. Das TSP ist in ANSI C realisiert.

7.1.2 Virtuelle Tycoon Maschine

Die Tycoon Maschine umfaßt den Interpreter des Systems und das dazugehörige Laufzeitsystem, die beide in ANSI C erstellt sind und bereits auf verschiedene Betriebssystemplattformen portiert wurden. Weiterhin enthält die TM eine Reihe von statisch oder dynamisch gebundenen Objektbibliotheken, die dem System zusätzliche externe Dienste zur Verfügung stellen. Die Schnittstelle, die den Zugriff auf die TM ermöglicht, heißt virtuelle Tycoon

[1] Tycoon: Typed communicating objects in open environments.

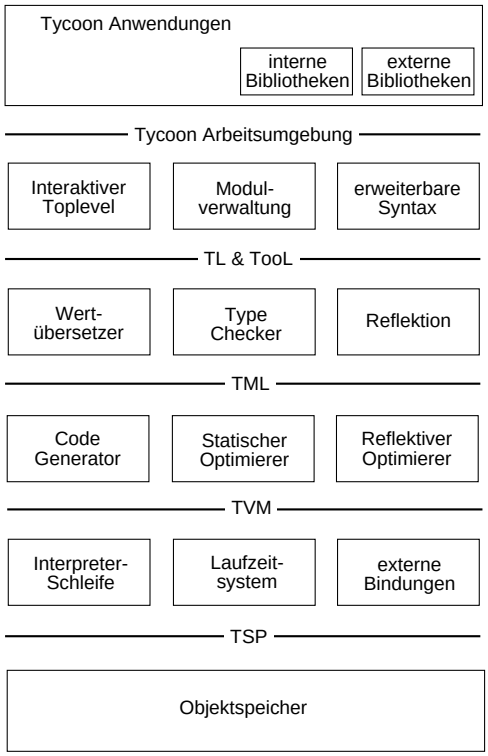

Abbildung 7.1: Ein Überblick über die Tycoon Systemschichten [MSS95]

Maschine (*Tycoon Virtual Machine, TVM*) [MSS95]. Sie besteht aus einer Reihe von Befehlen in Bytecode, der zwischen heterogenen Rechnerknoten ohne eine erneute Übersetzung übermittelt werden kann.
Die externen Bibliotheken können unabhängig davon, ob sie statisch oder dynamisch gebunden sind, von der TVM genutzt werden. Hierzu müssen zur Laufzeit der Name der Bibliothek und der aufgerufenen Funktion sowie eine Beschreibung der Funktionsargumente und ihrer Typen übergeben werden. Der Interpreter verfügt über eine Liste statisch gebundener Bibliotheken und Funktionen, die er zunächst nach der aufgerufenen Funktion durchsucht. Ist diese Suche erfolglos, so ruft der Interpreter eine Betriebssystemroutine auf, die versucht, eine dynamische Bindung der Funktion herzustellen.
Es kann jedoch nicht nur im Code der TVM auf externe Funktionen zugegriffen werden. Auch der Aufruf von TVM-Code in externem Code ist möglich. Durch einen speziellen Bindemechanismus wird ein externer Zeiger auf das jeweilige TVM-Funktionsobjekt erzeugt und nach außen übergeben.
Der Austausch von Funktionen zwischen dem Tycoon System und externen Bibliotheken beschränkt sich auf Funktionen, deren Parameter unstrukturierte Werte, wie z.B. Zahlen, Zeichenketten aber auch Zeiger, sind. Strukturierte Werte müssen vor der Übergabe in ihre

Komponenten aufgegliedert und anschließend wieder zusammengesetzt werden.
Durch den Aufruf externer Funktionen im TVM-Code kann es dazu kommen, daß persistente TVM-Programme mit Zeigern arbeiten, die in Speicherbereiche außerhalb des TSP-Speichers verweisen. Die Lebensdauer von Objekten außerhalb des TSP-Speichers ist jedoch auf die Dauer eines Betriebssystemprozesses begrenzt. Dadurch besteht die Gefahr, daß das persistente Programm nach einem Neustart des Tycoon Systems auf Objekte zugreift, die nicht mehr im Speicher vorhanden sind. Um dieser besonderen Anforderung Rechnung zu tragen, bietet die TVM einen Mechanismus, der das Anlegen und Löschen von Objekten außerhalb des TSP-Speichers aufzeichnet. Anhand der Aufzeichnungen ist es möglich, die betreffenden Objekte wiederherzustellen, falls persistenter TVM-Code darauf zugreifen sollte.

7.1.3 Weitere Schichten des Tycoon-Systems

Der Bytecode der TVM wird aus Termen der Tycoon Maschinensprache (*Tycoon Machine Language, TML*) generiert. Diese dient als Zwischenrepräsentation zwischen den verschiedenen höheren Programmiersprachen des Systems und der TVM. Ihre Aufgabe ist es, die Code-Analyse zur Übersetzungszeit und die Code-Analyse zur Laufzeit zu vereinheitlichen und den Code der höheren Programmiersprachen zu optimieren.
Im Laufe des Tycoon Projektes wurden die Tycoon Sprache (*Tycoon Language, TL*, vgl. Abschnitt 7.2) und die objektorientierte Tycoon Sprache (*Tycoon object-oriented Language, TooL*) als höhere Programmiersprachen entwickelt. Sie setzen auf der TML auf und sind ihrerseits in eine Arbeitsumgebung integriert, die die interaktive Programmierungs- und Benutzungsschnittstelle (*Tycoon top level*) und die Modulverwaltung umfaßt sowie das Konzept der erweiterbaren Syntax des Systems realisiert.
Anwendungen, die in den höheren Programmiersprachen des Tycoon-Systems implementiert sind, werden in Bibliotheken verwaltet.

7.2 Programmiersprache TL

Eine der höheren Programmiersprachen, die in das Tycoon System integriert sind, ist die Tycoon Sprache TL [MMS94]. Sie besitzt einheitliche Benennungs-, Typisierungs- und Bindungskonzepte für alle Sprachelemente, die damit zu Elementen erster Klasse werden und orthogonal kombiniert werden können.
TL folgt dem Konzept des Polymorphismus. Sowohl Subtyppolymorphismus als auch parametrischer Polymorphismus sind hier realisiert. Auf diese Weise wird die Kombination verschiedener TL-Programmteile vereinfacht.
Die Orthogonalität, der Polymorphismus und die strenge Typisierung in TL ermöglichen die Definition, Integration und Kombination verschiedener generischer Dienste. Durch die polymorphe Typisierung wird eine Abstraktion von den Besonderheiten des jeweiligen Dienstes erreicht. Das Typsystem unterbindet statisch die fehlerhafte Verwendung von Operationen und gewährleistet hierdurch die Konsistenz zwischen den verschiedenen Diensterbringern.

Die einzelnen Dienste sind in einem System aus Modulen, Schnittstellen und Bibliotheken eingebunden (vgl. Abschnitt 7.2.4). TL beinhaltet ein Sprachkonstrukt zur Anbindung von Bibliotheken und Programmierschnittstellen der Programmiersprache C.
TL ist über die Tycoon Maschinensprache TML mit den weiteren Schichten des Tycoon Systems verbunden (vgl. Abschnitt 7.1.3). Dadurch kann in der Sprache die Persistenz des Systems genutzt werden, ohne die systemimmanenten Anforderungen, wie z.B. Verwaltung des Haupt- und Zwischenspeichers oder Speicherbereinigung (*garbage collection*), erwägen zu müssen.
Das Laufzeitsystem von TL unterstützt ein Verfahren zur Fehlerbehandlung, das Programme auch in zuvor definierten Ausnahmezuständen korrekt terminieren läßt.
Die Sprache TL wird in einer interaktiven Programmier- und Benutzungsumgebung, dem *Tycoon top level*, verwendet, die der Umgebung interaktiver Sprachen, wie ML oder Lisp, ähnlich ist. Der Tycoon top level unterstützt die direkte Eingabe von TL-Konstrukten, die Ausführung von TL-Programmen und die Eingabe von Befehlen, die den Zustand des top levels selbst betreffen. Jeder Benutzer arbeitet auf einem eigenen top level, der dadurch jeweils einen lokal begrenzten Namensraum bildet.

