Die Sphere-Search-Suchmaschine zur Graphbasierten Suche auf Heterogenen, Semistrukturierten Daten

Dissertation
zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

vorgelegt von
Jens Graupmann

Saarbrücken, 2006

Dekan der Naturwissenschaftlich- Technischen Fakultät I:	Prof. Dr. Thorsten Herfet
Vorsitzender der Prüfungskommission:	Prof. Dr. Reinhard Wilhelm
Erstgutachter:	Prof. Dr. Gerhard Weikum
Zweitgutachter:	Prof. Dr. Christoph Koch
Akademischer Beisitzer:	Dr. Ralf Schenkel
Tag des Promotionskolloquiums:	11.05.2006

Inhaltsverzeichnis

Kurzfassung

In dieser Arbeit wird die neuartige SphereSearch-Suchmaschine vorgestellt, die ein einheitliches ranglistenbasiertes Retrieval auf heterogenen XML- und Web-Daten ermöglicht. Ihre Fähigkeiten umfassen die Auswertung von vagen Struktur- und Inhaltsbedingungen sowie ein auf IR-Statistiken und einem graph-basierten Datenmodell basierendes Relevanz-Ranking. Web-Dokumente im HTML- und PDF-Format werden zunächst automatisch in ein XML-Zwischenformat konvertiert und anschließend mit Hilfe von Annotations-Tools durch zusätzliche Tags semantisch angereichtert. Die graph-basierte Suchmaschine bietet auf semi-strukturierten Daten vielfältige Suchmöglichkeiten, die von keiner herkömmlichen Web- oder XML-Suchmaschine ausgedrückt werden können: konzeptbewusste und kontextbewusste Suche, die sowohl die implizite Struktur von Daten als auch ihren Kontext berücksichtigt. Die Vorteile der SphereSearch-Suchmaschine werden durch Experimente auf verschiedenen Dokumentenkorpora demonstriert. Diese umfassen eine große, vielfältige Tags beinhaltende, nicht-schematische Enzyklopädie, die um externe Dokumenten erweitert wurde, sowie einen Standard-XML-Benchmark.

Abstract

This thesis presents the novel SphereSearch Engine that provides unified ranked retrieval on heterogeneous XML and Web data. Its search capabilities include vague structure and text content conditions, and relevance ranking based on IR statistics and a graph-based data model. Web pages in HTML or PDF are automatically converted into an intermediate XML format, with the option of generating semantic tags by means of linguistic annotation tools. For semi-structured data the graph-based query engine is leveraged to provide very rich search options that cannot be expressed in traditional Web or XML search engines: concept-aware and link-aware querying that takes into account the implicit structure and context of Web pages. The benefits of the SphereSearch engine are demonstrated by experiments with a large and richly tagged but non-schematic open encyclopedia extended with external documents and a standard XML benchmark.

Zusammenfassung

Suchmaschinen für das World-Wide-Web liefern für einfache Anfragen, deren Antwort vielfach zu finden ist, hervorragende Ergebnisse. Falls aber die Anfrage selbst komplexer oder die gesuchte Information nur in wenigen Dokumenten zu finden ist, ist die Ergebnisqualität meist unbefriedigend. Dies liegt auf den ersten Blick hauptsächlich an der im World-Wide-Web vorherrschenden Keyword-basierten Suchmethodik. Allerdings würde eine Erweiterung der Anfragesprache alleine zu keiner Verbesserung führen, da schon bei der Indexierung nur die Worte der Dokumente selbst untereinander ungeordnet als „Bag-of-Words“ im Index abgelegt werden.

XML-Technologien hingegen bieten die Möglichkeit, Information strukturiert zu repräsentieren, zu speichern und zu suchen. Suchmaschinen für XML erfordern allerdings meist schon a priori die Kenntnis der Struktur der gesuchten Information, was in vielen Anwendungsfällen nicht vorausgesetzt werden kann.

In dieser Arbeit stellen wir die SphereSearch-Suchmaschine vor, die ein einheitliches leistungsfähiges Retrieval auf heterogenen Daten - von Web-typischen HTML-Dokumenten bis hin zu reich annotierten XML-Daten - ermöglicht und vom Benutzer weder Kenntnisse der Dokumentstruktur, noch das Erlernen einer komplexen Anfragesprache erfordert. Diese Suchmaschine ermöglicht eine konzeptbewusste, kontextbewusste und abstraktionsbewusste Formulierung und Auswertung von Suchanfragen.

Konzeptbewusstsein beschreibt die Fähigkeit der Suchmaschine, unterscheiden zu können welchem semantischen Konzept ein Suchbegriff zugeordnet ist, d.h. in welcher Rolle er vorkommt. So kann die Suchmaschine unterscheiden ob „Rice“ in der Bedeutung als Nahrungsmittel oder Personenname gesucht wird. Kontextbewusstsein besagt, dass die Suchmaschine nicht nur isoliert einzelne Dokumentelemente oder Dokumente betrachtet, sondern auch ihr Umfeld. So können auch Treffer, die sich über mehrere per HREF-Link oder XLink verbundene Dokumente erstrecken bestimmt werden. Abstraktionsbewusstsein beschreibt die Eigenschaft, nicht nur inhaltlich exakte Treffer, sondern auch semantisch ähnliche Treffer finden zu können. So liefert eine Suche nach Zauberern und Hexenmeistern auch Dokumente über Schamanen und die Suche nach Komponisten des 18. Jahrhunderts auch Dokumente, die nicht den Suchbegriff „1800-1899“ oder „18.Jhdt.“, aber die Jahreszahl 1815 enthalten.

Zur Formulierung von Anfragen wird die *Sphere Search Language* (SSL) vorge-

stellt, die diese Eigenschaften in sich vereinigt. Sie bleibt bei erheblich stärkerer Ausdruckskraft gegenüber Web-Anfragesprachen deutlich einfacher als existierende XML-Anfragesprachen.

Um eine einheitliche Repräsentation aller Dokumente zu gewährleisten, werden zunächst Dokumente aller Formate in ein XML-Zwischenformat umgewandelt. Ziel der Umwandlung ist es, die semantische Struktur, die den Dokumenten zugrunde liegt, in eine explizit für die Suchmaschine nutzbare Repräsentation zu überführen. Hierzu werden Transformationen durchgeführt, die teilweise von heuristischer Natur sind. Den nach der Umwandlung indexierten Dokumenten werden anschließend durch den Einsatz von linguistischen Analysewerkzeugen Annotationen hinzugefügt.

Für die Auswertung der Anfragen wird ein leistungsfähiges und flexibles Graph-basiertes Datenmodell vorgestellt. Dieses repräsentiert den gesamten Dokumentkorpus mit seinen Annotationen als Graph und erlaubt eine dokumentgrenzenübergreifende Auswertung. Knoten sind hierbei typisierte Fragmente der ursprünglichen Dokumente.

Zur Bestimmung und Relevanzbewertung von potentiellen Ergebnissen auf diesem Graphmodell werden die neuen Konzepte Sphären und Kompaktheit eingeführt und mit Standard-Technologien des Information-Retrieval kombiniert. Eine Sphäre umfasst hierbei die Knoten im Umfeld eines Knotens und beinflusst somit den Relevanzwert eines Treffers. Kompaktheit misst auf Basis graphbasierter Algorithmen die Ausdehnung eines Treffers. Um diese Algorithmen effizient ausführen zu können, wird ein Top-k-Algorithmus vorgestellt.

Um die Umsetzbarkeit und Leistungsfähigkeit dieses Ansatzes zu demonstrieren, wird das implementierte Prototypsystem beschrieben und durch Experimente auf verschiedenen heterogenen Korpora evaluiert. Diese Evaluation belegt einen deutlichen Gewinn hinsichtlich der Ergebnisqualität bei komplexeren Suchanfragen. Um aufzuzeigen, dass dieser Ansatz auch auf reinen XML-basierten Korpora gute Ergebnisse liefert, wird SphereSearch auf dem XML-basierten Benchmark INEX ausgeführt.

Summary

Web search engines provide excellent results for simple queries whose answers can be found on numerous web pages. However, if the query itself is more complex or the demanded information is only contained in few documents, the result quality is mostly unsatisfactory. At first sight, this is due to the predominating keyword-based search-methodology on the web. Just extending the query language would not improve the situation as the indexing process only stores the unsorted document content as a„bag of words".

In contrast, XML technologies provide means to represent, store and search information in a structured way. But XML search engines expect the user to know the structure of the actual results in advance, which is simply not often the case.

In this thesis we introduce the SphereSearch search engine that provides powerful unified retrieval on heterogeneous data - ranging from typical HTML documents to richly annotated XML documents - without expecting the user to have any prior knowledge of the document structure or to learn a complex query language. This search engine makes a concept-aware, context-aware and abstraction-aware formulation and evaluation of queries possible.

Concept awareness denotes the ability of the search engine to differentiate between different semantic concepts that could be assigned to a query term in a document, viz. its specific role. This enables the search engine to decide wether e.g. „Rice" is meant as nutrition or as a person's name.

Context awareness means that the engine does not only consider isolated document fragments or documents but also their environment. As a consequence, matches can be found which consist of a set of pages that are closely connected via (HREF-,X-) Links.

Abstraction awareness characterizes the abilitiy to determine not only exact matches but also semantically related ones.

An abstraction aware search engine returns when asked for documents about wizards not only results about wizards but about shamans as well. A query asking for composers of the 18th century also provides results that do not contain „18th century" or „1800-1899" but the year „1815".

To phrase such queries the *Sphere Search Language* (SSL), which incorporates these features, is introduced. Although SSL being much simpler than existing XML query languages its expressivness is, compared to Web query language, much more

powerful.

To provide a uniform representation of all documents in different formats they are initially transformed into an intermediate XML format. The goal of this transformation is to transfer the underlying semantic structure of the document into a representation that can be leveraged by the search engine. This is provided by transformations that are partially of heuristic nature. After transformation semantically meaningful tags are added by means of linguistic tools.

A powerful and flexible graph-based data model is proposed to evaluate queries. In this model the whole corpus and its attached annotations are represented as a graph that facilitates evaluation across document borders. Each single node of this graph represents a typed fragment of the original document.

To determine the relevance, based on the proposed data model, of a potential answer the novel concepts *spheres* and *compactness* are introduced and combined with standard IR techniques. A sphere comprises all nodes in the environment of a node and therefore influences the node's score. Compactness measures, based on graph-based algorithms, the extension of a match. To execute this algorithm efficiently a Top-k algorithm is introduced.

To demonstrate the feasibility and effectiveness of our appraoch the prototype system is described and evaluated by experiments on diverse heterogeneous document corpora. These experiments clearly show a significant improvement in result quality for complex queries. Furthermore we show that the system also performs very well on purely XML-based corpora by conducting experiments with the XML-based INEX benchmark.

1 Einführung

1.1 Motivation

Die Entwicklung zur Informationsgesellschaft und die damit einhergehende Explosion an verfügbarer Information führt, zumindest auf den ersten Blick, zu dem Paradoxon, dass aufgrund der wachsenden Informationsflut das (persönliche) Informationsdefizit steigt: Man findet einfach nicht mehr die gesuchte Information, d.h. die Suche nach Information wird immer häufiger zur Suche nach der Nadel im Heuhaufen, obwohl der Zugang zu ihr keinesfalls versperrt ist.

Suchmaschinen, die helfen sollen der Informationsflut Herr zu werden, präsentieren aber immer häufiger nur 'Mainstream'-Ergebnisse. Dies führt auch zu hervorragenden Ergebnissen, falls die Anfrage selbst eine 'Mainstream'-Anfrage war: „Britney Spears ring tone“ sollte von jeder Suchmaschine problemlos beantwortet werden können. Falls aber die Anfrage eine differenziertere Bedeutung hat, es sich also um eine „Experten-Anfrage“ handelt, oder die Anfrage mehrdeutig ist, sind die Ergebnisse meist wenig überzeugend.

Dann nämlich wird durch eine Kombination aus Algorithmen, die die Linkstruktur nutzen, um den Ergebnisrang eines Treffers zu heben, und durch ausdrucksschwache unstrukturierte Anfragesprachen das Ergebnis hoffnungslos tief in der Resultatsmenge, die nicht selten viele hunderttausend Treffer umfasst, regelrecht verschüttet.

Ein Beispiel hierfür ist eine Anfrage nach den Eltern von Britney Spears („Britney Spears parents“). Denn kaum ein Treffer, den Suchmaschinen mit Vorkommen der Begriffe „parents“ und „Britney Spears“ präsentieren, beinhaltet Informationen über die Eltern *von* Britney Spears.