7.2.1 Typisierung in TL

TL ist eine strikt typisierte Sprache [Mat93]. Wert- und Typbezeichner können an jeder Stelle eines TL-Programmes explizit durch die ihnen entsprechende Typinformation ergänzt werden. Für Bezeichner, die dynamischen oder rekursiven Bindungen unterliegen, wie z.B. Parameter von Funktionen, ist diese explizite Form der Typisierung obligatorisch. Für andere Bezeichner kann die Typinformation bei der Übersetzung abgeleitet werden [Mat93].
Die Typisierung eines Bezeichners x mit dem Typ A wird in TL wie folgt zum Ausdruck gebracht:

$x : A$

Ein TL-Konstrukt dieser Form heißt Signatur. Durch eine Signatur wird der betreffende Bezeichner jedoch nur teilweise spezifiziert. Ein Wert der Variablen x muß zumindest die Spezifikation, die durch den Typ A gegeben ist, erfüllen. Er kann darüber hinaus aber weitere Merkmale besitzen. Hieraus ergibt sich eine partielle Ordnung auf der Menge der TL-Typen, die sich in der Subtyprelation der Sprache ausdrückt. Das Konstrukt

$B <: A$

bedeutet, daß B präziser ist als A oder, mit anderen Worten, daß B Subtyp von A ist [MMS94]. Die induktiv definerte Subtyprelation ist eine der Grundlagen des in TL realisierten Konzeptes des Polymorphismus [Mat93].

7.2.2 Persistenz in TL

Das Konzept der Persistenz besteht in TL gleichermaßen für Werte, Funktionen und Typbindungen. Persistente TL Objekte sind entweder aus einem gebundenen Modul oder vom Tycoon top level aus erreichbar.

Wird im Rahmen eines TL-Programmes ein konsistenter Zustand des Tycoon Speichers erreicht, so kann dieser Zustand explizit stabilisiert werden. Dafür steht sowohl eine Funktion innerhalb eines Moduls als auch ein top level Befehl zur Verfügung. Beide sichern den augenblicklichen Zustand des Tycoon Speichers. Verläßt der Benutzer den top level oder kommt es zu einem Systemabsturz, so werden alle Änderungen seit der letzten Stabilisierung zurückgenommen. Zu Beginn der nächsten Sitzung mit dem Tycoon System befindet sich der Tycoon Speicher somit in dem zuletzt stabilisierten Zustand.
Das Rücksetzen des Systems auf einen Stabilisierungspunkt während einer Sitzung ist ebenfalls möglich [MMS94].

7.2.3 Anbindung externer C-Bibliotheken

Der Austausch von Funktionen zwischen den Programmiersprachen TL und C ist in beide Richtungen möglich. C-Funktionen können als Funktionswerte in TL aufgenommen werden, und TL-Funktionen können so umgeformt werden, daß sie in C in Form eines Zeigers auf eine Funktion genutzt werden können [MMS94].

Aufruf externer C-Funktionen in TL

Durch die vordefinierte Funktion *bind* werden externe Funktionen an TL-Funktionswerte gebunden. Die Funktion besitzt folgende Signatur:

*bind(Function <:**Ok** library, label, format :String) :Function*

Der Funktion *bind* wird mit *Function* eine Beschreibung der resultierenden TL-Funktion übergeben. Der Parameter *library* bezeichnet die Bibliotheksdatei, in der sich die anzubindende C-Funktion befindet. Der Parameter *label* enthält den Namen dieser Funktion. Die Zeichenkette *format* beschreibt das Parameterformat der C-Funktion. Hierbei steht jedes einzelne Zeichen für einen C-Parameter. Die Zeichen sind wie die Parameter in C von links nach rechts geordnet. Der Rückgabeparameter ist durch das letzte Zeichen der Zeichenkette dargestellt.

Die Anbindung der C-Funktion

```
void inq_set_query_stemming (Boolean_t on_off)
```

sieht in TL folgendermaßen aus:

__let__ lib = "/local/dbis1/software/inquery/lib/libinquery.a"

__let__ setStemmingCall =
bind(:__Fun__(:Bool) :__Ok__ lib "inq_set_query_stemming" "bv")

Ein Aufruf von *setStemmingCall* in TL

setStemmingCall(true)

führt zur Ausführung der C-Funktion *inq_set_query_stemming*.

Aufruf von TL-Funktionen aus externen C-Programmen

Nicht nur der Aufruf von C-Funktionen aus TL heraus ist möglich, sondern auch umgekehrt der Aufruf von TL-Funktionen in externen C-Programmen. Dieses ist dann sinnvoll, wenn aus TL eine C-Funktion aufgerufen wird, die eine weitere Funktion als Übergabeparameter verlangt. Diese Parameterfunktion kann dann in TL definiert und in C benutzt werden.
Um den Implementationsaufwand auf der C-Seite möglichst gering zu halten, werden die hier zu nutzenden TL-Funktionen als C-Funktionszeiger übergeben. Der entsprechende TL-Typ hierfür ist im Modul *cCallback.tm* unter dem Namen *cCallback.T* definiert. Ein Wert dieses Typs kann mit der Funktion *cCallback.new*, die ebenfalls in *cCallback.tm* implementiert ist, erzeugt werden.

7.2.4 Module, Schnittstellen und Bibliotheken

Die strukturierte Implementation großer Programme wird in TL durch Modularisierung unterstützt. Das Gesamtprogramm wird in überschaubare Teile gegliedert, deren Code sich in Modulen befindet. Den Modulen entsprechende Schnittstellen definieren in einer Exportliste die nach außen hin sichtbaren Signaturen von Werten, Funktionen, Typen und Typoperatoren.
Da in realen Systemen wie dem Tycoon System die Zahl der Module schnell ansteigt, beinhaltet TL zusätzlich ein Bibliothekskonzept. Eine Bibliothek begrenzt den Sichtbarkeitsbereich der in ihr enthaltenen Module und Schnittstellen und erlaubt somit die Definition von Subsystemen. Die Implementation versteckter Module und Schnittstellen sowie die Zuordnung mehrerer Module zu einer Schnittstelle ist in TL-Bibliotheken zulässig.

8. Anbindung des Systems INQUERY an die Programmiersprache Tycoon

Die Schnittstelle für Anwendungsprogrammierung des Systems INQUERY ist in einer Reihe von C-Dateien implementiert (vgl. Abschnitt 5.2.3), die zusammen eine Bibliothek bilden. Diese wird statisch über die virtuelle Tycoon Maschine (vgl. Abschnitt 7.1.2) an das Tycoon System gebunden.
Ein Teil der in der INQUERY Schnittstelle enthaltenen Funktionen wurde zunächst eins zu eins in die Programmiersprache TL (vgl. Abschnitt 7.2) übernommen. Nach der Erstellung eines groben Datenmodells wurden diese Funktionen unter Berücksichtigung spezieller Eigenschaften von TL, wie z.B. der Typisierung und der Fehlerbehandlung, überarbeitet.
Die Struktur des Tycoon Systems führte bei der Anbindung zu besonderen Anforderungen hinsichtlich der Übergabe und Nutzung von C-Funktionen. Die strenge Typisierung der Programmiersprache TL wurde genutzt, um den korrekten Ablauf von INQUERY Operationen zu gewährleisten. Durch die Persistenz und die Interoperabilität des Tycoon Systems wurde der Nutzungsbereich des INQUERY Systems erweitert.

8.1 Angebundene Funktionsgruppen

Das Hauptinteresse bei der Anbindung gilt jenen Funktionsgruppen der C-Schnittstelle, die die Grundfunktionalität des INQUERY Systems enthalten. Die Gruppen, in denen

- der Zugang zu INQUERY Datenbanken,
- das Parsing von Anfragen,
- die Auswertung von Anfragen und
- der Zugang zu Dokumenten und Informationen über die Textsammlung

implementiert sind, werden im Tycoon System zugänglich gemacht. Hierdurch können in TL Anfragen an im INQUERY System enthaltene Testdatenbanken gestellt sowie die entsprechenden Ergebnisse verarbeitet werden. In TL wird die Gruppierung nach Funktionalität beibehalten, indem Funktionen verschiedener Gruppen in verschiedenen Schnittstellen

und Modulen (vgl. Abschnitt 7.2.4) zusammengefaßt werden. Hierdurch entsteht eine TL-Bibliothek, die folgende Dateien umfaßt:

- die Schnittstelle *Dbinfo.ti* (vgl. Anhang C.1) und das dazugehörige Modul *dbinfo.tm* für den Zugang zu INQUERY Datenbanken
- die Schnittstelle *Query.ti* (vgl. Anhang C.2) und das dazugehörige Modul *query.tm* für das Parsing und die Auswertung von Anfragen
- die Schnittstelle *Docs.ti* (vgl. Anhang C.3) und das dazugehörige Modul *docs.tm* für den Zugang zu Dokumenten und Informationen über die Textsammlung

8.2 Datenmodell des Systems INQUERY

Um einen Überblick über die verschiedenen Daten des INQUERY Systems zu gewinnen, wird unter Benutzung der *Object Modelling Technique OMT* [RBP+91] ein Datenmodell erstellt (vgl. Abbildung 8.1).