Dieses Problem betrifft gleichermaßen Gesellschaft, Industrie und Wissenschaft. Es tritt in großen Intranets, digitalen Bibliotheken und natürlich der größten aller Datensammlungen, dem World Wide Web auf. Es basiert zu einem Großteil darauf, dass die enthaltene Information für Menschen kodiert wurde; insbesondere wurde der Hauptaugenmerk auf die spätere Präsentation und nicht auf die inhaltliche Strukturierung gelegt. Wer eine Suchmaschine nutzt, delegiert nun die eigene Suche an eine Maschine. Für Maschinen allerdings bleibt die intendierte Struktur der Information, die aus ihrer für Menschen erzeugten visuellen Struktur ersichtlich wird, verschlossen. Klassische Information-Retrieval-Ansätze, die diesen Suchmaschinen zugrundeliegen, „lösen“ diese Probleme, indem sie jegliche Struktur ignorieren und

Dokumente als „Bag-Of-Words“ betrachten. Als Folge entsteht das oben geschilderte Problem: Differenzierte Anfragen können überhaupt nicht mehr beantwortet werden, weil die Grundlage zur Beurteilung einer guten Antwort - die Struktur - nicht mehr verfügbar ist.

Obwohl sich im geschäftlichen Umfeld auf Seite der Informationskodierung und des Informationsaustauschs strukturierter Daten XML als die *Lingua Franca* des Informationszeitalters durchgesetzt hat, ging mit dieser Lösung vieler Probleme keine vergleichbare Entwicklung in Bereichen, in denen Daten deutlich heterogener in Struktur und Inhalt sind, wie Intranets oder dem World Wide Web, einher. Eine seit einiger Zeit propagierte Lösung dieses Problems soll das *Semantic Web* bzw. die unter dieser Bezeichnung propagierten Technologien und Konzepte sein. In der visionären Welt des *Semantic Web* ist die gesamte Information in Form von mit Hilfe der *Web Ontology Language* OWL annotierten und strukturierten XML-Dokumenten abgelegt. OWL ist eine Beschreibungslogik, die somit automatische Inferenz ermöglicht, also das automatische Interpretieren und Folgern von Zusammenhängen und Beziehungen der Daten untereinander. Eine wesentliche Vorbedingung des Einsatzes dieser Technologien ist jedoch ein allgemeiner Konsens über das zur verwendende Vokabular, d.h. die verwendeten Ontologien, und eine konsequente Annotation bzw. Kodierung jeglicher Information mit Hilfe dieses Vokabulars.

Leider sieht die Realität anders aus: sowohl das Web als auch andere Datensammlungen sind weit entfernt von dieser Vision. Die meisten Seiten im Web sind nach wie vor einfaches HTML ohne semantische Annotation. Auch wenn XML eingesetzt wird, ist dies oft unschematisch und hochgradig heterogen in Hinblick auf Vokabular und Struktur. Einige Ausnahmen mögen Dateien sein, die aus anderen metadatenbehafteten Formaten wie SGML und Latex generiert wurden. Aber auch hier beschränkt sich die semantische Aussagekraft auf Tags wie $< section >$ oder $< item >$, wenn nicht auch hier semantisch inhaltslose Tags wie $< p >, < ps >, < h2 >, ...$ zu finden sind. Der Einsatz einer Standard-XML-Anfragesprache, wie z.B. XQuery, auf diesen Daten scheitert an der Unkenntnis der zugrundeliegenden Struktur der Daten. Auf der anderen Seite stehen die Web-Suchmaschinen, die aufgrund des ihres Keyword-basierten Suchparadigmas zwar universell, aber wesentlich ausdrucksschwächer sind. Die folgenden Beispiele sollen die Defizite beider Ansätze und das Potential einer Kombination aufzeigen. Jedes Beispiel verdeutlicht hierbei eine spezifische Eigenschaft, die aktuellen Ansätzen fehlt.

- **Konzeptbewusstsein**
  Die Suche nach Information über den berühmten Physiker Max Planck. Das Eintippen der Anfrage „Researcher Max Planck“ führt zwar zu vielen Ergebnissen über Physiker, die an Max-Planck-Instituten arbeiten, aber eben nicht über den gesuchten Max Planck selbst. Es fehlt die Möglichkeit auszudrücken, dass „Max Planck“ in einer bestimmten Rolle in den Ergebnissen

vorkommen soll, nämlich selbst als Person oder Forscher. Eine bessere, wenn auch an dieser Stelle fiktive *konzeptbewusste* Anfrage würde die Anfrage „researcher=Max Planck“ sein. Eine Beantwortung dieser Anfrage würde natürlich von einer entsprechenden Annotation der Daten profitieren, die z.B. mit Hilfe von State-of-the-Art-Technologien für Named-Entity-Recognition (für Personen, Orte, Zeitangaben etc.) bewerkstelligt werden könnte.

- **Kontextbewusstsein**
  Die Suche nach Professoren aus Deutschland, die eine Vorlesung über Datenbanken halten und Forschung über XML betreiben. Diese Anfrage kann schlecht von einer Web-Suchmaschine beantwortet werden, da kaum eine einzelne Web-Seite die gesamte gesuchte Information beinhaltet und somit ein Treffer wäre. Vielmehr würde eine typischen Antwort auf diese Anfrage eine eng verbundene Menge von Web Seiten sein: z.B. die Homepage eines Wissenschaftlers, die seine Anschrift und einen Link zu einer zweiten Seite über die von ihm gehaltenen Vorlesungen hält, sowie eine Seite mit seinen Forschungsprojekten.

  Eine erfolgreiche Ausführung einer solchen Anfrage erfordert die Berücksichtigung der Dokumente im Umfeld eines Dokuments, welches eine oder mehrere Teilanfragen erfüllt. Eine Suchmaschine muss also eine solche Anfrage *kontextbewusst* auswerten. Es ist zu beachten, dass eine solche Auswertung nicht nur baumartig strukturierte Daten berücksichtigen muss, sondern auch allgemeine, durch Links (z.B. XLinks oder XPointer) aufgespannte Graphstrukturen

- **Abstraktionsbewusstsein**
  Die Suche nach Dramen, in denen eine Frau einem Adeligen eine Prophezeiung macht, dass er König werden wird (*woman prophecy nobleman*). Diese Anfrage kann nicht einfach beantwortet werden, da ein Treffer nicht notwendigerweise die Begriffe „woman“, „prophecy“ und „nobleman“ etc enthält. Ein typisches Ergebnis würde vielmehr wie folgt aussehen:

  Third Witch: All hail, Macbeth, thou shalt be king hereafter!
  ...
  First Witch: All hail, Macbeth! hail to thee, thane of Glamis!

  Die Auswertung dieser Anfrage erfordert von einer Suchmaschine ein hohes Maß an *Abstraktionsbewusstsein*, um zu erkennen, dass:

  – es sich bei *witch* um eine Frau handelt

– *shalt be* sich auf eine Prophezeiung bezieht

– *thane* der Titel eines schottischen Edelmanns ist.

XML-Information-Retrieval-Ansätze mit Relevanz-Ranking wie XXL [TW02], XIRQL [FG01] oder XSearch [Coh03] haben sich teilweise mit diesen Problemen für reine XML-Korpora, deren Dokumente allerdings diversen Schemata folgen können, befasst. Allerdings selbst wenn man die Zukunft des Web optimistisch sieht und XML das Web dominieren wird, wird es aller Wahrscheinlichkeit nach heterogen in Hinblick auf Struktur, Vokabular, Annotationen und Dokumentformate bleiben. So kann nur Keyword-Suche nach Google-Art eingesetzt werden.

Trotzdem sind XML-IR-basierende Ansätze mit Relevanz-Ranking im Vergleich zu Web-IR und Standard-XML-Anfragesprachen am vielversprechendsten um akzeptable Treffer für solch anspruchsvolle Anfragen zu finden. Die Forderung nach Relevanz-Ranking ist an dieser Stelle essentiell, da eine Suchmaschine, die sehr viele potentielle Ergebnisse ermittelt, diese aber nicht nach Relevanz sortiert, nutzlos ist.

Aus den obigen Beispielen und der Diskussion kann man folgende Anforderungen an eine bessere Suchmaschine für das Web und andere heterogene Datensammlungen ableiten:

- Sie soll Anfragen auf XML- und HTML-basierten Web-Dokumenten in einer einheitlichen Form mit Relevanz-Ranking ermöglichen.

- Sie soll *konzeptbewusste*, *kontextbewusste* und *abstraktionsbewusste* Suche unterstützen. Für XML-Daten ist dies schon teilweise verfügbar, für HTML wäre es ein deutlicher Forschritt im Vergleich zum State-Of-The-Art.

1.2 Beitrag dieser Arbeit

Diese Arbeit befasst sich mit der Lösung der oben geschilderten Probleme. Der Hauptbeitrag dieser Arbeit liegt darin aufzuzeigen, wie graph-basierte XML-IR-Technologien mit Web-Suchtechnologien kombiniert und auf heterogene Daten angewendet werden können. Hierzu werden eine neuartige Anfragesprache und eine neuartige graphbasierte Auswertungsstrategie entwickelt, um eine einheitliche Suche auf heterogenen XML- und Web-Daten zu ermöglichen. Diese Konzepte und Methode sind im SphereSearch-Prototypsystem implementiert, der sowohl zu Teilen von XML-IR-Suchmaschinen, wie XXL, als auch von Techniken aktueller Web-Suchmaschinen inspiriert wurde.

SphereSearch hebt sich von vorherigen Arbeiten ab, da es alle Dokumente als Graphen und nicht als Bäume interpretiert. Die besonderen Features von SphereSearch sind:

1. SphereSearch ist deutlich leistungsfähiger als State-of-the-Art-Suchmaschinen, indem es nicht nur ranglistenbasiertes Information Retrieval (Ranked Information Retrieval) umsetzt, sondern auch konzept-, kontext- und abstraktionsbewusstes Retrieval ermöglicht.

2. Die SphereSearch-Anfragesprache ist deutlich einfacher als existierende Anfragesprachen für XML, wie XPath oder XQuery, aber dennoch ausdrucksstark.

3. SphereSearch behandelt XML, HTML und andere Daten in einer einheitlichen Weise, indem jedes Ursprungsformat typgerecht nach XML konvertiert wird. Hierbei werden Heuristiken und linguistische Tools eingesetzt, um *semantisch reiches* XML zu erzeugen.

4. SphereSearch erweitert XML-IR auf beliebige Graphstrukturen, indem es Dokumentgrenzen überschreitet und mit einer Bewertungsfunktion arbeitet, die die Kompaktheit des Ergebnisses widerspiegelt.

1.3 Gliederung der Arbeit

Die vorliegende Arbeit ist wie folgt aufgebaut.

Im folgenden **Kapitel 2** wird auf verwandte Arbeiten aus verschiedenen Forschungsrichtungen eingegangen, die sich mit der Suche auf semi-strukturierten und strukturierten Daten befassen. Anschließend werden einige Grundlagen dieser Arbeit erläutert. Hierzu gehören insbesondere die Indexierung von XML-Dokumenten, Informationsextraktion und die WordNet-Ontologie.

In **Kapitel 3** wird zur Orientierung ein kurzer Überblick über die Architektur des hier vorgestellten Gesamtsystems gegeben.

Kapitel 4 behandelt die Umwandlung von HTML-Dokumenten in das semantisch angereicherte XML-Zwischenformat des Systems. Einige der hierzu verwendeten Techniken und Heuristiken werden vorgestellt.

In **Kapitel 5** wird das SphereSearch zugrunde liegende Datenmodell motiviert und definiert. Insbesondere wird ein Schwerpunkt auf die graphbasierte Repräsentation von Dokumenten und die Unterstützung von Datentypen und Ähnlichkeitsoperationen gelegt.

Die SphereSearch-Anfragesprache wird in **Kapitel 6** erst anhand eines Beispiels anschaulich eingeführt und erläutert und dann formal spezifiziert.

In **Kapitel 7** wird auf die Auswertung der Anfragen, d.h. die Relevanzwertbestimmung von Treffern auf Basis des vorgestellten Datenmodells, eingegangen. Die Relevanzwertbestimmung in SphereSearch ist hierbei zweistufig. Zuerst wird auf die Bestimmung von lokalen Score-Werten, den Sphären-Scores, und anschließend auf die Berechnung der globale Score-Komponente, der Kompaktheit, eingegangen.