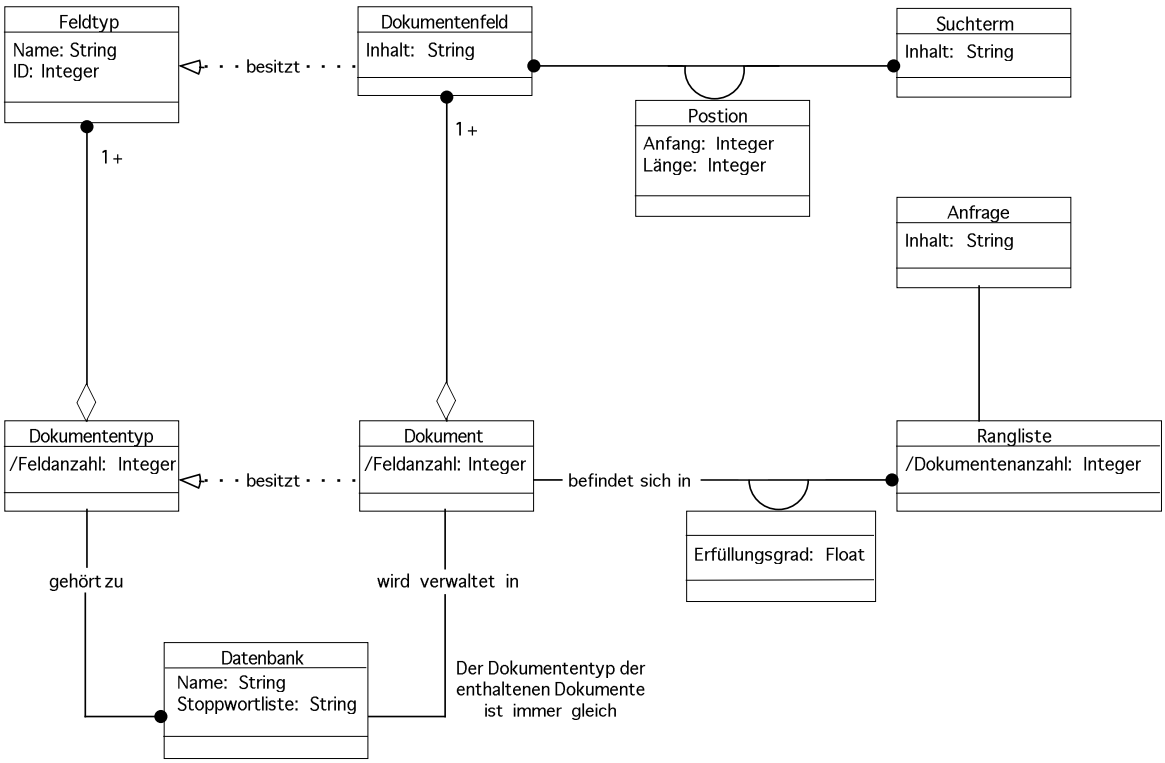

Abbildung 8.1: Das System INQUERY im OMT-Datenmodell

Zu einer INQUERY Datenbank gehört ein Name und eine Stoppwortliste, die beim Parsing zur Stoppwortelimitierung (vgl. Kapitel 2) genutzt wird. Jede Datenbank enthält ein oder mehrere Dokumente.
Alle diese Dokumente besitzen denselben Dokumententyp. Der Dokumententyp ist hierbei eine Aggregation aus einem oder mehreren Feldtypen, die Namen, wie z.B. "text" oder "title", tragen und in der INQUERY Schnittstelle für Anwendungsprogrammierung (vgl. Abschnitt 5.2.3 durch ganzzahlige Bezeichner repräsentiert sind. Ein Dokument eines Dokumententyps besteht aus den den Feldtypen entsprechenden Feldern, in denen sich der Feldinhalt, wie beispielsweise der eigentliche Text oder Titel des Dokuments, befindet. Die Anzahl der Felder (vgl. Kapitel 2) in einem Dokument ergibt sich genau wie die Anzahl von Feldtypen, die einen Dokumententyp bilden, durch die Aggregation und ist ein Attribut des jeweiligen Dokuments bzw. Dokumententyps.
Aus einer Anfrage an das INQUERY System resultiert eine Rangliste, in der sich eine Zahl von relevant erscheinenden Dokumenten befindet. Jedem Dokument wird dabei ein Erfüllungsgrad zugeordnet, der die Position des Dokuments in der Rangliste bestimmt.
Nach der Suche ist es möglich, die Position von zuvor bestimmten Suchtermen in den Feldern der Ergebnisdokumente auszumachen. Die Position besteht dabei aus zwei ganzen Zahlen, die die Länge des Suchterms sowie seinen Anfang in der Zeichenkette des Feldes angeben.
Dieses Datenmodell stellt eine von vielen möglichen Interpretationen der mit dem INQUERY System vorliegenden Daten dar. Naturgemäß erfaßt es nicht alle Zusammenhänge des Systems. Es bietet jedoch Unterstützung dabei, die Funktionen und Datentypen der Schnittstelle für Anwendungsprogrammierung zu verstehen und in TL eine entsprechende Typisierung vorzunehmen.

8.3 Besondere Anforderungen bei der Anbindung

Bei der Anbindung externer Bibliotheken an das Tycoon System müssen einige systemimmanente Besonderheiten beachtet werden. So ist es nicht möglich, Funktionen, die strukturierte Werte als Parameter erhalten oder zurückgeben, direkt an das System anzubinden. Die unterschiedlichen Speicherbereiche, auf die interne und externe Funktionen des Tycoon Systems zugreifen, stellen ebenfalls ein Problem dar (vgl. Abschnitt 7.1.2).

8.3.1 Strukturierte Werte als Parameter externer Funktionen

Die TVM läßt die Anbindung externer Funktionen, die strukturierte Werte als Parameter besitzen, nicht zu. Daher ist es im Rahmen der Anbindung des Systems INQUERY an Tycoon notwendig, derartige Parameter explizit als TL-Tupel aufzubereiten.
Zur Verdeutlichung werden die Funktion *check* und ihr Pendant auf der INQUERY Seite, *inq_check_query*, betrachtet. *inq_check_query* erhält einen Zeiger auf eine Datenbankinformation und eine Zeichenkette, die eine ungeparste Anfrage repräsentiert, als Parameter. Ihr Ergebnis ist ein Zeiger auf eine Struktur vom Typ *query_info_t*:

```
query_info_t *inq_check_query (dbinfo *db, Char_t *qstring)
```

```
typedef struct
{
  Int_t             error_code;      /* zero for no error */
  Int_t             error_offset;    /* only meaningfull if error_code != 0 */
  Char_t            *parsed_query;   /* final parsed query */
  Int_t             num_terms;       /* total number of terms in query */
  Int_t             num_stops;       /* number of stop words in query */
  Int_t             num_not_found;   /* number of non-indexed words */
  Int_t             num_xformers;    /* number of identified transformers */
} query_info_t;
```

Die entsprechende TL-Funktion soll folgende Signatur besitzen:

check(db :dbinfo.T qstring :String) :Info

Hierbei ist der Typ *Info* folgendermaßen implementiert:

```
Let Info = Tuple
             errorCode :Int
             errorOffset :Int
             parsedQuery :T
             numTerms :Int
             numStops :Int
             numNotFound :Int
             numXFormers :Int
           end
```

das TL-Konstrukt *bind* (vgl. Abschnitt 7.2.3) wird verwendet, um die Funktion *inq_check_query* an Tycoon anzubinden.

```
let checkCall =
bind(:Fun(:word.T :String) :word.T lib "inq_check_query" "wsw")
```

Ein Aufruf von *checkCall* liefert dem Tycoon System jedoch nur einen Zeiger in den von C verwalteten Speicherbereich. Die C-Struktur, auf die dieser Zeiger weist, kann in TL auf zwei Arten zugänglich gemacht werden.
Zum einen bietet das Tycoon System die Schnittstelle *WordOp.ti* zur Interpretation von Maschinenwörtern, wie Bitmustern oder Adressen, an. Diese stellt Funktionen zur Verfügung, mit denen der Inhalt des Speicherbereichs, auf den ein C-Zeiger zeigt, bitweise interpretiert und in TL-Werte umgesetzt werden kann. Diese maschinennahe Art des Zugriffs besitzt jedoch einen Nachteil. Implementationen, die auf bitweisem Zugriff basieren, sind von der zugrundeliegenden Rechnerarchitektur abhängig, so daß sie nur eingeschränkt portabel sind.

Aufgrund dieser eingeschränkten Portabilität wird für die Anbindung des Systems INQUERY ein anderer Weg gewählt. Die C-Strukturen werden bereits auf C-Seite in ihre Komponenten aufgespalten, einzeln an das Tycoon System übergeben und dort zu einem Tupelwert zusammengesetzt.
Für das obige Beispiel wird also wie folgt verfahren. Zunächst wird der aus dem Aufruf der Funktion *checkCall* resultierende Zeiger in einem TL-Wert abgelegt.

let *ptr = checkCall(db "all about Information Retrieval")*

Auf C-Seite werden Funktionen definiert, die den Zugriff auf die einzelnen Komponenten der Struktur, auf die der Zeiger weist, erlauben.

```
Int_t get_error_code(query_info_t *info)
{
  return info->error_code;
}


Int_t get_error_offset(query_info_t *info)
{
  return info->error_offset;
}

...
```

Die beiden Funktionen *get_error_code* und *get_error_offset* dienen hier als Beispiel. Die Funktionen, die die weiteren Komponenten einer Struktur vom Typ *query_info_t* liefern, sind entsprechend aufgebaut. Die C-Funktionen wurden wiederum an das Tycoon System gebunden.

let *checkCall2 =*
bind(*:**Fun***(*:word.T) :Int lib "get_error_code" "wi")*

let *checkCall3 =*
bind(*:**Fun***(*:word.T) :Int lib "get_error_offset" "wi")*

...

Mit Hilfe dieser Bindungen wird die TL-Funktion *check* implementiert.

```
let check(db :dbinfo.T  qstring :String) :Info =
  begin
    let ptr = checkCall(db qstring)
    let result =  tuple
                    checkCall2(ptr)
                    checkCall3(ptr)
                    ...
                  end
    word.free(ptr) (* free C-pointer *)
    result
  end
```

Ein Aufruf von *check* liefert somit einen Tupelwert im Tycoon Speicher zurück, dessen Komponenten den Komponenten der zugrundeliegenden C-Struktur entsprechen.

8.3.2 Persistenz in unterschiedlichen Speicherbereichen

Das Tycoon System bietet die Möglichkeit, Objekte, die in den systemeigenen Programmiersprachen, wie z.B. TL, definiert worden sind, persistent im Speicher abzulegen. Werte, die während der Nutzung externer Bibliotheken entstehen, werden jedoch nicht im Speicherbereich des Tycoon Systems gespeichert und sind daher flüchtig.
Bestehen innerhalb von Tycoon Werte, die Zeiger auf den externen Speicherbereich enthalten, so werden diese bei einer Stabilisierung des Systemzustandes ebenso persistent wie alle anderen Tycoon Werte. Wird das Tycoon System heruntergefahren, so geht der externe Wert, der durch den persistenten Zeiger adressiert wurde, verloren. Der Zeiger wird in jeder weiteren Sitzung mit dem Tycoon System einen Bereich adressieren, der nicht mehr den ursprünglichen Wert enthält.
Auch bei der Anbindung des INQUERY Systems werden Funktionen implementiert, die Tycoon Zeiger auf den externen Speicherbereich zum Ergebnis haben. So ist eine geöffnete INQUERY Datenbank durch einen Zeiger repräsentiert. Hier gilt jedoch die Prämisse, daß nur mit einer INQUERY Datenbank zur Zeit gearbeitet werden kann. Dadurch muß eine Dokumentendatenbank vor dem Zugriff geöffnet und anschließend wieder geschlossen werden. Dieses Öffnen und Schließen ist in jedem Anwendungsprogramm für INQUERY erforderlich. Hier läßt sich auch das Anlegen eines neuen Zeigers vor jedem Öffnen einer Datenbank, bzw. direkt nach jedem Hochfahren des Tycoon Systems integrieren. Wünschenswert wäre es jedoch, den Anwendungsprogrammierer von derartigen Aufgaben der Systemverwaltung zu entlasten.
Das Tycoon System stellt einen Mechanismus bereit, der das Anlegen und Löschen externer Werte protokolliert und diese beim nächsten Systemstart erneut generiert (vgl. Abschnitt 7.1.2). Im Rahmen der vorliegenden Arbeit wird diese Möglichkeit nicht betrachtet. Die Verwendung des beschriebenen Mechanismus kann jedoch der Gegenstand zukünftiger praktischer Arbeiten sein, zumal er für andere INQUERY Daten, wie Dokumente oder Textstellen,

an denen Suchterme gefunden wurden, zum Einsatz kommen könnte.
Ein weiterer Ansatz ist es, Daten wie Dokumente nicht als Zeiger auf den externen Speicher abzulegen, sondern die dort enthaltenen Werte direkt als TL-Werte zu übernehmen. Dieses Vorgehen würde dem Datenmodell des Systems INQUERY (vgl. Abschnitt 8.2) eher gerecht werden und den intuitiven Umgang mit dem System innerhalb von Tycoon erleichtern.
Ein Dokument beispielsweise ist momentan als Zeiger auf den externen Speicher definiert.

```
Let Doc = word.T (* pointer *)
```

Nach dem INQUERY Datenmodell ist auch die folgende Definition eines Dokumententyps geeignet:

```
Let Doc =  Tuple
             numberOfFields :Int
             fields :list.T(Field)
           end
```

Die Tatsache, daß ein Dokument durch eine Aggregation von Feldern entsteht, ist hier durch eine Liste dieser Felder dargestellt. Zusätzlich enthält das Dokument eine Angabe darüber, wieviele Felder es enthält.
Diese Art der Implementation bedeutet jedoch einen zusätzlichen Aufwand. So müßten beispielsweise für Dokumente alle Felder im voraus aus dem C-Speicherbereich des Systems INQUERY in den persistenten Tycoon Speicher geladen werden, obwohl zu diesem Zeitpunkt nicht klar ist, ob sie benötigt werden. Ob der Verständnisvorteil, der durch die modellnahe Definition von Typen entsteht, diesen Aufwand rechtfertigt, muß Gegenstand zukünftiger Betrachtungen bleiben.

8.4 TL-Typisierung in INQUERY

Die Typisierung der Sprache TL wird bei der Anbindung von INQUERY genutzt, um zwischen Datentypen, die in C einheitlich repräsentiert sind, genauer zu differenzieren. Dadurch wird der Definitionsbereich von Funktionen so eingeschränkt, daß eine fehlerhafte Nutzung zur Übersetzungszeit entdeckt werden kann. Das Typsystem unterstützt somit statisch den korrekten Ablauf von Programmen.
Bei der Anbindung von INQUERY wurden beispielsweise die Anfragen an eine Datenbank, die in der Programmierschnittstelle von INQUERY als einfache Zeichenketten definiert sind, so typisiert, daß bei einer Nutzung im Tycoon System generell zwischen geparsten und ungeparsten Anfragen unterschieden wird:

```
T <:String
(* A parsed query *)
```

Eine ungeparste Anfrage kann eine beliebige Zeichenkette sein, und auch der Typ T einer geparsten Anfrage ist Subtyp vom Typ *String*. Durch die Subtypisierung wird jedoch erreicht, daß Funktionen, die einen Wert vom Typ T als Parameter erwarten, ausschließlich Werte

dieses Typs oder eines Subtyps von T akzeptieren [Mat93]. Die Übergabe einer einfachen Zeichenkette anstelle einer geparsten Anfrage würde somit zur Übersetzungszeit zu einem Typfehler führen. So läßt sicherstellen, daß z.B. einer Funktion, die eine geparste Anfrage auswerten soll, auch nur eine solche übergeben wird:

evalParsed(db :dbinfo.T parsedQuery :T) :Results
(Evaluate the parsed query and return the results in a belieflist. *)*

Eine geparste Anfrage, also ein Wert vom Typ T, kann nur durch den Aufruf entsprechender Funktionen generiert werden.

getParsed(db :dbinfo.T queryString :String) :T
(Process a parsed query string *)*

Durch Typisierung werden somit logische Fehler in der Anwendungsprogrammierung unterbunden. Von der Implementation der jeweiligen Typen wird dabei abstrahiert.