Kapitel 8 befasst sich mit der effizienten Umsetzung der in Kapitel 7 vorgestellten Verfahren. Hierzu wird ein Top-k-Verfahren zur effizienten Bestimmung der besten Treffer einer Anfrage vorgestellt.

Das System wurde vollständig als Prototyp implementiert. **Kapitel 9** geht ausführlich auf seine Komponenten ein.

Um die Leistungsfähigkeit des Systems zu belegen, wurde das Prototypsystem experimentell evaluiert. Die Ergebnisse hierzu sind in **Kapitel 10** zusammengefasst.

Die meisten in dieser Arbeit behandelten Aspekte wurden auf wissenschaftlichen Konferenzen und Workshops präsentiert [GCS05, GS05, GSW05, Grau04, GBT+04] . In [GSW05] wurde erstmals ein kohärentes Gesamtbild der SphereSearch-Suchmaschine, wie es in dieser Arbeit in ausführlicher und überarbeiteter Form vorgestellt wird, präsentiert. Das in dieser Arbeit vorgestellte Datenmodell und die darauf basierende Auswertung typspezifischer Anfragen basiert auf [GS05].

2 Grundlagen und verwandte Arbeiten

2.1 Informationssuche auf (semi-)strukturierten Daten

Das Forschungsgebiet der Suche auf strukturierten und semi-strukturierten Daten hat eine lange Historie in unterschiedlichen Kontexten, die allerdings nicht völlig isoliert voneinander betrachtet werden können. So wurden viele aktuelle Arbeiten im Kontext von XML durch frühere Arbeiten im Kontext von SGML und Web-Anfragesprachen inspiriert, und neuere Arbeiten auf dem Bereich der Keyword-Suche auf relationalen Daten durch XML-Information-Retrieval-Ansätze beeinflusst. Abbildung 2.1 zeigt einige relevante Arbeiten in ihrem historischen und konzeptionellen Kontext. Im Folgenden werden verwandte und relevante Arbeiten in den fünf Kontexten *Web-Anfragesprachen*, *Information-Retrieval auf strukturierten Dokumenten*, *XML-Anfragesprachen*, *Information-Retrieval auf XML-Daten* und *Information-Retrieval auf relationalen Daten* vorgestellt.

2.1.1 Web-Anfragesprachen

Im Kontext von Web-(HTML-)Daten wurde auch vor dem Durchbruch von XML auf dem Gebiet von strukturierten Anfragesprachen gearbeitet. Meist orientierten sich diese an SQL und OQL, einschließlich Join-Operationen und anderer komplexer Prädikate, erweitert um textuelle Bedingungen. WebSQL [Mih96], W3QS [KS95] und Lorel [A+97] sind klassische Vertreter dieser Web-Anfragesprachen. WebOQL [AM98] und Araneus [AMM97] versuchen hingegen, Web-Daten zu extrahieren und in ein geeigneteres Format zu überführen. So werden bei Araneus die Daten in ein relationale Format gebracht, um später SQL-Anfragen darauf ausführen zu können. Da in allen genannten Ansätze eine relationale Sicht auf das Web vorherrschend ist, basieren sie dementsprechend auch auf reinem Boole'schen Retrieval ohne Ranking.

2.1.2 Information-Retrieval auf strukturierten Dokumenten

Parallel zu den Web-basierten Ansätzen befassten sich Systeme wie Hyperstorm [BANY97] und Hyspirit [FR98] mit dem Transfer von Konzepten aus Information-

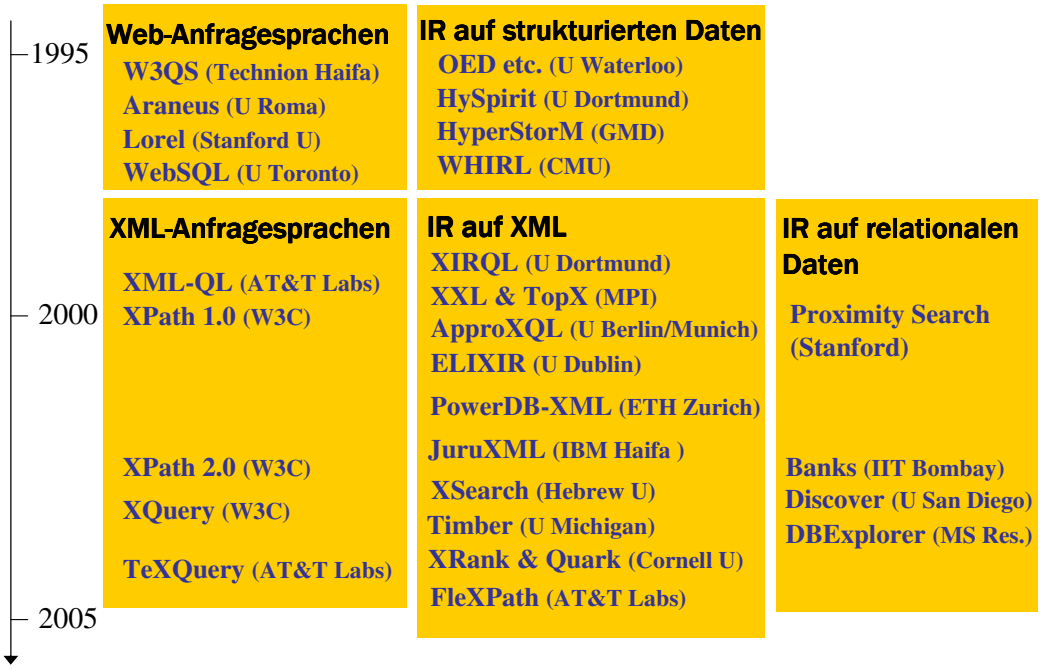


Abbildung 2.1: Verwandte Arbeiten

Retrieval und Datenbanken auf die Gebiete Hypermedia-Retrieval und SGML. Andere Arbeiten befassten sich ausschließlich mit Datenmodellen [MJKZ98] oder Anfragesprachen [Coh98] zur Verarbeitung strukturierter Daten. WHIRL [Coh98] ist hierbei, wie auch die meisten zuvor genannten, ein logikbasierter Ansatz, der komplexe Anfragen über Struktur und Inhalt zulässt und hierbei Methoden des Information-Retrieval integriert. Im Gegensatz zu den Web-basierten Ansätzen berücksichtigen diese Ansätze sowohl Ranking als auch Ähnlichkeitsanfragen. Auch im Bereich relationaler Datenbanken wurde an ersten Ansätzen zur Integration von IR-Features zur Unterstützung von Suchanfragen auf strukturierten Text, insbesondere SGML, gearbeitet [BCJ+94]. Diese Funktionalität wurde meist als Erweiterung von SQL umgesetzt.

2.1.3 XML-Anfragesprachen

XML-Anfragesprachen setzen boole'sches Retrieval, analog zu SQL auf relationalen Daten, auf XML um. Aus den frühren Ansätzen XML-QL [DFF+98], XQL [BN02] und Quilt [CRF00] ist XQuery [XQUERY] hervorgegangen, das sich durch Unterstützung des W3C zum De-Facto-Standard für XML-Anfragesprachen durchgesetzt hat. XQuery subsumiert die Ausdruckskraft von XPath [XPATH02] und erweitert sie um komplexe Anfragekonstrukte und Elemente zur Restrukturierung

des Ergebnisses. Allerdings kann XPath selbst, da es die Spezifizierung von Pfad- und Inhaltsbedingungen erlaubt, als, wenn auch im Vergleich zu XQuery einfache, Anfragesprache für XML angesehen werden.

2.1.4 Information-Retrieval auf XML-Dokumenten

Die zahlreichen *XML-IR*-Ansätze sind motiviert durch die Erkenntnis, dass boole'sches Retrieval, wie es erfolgreich im Kontext stark strukturierter Daten eingesetzt wird, für die Suche auf heterogenen XML-Daten mit teilweise unbekannter Struktur nicht angemessen ist. Viele XML-IR-Ansätze, wie XRank [Coh03] und XSearch [G+03] transferieren IR-typische Schlüsselwortbedingungen auf die Suche auf XML-Daten. XRank überträgt hierbei Konzepte wie Autoritäts-Ranking aus dem Web-Kontext auf die Suche auf XML-Daten, berücksichtigt allerdings nur Schlüsselwortbedingungen und keinerlei strukturelle Bedingungen. XSearch erlaubt hingegen sehr einfache strukturelle Einschränkungen, wie die Spezifikation eines den Suchbegriff umgebenden Tags.

Der Großteil der Ansätze unterstützt allerdings sowohl Inhalt- als auch Strukturbedingungen, indem XML-Anfragesprachen um IR-Features erweitert werden (ELIXIR [CK01], XIRQL [FG01], timber [YJR03], powerdb-xml [GS03]). So erweitert TexQuery die XML-Anfragesprache XQuery und ELIXIR die Sprache XML-QL um umfangreiche Keyword-Features. Bei ELIXIR werden die Inhaltsbedingungen mit Hilfe von WHIRL ausgewertet, das schon in Abschnitt 2.1.2 in anderem Kontext genannt wurde. In GalaTex [CABF05] wurde die vom W3C favorisierte XQuery-Volltext-Erweiterung auf Basis der XQuery-Implementierung Galax [FSC+03] implementiert. Die Systeme TopX [Theo06], XXL [TW02], XIRQL [FG01] und dem System von Hayashi [HTK00] wurden insbesondere im Hinblick auf Ranked Retrieval entwickelt. XXL integriert hierbei umfassend die Unterstützung von Ontologien zur Auswertung von vagen Inhalts- und Strukturbedingungen, während TopX effiziente Top-k-Auswertung auf XML-Daten umsetzt. Andere Ansätze, wie ApproXML [Sch02] und die Arbeiten von Amer-Yahia et al. [ACS02][AKS03][ALP04] fokussieren hingegen auf strukturelle Ähnlichkeiten von Anfragemustern zu Dokumenten, basierend auf Graph-Transformationen und Relaxierungen von Anfragemustern. Zusätzlich zu Transformationen und strukturellen Ähnlichkeitsmaßen berücksichtigt [ACS02] auch Ontologie-basierte Generalisierung von Tag-Namen.

2.1.5 Information-Retrieval auf relationalen Daten

Seit Anfang der 90er-Jahre wurde im Bereich relationaler Datenbanken an der Integration von IR-Features zur Unterstützung von Suchanfragen auf strukturiertem und unstrukturiertem Text gearbeitet. Die meisten Ansätze konzentrieren sich allerdings auf SQL-Erweiterungen, die die genaue Kenntnis des Schemas und somit die

Spezifizierung der zu durchsuchenden Relationen und Attribute erfordern. Alle verbreiteten Datenbanksysteme haben mittlerweile IR-Volltextsuchoperationen implementiert. Allerdings haben sich nur wenige Arbeiten und Systeme mit dem Problem der Keyword-Suche auf relationalen Daten, ohne Kenntnis des zugrundeliegenden Schemas, beschäftigt ([GSBG98] [BHN+02] [ACD02][HP02]). Drei im Kontext dieser Arbeit besonders interessante Ansätze jüngerer Zeit sind Banks [BHN+02] , DBExplorer [ACD02] und DISCOVER [HP02]. In diesen Ansätzen wird die Datenbank als ein Graph mit Tupeln als Knoten und Schlüsselbeziehungen als Kanten angesehen. Antworten sind in diesem Model Teilgraphen des Gesamtdatenbankgraphs. Diese Sichtweise ist dem in dieser Arbeit vorgestellten Datenmodell und der darauf basierenden Auswertungsstrategie ähnlich.

2.2 Indexierung von XML-Dokumenten

Um Daten effizient Suchen zu können, werden die für die Suche notwendigen Informationen in einer Datenbank abgelegt. Um baumstrukturierte XML-Dokumente in einer relationalen Datenbank zu speichern, müssen sie zunächst auf ein relationales Schema abgebildet werden.

Im Unterschied zur Suche auf Standard-Formaten wie reinem Text oder HTML ist die Ergebnisgranulariät bei XML ein einzelnes Element. Aus diesem Grund müssen bei der Indexierung die Struktur und der Inhalt erhalten bleiben. Somit muss auch ein Index für XML-Dokumente mehr Strukturinformation beinhalten.

Da die zu speichernden Dokumente keine einheitliche Struktur aufweisen, müssen sie auf ein generisches, d.h. schemaunabhängiges Datenbankschema abgebildet werden. Hierbei wird jedes XML-Element als (mindestens) ein Tupel in einer Relation abgelegt.