8.5 Möglichkeiten der erweiterten Nutzung von INQUERY innerhalb von Tycoon

Das Tycoon System ist eine Systementwicklungsumgebung mit der Möglichkeit, nicht nur Deklarationen, sondern auch beliebige konkrete Werte persistent abzulegen . Sowohl Schnittstellen, die in systeminternen Programmiersprachen implementiert sind, als auch externe Bibliotheken können auf orthogonale Weise genutzt werden und interagieren (vgl. Abschnitt 7.2). Im Information Retrieval wird eine Vielzahl an Informationen verwaltet. Verschiedene Benutzer haben auf ein Softwaresystem wie INQUERY Zugriff. Der Einsatz von INQUERY innerhalb des Tycoon Systems bietet hier erweiterte Möglichkeiten der Anwendungsprogrammierung.

8.5.1 Persistenz im Information Retrieval

Die Persistenz des Tycoon Systems ist im Information Retrieval sowohl für Anfragen an das System als auch für zurückgelieferte Ergebnisse sinnvoll nutzbar. Parameter und Ergebnisse eines Information Retrieval Prozesses können zwar auch ohne Anbindung an Tycoon von INQUERY protokolliert werden. Dieses bleibt jedoch dem Benutzer unter Verwendung des jeweiligen Dateisystems überlassen. Die Verwaltung von Anfrage- und Ergebnisdaten läßt sich in Tycoon so automatisieren, daß dem Benutzer die zugrundeliegende Systemstruktur verborgen bleibt.
Unter diesen Voraussetzungen ist es möglich, die Funktionalität des INQUERY Systems zu erweitern. Einmal gestellte Anfragen könnten bespielsweise für verschiedene Benutzer in verschiedenen Tycoon Speicherbereichen abgelegt werden. Dieses begünstigt den Vorgang der Informationsfilterung [BC92], in dem regelmäßig von einem Benutzerprofil abhängige Anfragen an ein Informationssystem gestellt werden.

Weiterhin ist ein Katalog von Standardanfragen an eine Datenbank denkbar, der zum einen der Auswertung von Systemveränderungen dient, aber zum anderen auch Benutzern als Alternative zur Eingabe eigener Anfragen zur Verfügung gestellt werden kann.
Eine persistente Ablage von Ergebnissen eines Information Retrieval Prozesses gibt Benutzern die Möglichkeit, sich eine Art elektronische Bibliothek mit für sie relevanten Dokumenten anzulegen. Mit Hilfe von TL-Datenstrukturen läßt sich diese zusätzlich sinnvoll, beispielsweise nach Themengebieten oder Anfragedaten, ordnen.

8.5.2 Interoperabilität im Information Retrieval

Das System Tycoon vereinigt eine Vielzahl von Bibliotheken (vgl. Abschnitt 7.2.4), deren Funktionen, Typen usw. beliebig für neue Implementationen genutzt werden können. Die Verbindung eingebundener, externer Bibliotheken untereinander wird durch die Orthogonalität der Sprache TL vereinfacht.
Auch für das INQUERY System lassen sich viele der Tycoon Bibliotheken nutzen. So werden verschiedene Strukturen zur Verwaltung von Massendaten, wie z.B. Hashtabellen, Bäume oder Listen, angeboten. Die Implementation ergonomischer Benutzerschnittstellen mit Hilfe von in Tycoon integrierten Fenstersystemen ist ebenso vorstellbar wie die Weiterverarbeitung von Dokumenten über einen angebundenen Texteditor.

9. Zusammenfassung

Information Retrieval ist als automatisierte Auswertung von Anfragen an in elektronischer Form vorliegende Textbibliotheken definiert. Die natürlichsprachlichen Inhalte sowohl der Anfragen als auch der in den Bibliotheken enthaltenen Texte sind nicht präzise formal auf dem Rechner repräsentierbar. Es wurden jedoch verschiedene Konzepte, wie z.B. Stoppwortelimitierung, Stammformreduktion, Begriffserkennung oder Feldsuche, die sich mit bestimmten Eigenheiten der natürlichen Sprache und ihrer Semantik auseinandersetzen, entwickelt, um der Ausdruckskraft natürlicher Sprache in Rechnerrepräsentationen möglichst nahe zu kommen. Viele von ihnen finden sich in Gesamtmodellen, die vorgeben, auf welche Weise die Repräsentation und Auswertung von Texten als Ganzes zu erfolgen hat, wieder. Das Ziel hierbei ist, Information Retrieval in einer Form zu betreiben, die der intuitiven menschlichen Auffassung dieses Prozesses zumindest annäherungsweise entspricht (vgl. Kapitel 2).
Die Modelle des Information Retrieval lassen sich in zwei Klassen unterteilen. Modelle mit exakter Übereinstimmung legen für jedes Dokument einer Textdatenbank fest, ob es für eine Anfrage relevant ist oder nicht. Modelle mit bestmöglicher Übereinstimmung haben für jedes Dokument einen Grad zum Ergebnis, zu dem das Dokument einer Anfrage genügt. Die feine Abstufung, die diese Modelle liefern, entspricht dem intuitiven Blick auf das Information Retrieval eher als die zweiwertige Logik der Modelle mit exakter Übereinstimmung. Modelle mit bestmöglicher Übereinstimmung lassen sich nach der Art der Repräsentation und Auswertung von Texten und Anfragen weiter aufgliedern in Vektorraum- und probabilistische Modelle (vgl. Kapitel 3).
Um die Qualität der verschiedenen Modelle beurteilen zu können, werden für das Information Retrieval die Bewertungskriterien des Rücklaufs und der Genauigkeit definiert. Der Rücklauf gibt die Fähigkeit eines Systems wieder, nützliche Dokumente zu finden, die Genauigkeit stellt dar, wie konsequent nutzloses Material ausgeschlossen wird (vgl. Kapitel 4).
Verschiedene Information Retrieval Systeme kommen in der Praxis zum Einsatz. Nachdem hier zunächst Systemen, die nach Modellen der exakten Übereinstimmung arbeiten, der Vorzug gegeben wurde, werden sie gegenwärtig von Systemen, die dem Prinzip der bestmöglichen Übereinstimmung folgen, abgelöst. Mit dem System INQUERY, das auf dem probabilistischen Modell der Inferenznetze basiert, und dem System WAIS, das auf einem Vektorraummodell beruht, werden in der vorliegenden Arbeit zwei dieser neueren Systeme skizziert (vgl. Kapitel 5 und 6).

Das System INQUERY wird in die Systementwicklungsumgebung Tycoon integriert. Dabei werden spezielle Eigenschaften von Tycoon und der systemeigenen höheren Programmiersprache TL eingebracht (vgl. Kapitel 7). Unter Verwendung des TL-Typsystems können korrekte Abläufe innerhalb des Systems INQUERY statisch zugesichert werden. Zusätzlich bietet das Tycoon System Persistenz für alle Elemente der Sprache TL und eine Interoperabilität zwischen unterschiedlichen generischen Diensten. Beides schafft Raum für erweiterte, benutzernahe Anwendungen innerhalb des Information Retrieval (vgl. Kapitel 8).

10. Ausblick

Das Gebiet des Information Retrieval befindet sich nach wie vor in intensiver Weiterentwicklung. Hierbei scheint jedoch eine neue Phase begonnen zu haben. Die zugrundeliegenden linguistischen Voraussetzungen zur Darstellung von natürlicher Sprache auf dem Rechner sind zwar immer noch unvollständig erarbeitet. Doch die Randbedingungen des Information Retrieval haben sich in einer Richtung verändert, die wissenschaftliche Untersuchungen abseits von linguistischen Konzepten notwendig macht.
Große, offene Softwaresysteme ermöglichen neue Anwendungen innerhalb des Information Retrieval. Der Benutzer eines Retrieval Systems wird neben der Möglichkeit für einfache Anfragen mit Relevanzrückkopplung schon bald eine erweiterte Funktionalität erwarten: Die Erstellung eines Benutzerprofils, die Speicherung von Ergebnisdaten in Benutzerbibliotheken, die regelmäßige Wiederholung von Anfragen an schnell veränderliche Datenbanken, Querverweise auf in der Vergangenheit gestellte Anfragen oder eine direkte Weiterverarbeitung von Ergebnissen sind Fragmente eines vorstellbaren Szenarios.
Ein weiterer Forschungsschwerpunkt sollte auf den Daten liegen, die zum Information Retrieval herangezogen werden. Zum einen ist in den modernen globalen Rechnernetzen neben Textdokumenten auch eine Vielzahl an Bildern, Videos und Audiodokumenten zu finden. Benutzer von Information Retrieval Systemen sollten beim Zugriff auf derartige Daten dieselbe Unterstützung erfahren wie beim Zugriff auf Texte. Zum anderen ist die Kombination von unscharfen Daten im Information Retrieval mit strukturierten Daten aus herkömmlichen Datenbanken, wie Personendaten o.ä., zu betrachten. Hier stellt sich auch die Frage, ob nicht die Anwendung der Konzepte des Information Retrieval auf strukturierte Daten sinnvoll sein kann. Als Beispiel kann hier ein Personalchef dienen, der aus seiner Datenbank alle Mitarbeiter "um die vierzig" herausfiltern möchte. Obwohl die Geburtsdaten der Mitarbeiter eindeutig sind, können auch auf ihrer Basis Mengen gebildet werden, deren Elemente nicht exakt bestimmbar sind. Konzepte, die in Information Retrieval Modellen mit bestmöglicher Übereinstimmung definiert wurden, könnten hier ein neues Anwendungsfeld finden.