Die Indexierung von XML-Dokumenten ist ein sehr aktiv erforschtes Gebiet. Die meisten Ansätze beruhen darauf, jedem Knoten eine ID zuzuweisen (das Knoten-Label), welches für die effiziente Auswertung von Operationen genutzt werden kann. Existierende Knoten-Labeling-Schemata können auf verschiedene Weisen kategorisiert werden. Der Kategorisierung von [WSM05] folgend, lassen sie sich in nummernbasierte und pfadbasierte Schemata einteilen. Nummernbasierte Ansätze benutzte ganze Zahlen um Knoten zu identifizieren, wohingegen pfadbasierte Ansätze die Position des aktuellen Knoten durch Kodierung des Pfades im Label umsetzten.

Zu den nummernbasierten Ansätzen gehören beispielsweise alle Pre- und Postorder-basierten Ansätze ([Gru02]) und das Bird-Label-Schema ([WSM05]). Die pfadbasierten Ansätze basieren oft auf dem Dewey-Encoding [TVK+02][Dewey] oder seiner optimierten Form ORDPATH [One04]. Orthogonal zu dieser Kategorisierung ist die Einteilung in Ansätze, die externe Pfadindizies nutzen, wie APEX ([C+02]) oder Data-Guides ([GW97]), und Ansätze die die Information kombiniert speichern. Da aber die Vorteile von Pfadindizes, die schnelle Auswertung von exakten Pfa-

danfragen, in dieser Arbeit nicht relevant sind, werden aufgrund der einfacheren Handhabbarkeit nur Ansätze ohne externe Pfadindizes berücksichtigt.

Im Folgenden wird auf die zwei am häufigsten verwendeten Ansätze eingegangen. Dies sind das nummernbasierte Pre-/Postorder-Labeling und das pfadbasierte Dewey-Labeling, von denen letzteres in dieser Arbeit eingesetzt wird.

Im Wesentlichen geht es darum, die die Elemente repräsentierenden Tupel so in einer Datenbank zu speichern, dass die folgenden Operationen effizient ausgeführt werden können:

1. Suchoperationen nach Tag-Namen oder Elementinhalten. Zu einem gesuchten Begriff sollen sowohl die Dokumente, die Treffer beinhalten, als auch die genaue Position des entsprechenden Elements innerhalb des Dokumenthierarchie ermittelt werden.

2. Bestimmung der relativen Position von zwei gegebenen Knoten zueinander. Hierbei sollen sowohl die XPath-Achse [XPATH02], auf der beide Knoten liegen (z.B. Nachfolger, Vorfahre, Geschwisterknoten etc.), als auch die Distanz der Knoten zueinander bestimmt werden können. Insbesondere für das spätere Ergebnisranking ist eine effiziente Auswertung dieser Anfragen essentiell. Für die Distanzberechnung wird hierbei der Baum als ungerichteter Graph betrachtet und die Distanz als Länge des Pfades zwischen den beiden Knoten.

3. Rekonstruktion von Dokumenten bzw. Dokumentfragmente aus dem Index. Hierfür muss jedes Tupel Information beinhalten, aus der die Position innerhalb des ursprünglichen XML-Baums abgeleitet werden kann.

2.2.1 Pre-/Postorder-basierte Ansätze

Viele Ansätze basieren auf dem Preorder- und dem Postorder-Rang einer Depth-First-Traversierung des Dokumentbaums ([DS87] [Gru02]). Der Preorder-Rang ist hierbei die Schrittnummer, in der ein Depth-First-Algorithmus einen Knoten betritt. Der Postorder-Rang ist analog die Schrittnummer, bei der ein rekursiver Depth-First-Algorithmus den Knoten wieder verlässt (Rücksprung). Abbildung 1 zeigt den Pseudo-Code für die Zuweisung der Pre- und Postorder-Ränge.
Der Aufruf geschieht mit $traverse(wurzelknoten, 0, 0)$.

Mit Hilfe der Post- und Preorder-Ränge kann das Verhältnis zweier Knoten in Hinblick auf die vier XPath-Achsen *Ancestor* (Vorfahre), *Following* (Folgender), *Preceding* (Vorhergehender) und *Descendant* (Vorfahre) bestimmt werden. Abbildung 2.2 zeigt einen Beispielbaum mit Pre- und Postorder-Nummerierung sowie ein Koordinatensystem, in dem alle Knoten mit ihren Rängen eingezeichnet sind. Die Preorder-Ränge sind hierbei links der Knoten in rot, die Postorder-Ränge rechte der Knoten in blau dargestellt.

Algorithm 1 Baumtraversierung mit Pre- und Postorder-Rang

```
1: int traverse(node n,int preRank,int postRank)
2: n.preRank = preRank
3: for all children c of n from left to right do
4:     postrank := traverse(c, prerank + 1, postrank)
5: end for
6: n.postrank = postrank
7: return postrank + 1
```

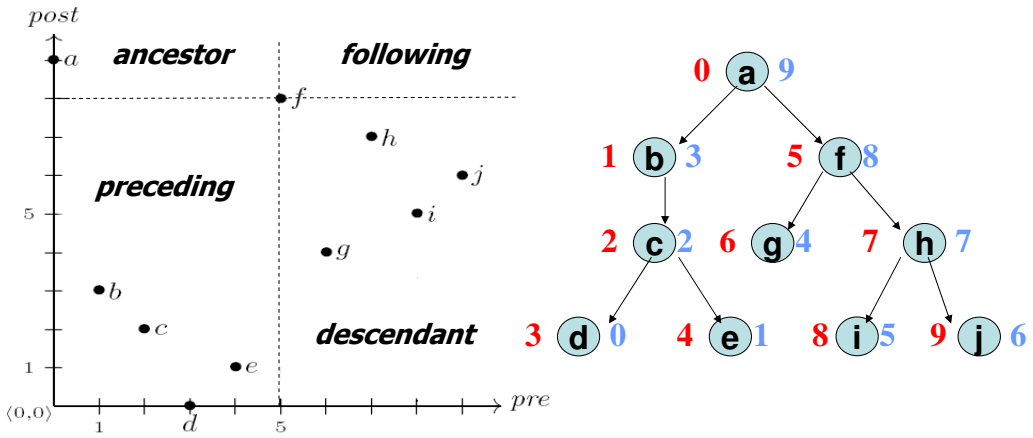


Abbildung 2.2: Pre- und Postorder-Ränge

So können die Nachfahren (Descendants) eines Knotens mit Hilfe ihrer Pre- und Postorder-Ränge wie folgt charakterisiert werden:

v ist Nachfolger von $u \Leftrightarrow pre(u) < pre(v) \wedge post(v) < post(u)$

Ähnlich können die XPath-Achsen *Ancestor*, *Following* und *Preceding* ausgewertet werden. Abbildung 2.2 illustriert dies. Es ist zu sehen, wie durch Auswahl eines Knotens das Koordinatensystem in vier Bereiche segmentiert wird. Die in den Regionen liegenden Knoten entsprechen genau den Knoten der vier Achsen relativ zum ausgewählten Kontextknoten (in der Abbildung ist dies Knoten f).