A. Distanzmaße für Vektorraummodelle des Information Retrieval

In Information Retrieval Systemen, die auf Vektorraum-Modellen basieren, werden Anfragen und Dokumente als Vektoren repräsentiert. Bei der Auswertung von Anfragen an diese Systeme, wird die Distanz zwischen dem Anfragevektor und den verschiedenen Dokumentenvektoren ermittelt. Hierzu werden in verschiedenen Systemen verschiedene Distanzmaße verwendet.
Ein in Information Retrieval Systemen oft benutztes Maß für k-dimensionale Vektorräume ist der Cosinus des Winkels α zwischen dem Anfragevektor $\vec{a}$ und den Dokumentenvektoren $\vec{d}$ [TC92]. Je größer dabei der Cosinus, desto besser erfüllt ein Dokument die Anfrage. Der Cosinus berechnet sich wie folgt [BS95]:

$$\cos\alpha = \frac{\vec{a} \cdot \vec{d}}{\| \vec{a} \| \cdot \| \vec{d} \|}$$

Dabei ist $\vec{a} \cdot \vec{d}$ das Skalarprodukt von $\vec{a}$ und $\vec{d}$ mit

$$\vec{a} \cdot \vec{d} = a_1 d_1 + a_2 d_2 + \cdots + a_k d_k = \sum_{j=1}^{k} a_j d_j$$

$\| \vec{a} \|$ bzw. $\| \vec{d} \|$ sind die euklidischen Normen von $\vec{a}$ und $\vec{d}$ mit

$$\| \vec{a} \| = \sqrt{{a_1}^2 + {a_2}^2 + \cdots {a_k}^2} = \sqrt{\sum_{j=1}^{k} {a_j}^2}$$

und

$$\| \vec{a} \| = \sqrt{{d_1}^2 + {d_2}^2 + \cdots {d_k}^2} = \sqrt{\sum_{j=1}^{k} {d_j}^2}$$

Somit läßt sich der Cosinus des Winkels zwischen $\vec{a}$ und $\vec{d}$ mit Hilfe der Komponenten dieser beiden Vektoren wie folgt berechnen:

$$\cos\alpha = \frac{\sum_{j=1}^{k} a_j d_j}{\sqrt{\sum_{j=1}^{k} {a_j}^2 \sum_{j=1}^{k} {d_j}^2}}$$

Dieses Maß ist für das Information Retrieval jedoch nur begrenzt geeignet, da es die Länge der jeweiligen Vektoren außer acht läßt. Dadurch werden unter Umständen Dokumente, die nur minimal von den Erfordernissen der Anfrage abweichen, schlechter bewertet als solche, die der Anfrage nur in sehr geringem Maße genügen.
Zur Verdeutlichung betrachte man folgendes Beispiel: Sei $\vec{a}$ ein Anfragevektor mit folgenden Eigenschaften:

$$\vec{a} = (0,8\ ,\ 0,8\ ,\ 0,8)$$

In der Dokumentensammlung gibt es drei Dokumente, die folgendermaßen repräsentiert seien:

$$\vec{d_1} = (0,8\ ,\ 0,8\ ,\ 0,8)$$

$$\vec{d_2} = (0,8\ ,\ 0,7\ ,\ 0,7)$$

$$\vec{d_3} = (0,1\ ,\ 0,1\ ,\ 0,1)$$

Sei $\cos\alpha_i$ der Cosinus des Winkels zwischen dem Anfragevektor $\vec{a}$ und dem jeweiligen Dokumentenvektor $\vec{d_i}$. So ergibt sich für $\vec{d_1}$:

$$\cos\alpha_1 = 1$$

Dieses Ergebnis entspricht der intuitiven Auffassung von Information Retrieval. Der Anfragevektor und der Dokumentenvektor beschreiben exakt die gleichen Eigenschaften, so daß die höchste Bewertung mit 1 erwartet werden konnte.
Betrachtet man das zweite Dokument, so erhält man:

$$\cos\alpha_2 \approx 0,9979$$

Im Vergleich zu dem Ergebnis des ersten Dokuments entspricht aus dieses Ergebnis den Erwartungen. Schließlich weicht es bei der zweiten und dritten Eigenschaft leicht von der Anfrage ab, so daß ein etwas schlechteres Ergebnis nicht ungewöhnlich erscheint.
Bei der Betrachtung des dritten Dokuments ergibt sich jedoch folgender Wert:

$$\cos\alpha_3 = 1$$

Dieses Ergebnis ist konnte nicht erwartet werden, denn das dritte Dokument erfüllt alle drei geforderten Eigenschaften nur in sehr geringem Maße. Dennoch wird es ebenso gut wie das erste und sogar besser als das zweite Dokument bewertet.
Der Grund hierfür liegt darin, daß die Vektoren des ersten und des dritten Dokuments sowie der Anfragevektor sich im Vektorraum überdecken. Der Winkel zwischen ihnen ist jeweils 0 und der Cosinus dieses Winkels damit 1.
Aus diesem Fallbeispiel muß man schließen, daß ein Distanzmaß für Vektorraum-Modelle im Information Retrieval nur dann sinnvoll eingesetzt werden kann, wenn sowohl der Winkel zwischen den Vektoren als auch deren Länge betrachtet werden. Dieses Voraussetzungen erfüllt das oben bereits definierte Skalarprodukt, das sich ebenfalls aus den Vektorkomponenten bilden läßt [BS95]:

$$\vec{a} \cdot \vec{d} = a_1 d_1 + a_2 d_2 + \cdots + a_k d_k = \sum_{j=1}^{k} a_j d_j$$

Die folgende Tabelle zeigt, daß dieses Distanzmaß für das Information Retrieval besser geeignet ist.

Dokumenten-vektor im Vergleich zum Anfragevektor (0,8 , 0,8 , 0,8)	Cosinus d. Winkels zwischen Anfrage- u. Dokumenten-vektor	entsprechender Rang	Skalarprodukt von Anfrage- u. Dokumenten-vektor	entsprechender Rang
(0,8 , 0,8 , 0,8)	1	1	1,92	1
(0,8 , 0,7 , 0,7)	0,9979	3	1,76	2
(0,1 , 0,1 , 0,1)	1	1	0,24	3

B. Operatoren der INQUERY-Anfragesprache

Die folgende Tabelle beschreibt die Operatoren der INQUERY-Anfragesprache. Dabei gibt es zwei Typen von Parametern:

- Terme T_i sind entweder Wörter der natürlichen Sprache oder wiederum Operatoren.
- Gewichtungen W_i sind reelle Zahlen mit $0 \leq W_i \leq 1$.

Beim geordneten Abstandsoperator und beim ungeordneten Fensteroperator steht der Platzhalter N für eine beliebige natürliche Zahl.

OPERATOR	AKTION
#sum(T1 ... Tn) z.B.: #sum(retrieval filtering #not(images))	**Summenoperator**: Die Terme, die dem Operator übergeben werden, haben alle den gleichen Einfluß auf das Ergebnis. Ihr Erfüllungsgrade werden gemittelt, um den Erfüllungsgrad des #sum-Knotens zu berechnen.
#wsum(Ws W1 T1 ... Wn Tn) z.B.: #wsum(0.9 0.9 retrieval 0.6 filtering 0.1 #not(images))	**gewichteter Summenoperator**: Die Terme, die dem Operator übergeben werden, haben unterschiedlichen Einfluß auf das Ergebnis. Sie werden mit den ihnen zugewiesenen Gewichtungen bewertet. Mit Ws wird angegeben, zu welchem Grad der Operator die gesamte Anfrage beeinflußt.
#N(T1 ... Tn) oder #odN(T1 ... Tn) z.B.: #2(House Representatives)	**geordneter Abstandsoperator**: Wird erkannt, wenn alle Argumente in der angegebenen Reihenfolge gefunden werden, so daß nicht mehr als (N - 1) Wörter zwischen zwei benachbarten Argumenten stehen. Z.B. wird #3(A B) von "A B", "A c B" und "A c c B" erfüllt, nicht aber von "A c c c B".

OPERATOR	AKTION
#and(T1 ... Tn) z.B.: #and(information retrieval system)	**Und-Operator**: Je mehr Argumente des Operators in einem Dokument gefunden werden, desto höher ist der Erfüllungsgrad des Dokuments. Dieser Operator entspricht *nicht* dem booleschen Und-Operator!
#band(T1 ... Tn) z.B.: #band(information retrieval)	**boolescher Und-Operator**: Alle Argumente des Operators müssen in einem Dokument gefunden werden, damit dieser Operator zum Erfüllungsgrad des Dokuments beiträgt.
#or(T1 ... Tn) z.B.: #or(#band(information retrieval) #band(fulltext search))	**Oder-Operator**: Eines der Argumente des Operators muß in einem Dokument gefunden werden, damit dieser Operator zum Erfüllungsgrad des Dokuments beiträgt.
#not(T1) z.B.: #not(#or(Germany Europe))	**Negationsoperator**: Dokumente, die das Argument des Operators nicht enthalten, werden höher bewertet als andere.
#uwN(T1 ... Tn) z.B.: #uw50(precision recall #not(pertinence))	**ungeordneter Fensteroperator**: Die Argumente des Operators müssen in willkürlicher Reihenfolge innerhalb eines Fensters von N Wörtern gefunden werden, damit dieser Operator zum Erfüllungsgrad eines Dokuments beiträgt.
#phrase(T1 ... Tn) z.B.: #phrase(fulltext search)	**Phrasenoperator**: Der Wert ist die Funktion der Erfüllungsgrade, die durch Anwendung der Operatoren #3 und #sum entstehen. Die Intention dabei ist, Phrasen zu erkennen, wenn sie vorhanden sind, und einzelne Wörter zu erkennen, wenn keine Phrasen existieren.
#passageN(T1 ... Tn) #passage50(precision recall #not(pertinence))	**Abschnittsoperator**: Der Operator sucht nach einem Fenster von N Wörtern, in dem seine Argumente erscheinen. Das Dokument wird nach seinem besten Abschnitt bewertet.
#syn z.B.: #syn(FRG Germany)	**Synonymoperator**: Die Argumente werden als Synonyme betrachtet.

OPERATOR	AKTION
#max z.B.: #max(retrieval #band(information retrieval))	**Maximumsoperator**: Der Wert ist das Maximum der Erfüllungsgrade der Argumente.
#+ T1 z.B.: #+#band(information retrieval) filtering search	**Operator zur stärkeren Gewichtung**: Der Einfluß des Arguments in Bezug auf den Rest der Anfrage wird erhöht.
#- T1 z.B.: #band(information retrieval) search #-image	**Operator zur schwächeren Gewichtung**: Der Einfluß des Arguments in Bezug auf den Rest der Anfrage wird verringert.
#lit(T1 ... Tn) z.B.: #lit(INQUERY WAIS Tycoon)	**Literaloperator**: Der Operator erhält die ursprüngliche Form seiner Argumente. Stoppwortelimitierung oder Reduzierung auf den Wortstamm werden nicht ausgeführt. Großschreibung bleibt erhalten.

C. Tycoon Schnittstellen für das System INQUERY

C.1 *Dbinfo.ti*: Zugang zu INQUERY Datenbanken

```
interface Dbinfo
(* System:  Inquery
   File:    Dbinfo.ti
   Author:  Ulrike Steffens
   Date:    10-Oct-1995
   Purpose: Operations to administer the databases of an INQUERY
            information retrieval system
*)

import
   list
   word
   :DbinfoHidden

export

  T <:Ok

  Let Representation = Tuple db :word.T  isOpen :Bool end
  (* External Representation of Dbinfo.T *)

  getRepresentation(t :T) :Representation

  error :Exception end
```

new(dbName, stopWordFile :String
defaultBeliefValue, termFrequency :Real) :T
(Create a new dbinfo structure given the name of the*
database used, the name of a stopword file,
the name of a relevance file, the default belief value
*and the default term frequency value *)*

free(db :T) **:Ok**
(Free a dbinfo structure *)*

opendb(db :T) **:Ok**
(Open a database for use by the INQUERY retrieval system.*
*Currently only one database can be open at a time *)*

closedb(db :T) **:Ok**
(Close a database, should be called before opening a new*
*database and before exiting from an INQUERY session *)*

getDatabaseList(directoryName :String) :list.T(String)
(Return a list of databases available for INQUERY in a*
*certain directory *)*

setName(db :T dbName :String) **:Ok**
(Set the name field of a dbinfo structure *)*

getName(db :T) :String
(Get the name field of a dbinfo structure *)*

setStopWordFile(db :T stopWordFile :String) **:Ok**
(Set the stopword name field of a dbinfo structure *)*

setTermFrequency(db :T termFrequency :Real) **:Ok**
(Set the term frequency field of a dbinfo structure *)*

setDefaultBeliefValue(db :T defaultBeliefValue :Real) **:Ok**
(Set the default belief value of a dbinfo strucure *)*

setStopping(db :T onOff :Bool) **:Ok**
(Set stopword processing on or off *)*

```
  setStemming(db :T onOff :Bool) :Ok
  (* Set word stemming on or off *)

  getStopping(db :T) :Bool
  (* Get stopword processing flag *)

  getStemming(db :T) :Bool
  (* Get stemming processing flag *)

end;
```

C.2 *Query.ti*: Parsing und Auswertung von Anfragen

```
interface Query
(* System:  Inquery
   File:       Query.ti
   Author:  Ulrike Steffens
   Date:     10-Oct-1995
   Purpose: Operations to generate and evaluate queries for
            the use by an INQUERY information retrieval system
*)

import
   list
   dbinfo

export

   T <:String
   (* A parsed Query *)

   Info <:Ok
   (* Stores query term information containing the parsed
   query string and a list giving information about each term
   in the query *)

   BeliefElement <:Ok
   (* Information about one retrieved element, a part of :Result.
   May be accessed by getElementDocID and getElementBelief *)
```

*Results <:***Ok**
(Information about all retrieved elements. May be accessed by*
*getResultDefaultBelief, getResultDocCount and getResultBeliefList *)*

*error :***Exception end**

check(db :dbinfo.T qstring :String) :Info
(Parse a query string storing information about each of the*
recognized term. The resulting information can be accessed by
the following getInfo-functions. Raise exception if empty
*qstring is checked *)*

getInfoErrorCode(info :Info) :String
(Return possible error code of checking process *)*

getInfoErrorOffset(info :Info) :Int
(Return offset of error in query. Only meaningful if*
*error code is not "no error" *)*

getInfoParsed(info :Info) :T
(Return the final parsed query resulting of checking process *)*

getInfoNumOfTerms(info :Info) :Int
(Return the total number of terms in the query *)*

getInfoNumOfStopWords(info :Info) :Int
(Return the number of stop words in the query *)*

getInfoNumOfNotFound(info :Info) :Int
(Return the number of non-indexed words in the query *)*

getInfoNumOfTransformers(info :Info) :Int
(Return the number of transformers identified during*
*checking process *)*

getParsed(db :dbinfo.T queryString :String) :T
(Process a parsed query string. Processing is the same*
as that done by check, except that a history of the
processing steps is not retained in an :Info. Raise exception
*if empty qstring is parsed *)*

initTransformer(transformerName :String onOff :Bool) ***:Ok***
(Turn a named query transformer on or off. Available right now are: "stop_phrases", "location", "hyphens_caps", "company_tags" and "retain_capitalized_stopwords". For more infromation about transformers see documentation *)*

initAllTransformers(onOff :Bool) ***:Ok***
(Turn on or off all query transformers *)*

setStemming(onOff :Bool) ***:Ok***
(Set stemming flag during query processing. Query stemming is on by default *)*

getStemming() :Bool
(Get the current value of the query stemming flag *)*

eval(db :dbinfo.T queryString :String) :Results
(Evaluate the unparsed query and return the results in a belief list. A feedback function may display the progress of evaluation, but can be a function doing nothing *)*

evalTopN(db :dbinfo.T queryString :String
maxRetrievedDocs :Int) :Results
(Evaluate the unparsed query returning a belief list containing no greater than maxDocs documents, apart from that identical to function "eval" *)*

evalParsed(db :dbinfo.T parsedQuery :T) :Results
(Evaluate the parsed query and return the results in a belieflist. A feedback function may display the progress of evaluation *)*

evalParsedTopN(db :dbinfo.T parsedQuery :T
maxRetrievedDocs :Int) :Results
(Evaluate the parsed query returning a belief list containing no greater than maxDocs documents, apart from that identical to function "evalParsed" *)*

getResultDefaultBeliefValue(results :Results) :Real
(Return the default belief value of a set of retrieved elements *)*

```
  getResultNumOfRetrievedDocs(results :Results) :Int
  (* Return the number of retrieved elements *)

  getResultBeliefList(results :Results) :list.T(BeliefElement)
  (* Return a list of all retrieved elements *)

  getElementDocID(beliefElement :BeliefElement) :Int
  (* Return the document id of a retrieved element *)

  getElementBeliefValue(beliefElement :BeliefElement) :Real
  (* Return the belief value of a retrieved element *)

end;
```

C.3 *Docs.ti*: Zugang zu Dokumenten

```
interface Docs
(* System:  Inquery
   File:    Docs.ti
   Author:  Ulrike Steffens
   Date:    10-Oct-1995
   Purpose: Operations to administer documents retrieved by the
            INQUERY information retrieval system.
*)

import
   dbinfo
   query

export

   T <:Ok
   (* The actual document consisting of a number of component fields *)

   FieldType <:Ok

   newField(:String) :FieldType
   (* Return a new value of type FieldType in behalf of a certain
   database *)
```

FID :FieldType
FSOURCE :FieldType
FTEXT :FieldType
FTITLE :FieldType
(Determines which component field of a retrieved document is actually worked with. Take into consideration that not all possible combinations of Mode and FieldType actually make sense *)*

*Match <:***Ok**
*MatchesArray <:***Ok**
(Helpful structures to find out where the query has matched with the retrieved document, processed in dependence of a single document component field *)*

*Field <:***Ok**
(One document might consist of several component fields *)*

*error :***Exception end**

get(db :dbinfo.T docID :Int parsedQuery :query.T) :T
(Retrieve document specified by "id" in database. Get certain match information about a specified query string. Don't forget to free the document after use! *)*

*free(doc :T) :***Ok**
(Free the internal buffer that contains document text *)*

getField(doc :T fieldType :FieldType) :Field
(Get component fields of a document *)*

getFieldText(docField :Field) :String
(Get text out of a document field *)*

getNumOfFieldMatches(docField :Field) :Int
(Get the number of matches out of a document field *)*

getFieldMatches(docField :Field) :MatchesArray
(Get matches out of a document field *)*

numOfDocsInCollection(db :dbinfo.T) :Int
(Get number of documents in INQUERY database specified *)*

```
numOfTermPostingsInCollection(db :dbinfo.T term :String) :Int
(* Get number of documents in which a term occurs for the
specified database *)

termFrequencyInCollection(db :dbinfo.T term :String) :Int
(* Get the total number of occurrences of a term in the
specified collection *)

externalToInternalID(db :dbinfo.T exID :String) :Int
(* Convert an external document id to its internal INQUERY equivalent *)

internalToExternalID(db :dbinfo.T inID :Int) :String
(* Convert INQUERY's internal document id to it's external equivalent *)

sortMatchesByPosition(matches :MatchesArray) :Ok
(* Sort a list of query term matches by position (offset) *)

rewindMatchList(matches :MatchesArray) :Ok
(* Rewind the current match pointer to the beginning of the match list *)

getNextMatch(matches :MatchesArray) :Match
(* Get the item from a list of matches *)

getPreviousMatch(matches :MatchesArray) :Match
(* Get the previous match from a list of matches *)

getNumOfMatches(matches :MatchesArray) :Int
(* Get the number of matches out of a MatchesArray *)

getMatchTerm(match :Match) :String
(* Return a term string from the match *)

getMatchOffset(match :Match) :Int
(* Return the value from a match offset field, raise error
if match is empty *)

getMatchLength(match :Match) :Int
(* Return value in a match length field, raise error if match is empty *)

getMatchEnd(match :Match) :Int
(* Return end offset value in a match, raise error if match is empty *)
```

```
  addNewIndex(name, description :String) :Int
  (* Add a new index for field retrieval *)

end;
```

Literaturverzeichnis

ASU88 A. V. Aho, R. Sethi, and J. D. Ullman. *Compilerbau.* Addison-Wesley, 1988.

BC92 N. J. Belkin and W. B. Croft. Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM*, 35(12), pages 29–38, December 1992.

BCC94 J. Broglio, J. P. Callan, and W. B. Croft. INQUERY System Overview. In *Proceedings of the TIPSTER Text Programm (Phase 1)*, pages 47–67, San Francisco, 1994. Morgan Kaufmann.

BS95 I. Bronstein and K. Semendjajew. *Taschenbuch der Mathematik.* Verlag Harri Deutsch, Thun und Frankfurt/Main, 1995.

CCH91 J.P. Callan, W.B. Croft, and S.M. Harding. The INQUERY Retrieval System. In *Proceedings of the Third International Conference on Database and Expert Systems*, 1991.

Cha91 E. Charniak. Baysian Networks without Tears. *AI Magazine*, 14(4), pages 50–63, 1991.

CIIR94 Center for Intelligent Information Retrieval. *INQUERY Application Programmer's Interface Documentation, INQUERY Release Version 2.1.* University of Massachusetts Computer Science Department, Amherst, October 1994.

CIIR95a Center for Intelligent Information Retrieval. *INQUERY Document Retrieval System, INQUERY Documentation, INQUERY Release Version 2.1.* University of Massachusetts Computer Science Department, Amherst, January 1995.

CIIR95b Center for Intelligent Information Retrieval. *Text Collection Parsing, INQUERY Documentation, INQUERY Release Version 2.1.* University of Massachusetts Computer Science Department, Amherst, January 1995.

FMS91 H.P. Frei, S. Meienberg, and P. Schäuble. The Perils of Interpreting Recall and Precision Values. In *Proceedings Information Retrieval GI/GMD-Workshop*, 1991.

Fuh92 N. Fuhr. Probabilisitc Models in Information Retrieval. *Computer Journal*, 35(3), pages 243–255, 1992.

Fuh93 N. Fuhr. Information Retrieval, Skriptum zur Vorlesung im SS 93. Kapitel 2 IR-Konzepte, 1993.

HKW94 M. Hemmje, C. Kunkel, and A. Willett. Eine graphische Benutzerschnittstelle für ein Volltext-Retrieval-System auf der Basis interaktiver dreidimensionaler Visualisierung. Technical Report GMD-Studien Nr.232, Gesellschaft für Mathematik und Datenverarbeitung mbH, June 1994.

KR77 B.W. Kerningham and D.M. Ritchie. *The C Programming Language*. Prentice Hall, 1977.

LS79 M.E. Lesk and E. Schmidt. Lex - a lexical analyzer generator. In *UNIX Programmer's Manual*, Murray Hill, NJ, 1979. Bell Telephone Laboratories, Inc.

Mat93 F. Matthes. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung*. Springer-Verlag, 1993.

MMS94 F. Matthes, M. Müßig, and J.W. Schmidt. Peristent Polymorphic Programming in Tycoon: An Introduction. Technical Report FIDE/94/106, FIDE Project Coordinator, Department of Computing Sciences, University of Glasgow, Glasgow G128QQ, August 1994.

MSS95 F. Matthes, G. Schröder, and J.W. Schmidt. Tycoon: A Scalable and Interoperable System Environment. In M.P. Atkinson, editor, *Fully Integrated Data Environments*. Springer-Verlag, 1995.

NIS88 Information Standards Organization NISO. American National Standard Z39.50. In *Information Retrieval Service Definition and Protocol Specifications for Library Applications*. New Brunswick, 1988.

Pfe95 U. Pfeifer. HTTPs älterer Bruder; WAIS: Inhaltsorientierte Suche im Internet. *iX*, (1), page 120, 1995.

RBP+91 J. Rumbaugh, M. Blaha, M. Premerlani, E. Frederick, and Lorensen W. *Object-Oriented Modelling and Design*. Prentice Hall, 1991.

SBGK94 M. Scheller, K-P. Boden, A. Geenen, and J. Kampermann. *Internet: Werkzeuge und Dienste*. Springer-Verlag, 1994.

SM83 G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

TC91 H.R. Turtle and W.B. Croft. Evaluation of an Inference Network-Based Retrieval Model. *ACM Transactions on Information Systems*, 9(3), pages 187–222, 1991.

TC92 H.R. Turtle and W.B. Croft. A comparision of text retrieval models. *Computer Journal*, 35(3), pages 279–290, 1992.