Dies ermöglicht sowohl die effiziente Auswertung dieser Achsen als auch die Rekonstruktion des Originaldokuments, da bei sukzessiver Rekonstruktion über diese Werte eindeutig die Position des aktuell einzufügenden Knotens bestimmt werden kann. Zu weiteren Details sei auf [Gru02] verwiesen.

Auch wenn XPath-Anfragen effizient ausgewertet werden können, hat dieses Nummerierungs-Schema auch Nachteile:

1. Um festzustellen, ob ein Knoten ein Kind- oder Geschwisterknoten eines zweiten Knotens ist, wird zusätzlich der Rang (Pre- oder Postorder) oder die Tiefe der jeweiligen Vaterknoten der beiden Knoten benötigt.

2. Die Distanz zweier Knoten innerhalb des Dokumentbaums kann nicht aus ihren Pre- und Postorder-Rängen abgeleitet werden. Da für die Distanzbestimmung im SphereSearch-Kontext der Baum als ungerichteter Graph angesehen wird, muss zur Bestimmung des Pfades der nächste gemeinsame Vorfahre beider Knoten innerhalb der gerichteten Baumstuktur ermittelt werden, über den der Pfad zwischen diesen beiden Knoten führt. Für Knoten, die in keiner Vorfahre- bzw. Nachfolgerbeziehung zueinander stehen, ist dies auch mit zusätzlicher Information wie dem Rang oder der Tiefe des Vaterknotens nicht möglich.

Da gerade Distanzinformation für die SphereSearch-Auswertung wichtig ist, wiegt dieser Nachteil schwer.

2.2.2 Dewey-basierte Ansätze

Die Dewey-ID zur Knotennummerierung beruht auf Dewey-Dezimalklassifikation [Dewey]. Sie besteht aus einer Folge von Ziffern, die durch Punkte voneinander getrennt sind. Der Wurzelknoten hat die Dewey-ID 1. Die Dewey-ID aller übrigen Knoten besteht immer aus der ID des Vater-Knotens, konkateniert mit der Kindknoten-Position, mit einem Punkt als Trennelement: Das i-te Kind eines Knotens p hat somit die Dewey-ID: $DeweyID(p)||'.'||i$.

Abbildung 2.3 zeigt den Dokumentbaum des letzten Abschnitts 2.2 , zusätzlich nummeriert mit Dewey-IDs.

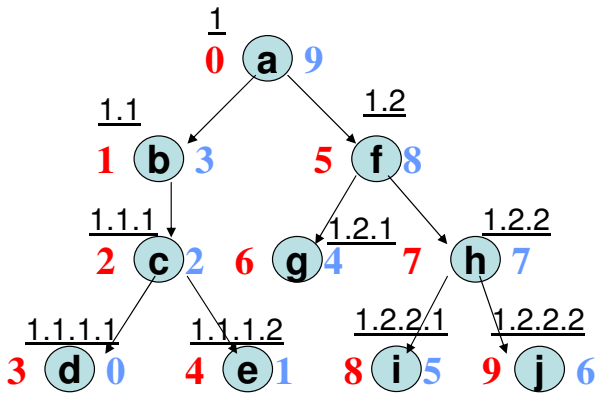


Abbildung 2.3: Baumnummerierung mit Dewey-IDs

Im Gegensatz zur Pre- und Postorder-Kodierung mit zusätzlicher Information über den Vaterknoten (zur Auswertung der Child- und Sibling-Achsen) stellt die Dewey-ID die gesamte benötigte Information in einem, statt in drei Werten bereit.

Bei großer Tiefe der zugrundeliegenden Dokumente wächst die Länge der Dewey-IDs schnell an, wobei allerdings dieser Nachteil gegenüber den Vorteilen vernachlässigbar ist, da die maximale Tiefe der meisten betrachteten Dokumente gering ist (< 8).

Da in der Dewey-ID der Pfad von der Wurzel bis zum Element kodiert ist, können auch Abfragen der XPath-Achsen *Child* und *Sibling* direkt ausgeführt bzw. für zwei gegebene Knoten einfach beantwortet werden.

Insbesondere die für das in dieser Arbeit verwendete Ranking-Verfahren notwendige Distanzbestimmung zwischen zwei Knoten ist einfach möglich. Eine Zusammenfassung der Ausdruckskraft verschiedener Knoten-Label-Schemata, die den Dewey-basierten Schemata zusammen mit dem relativ neuen Bird-Label-Schema die größte Ausdruckskraft bescheinigt, ist in [WSM05] zu finden.

2.3 Informationsextraktion

Informationsextraktion ist ein Gebiet, das von verschiedenen Gruppen weitestgehend unabhängig voneinander intensiv erforscht wird. Auch wenn die verschiedenen Gruppen dieselben Methoden, wie endliche Automaten oder Methoden des maschinellen Lernens einsetzen, ist ihre Herangehensweise signifikant unterschiedlich.

Die Forschungsbereiche, die sich mit Informationsextraktion beschäftigen sind das *Natural Language Processing* und die *Wrapper-Generierung*.

Das Gebiet des *Natural Language Processing (NLP)* hat sich im Kontext unstrukturierter Texte entwickelt. Es versucht durch inhaltliches Verständnis von Texten bestimmte inhaltliche Fakten und Objekte zu erkennen und extrahieren. Somit ist es meist stark kontextabhängig, insbesondere immer sprachabhängig.

Das Gebiet der *Wrapper-Generierung* wurde durch die zunehmende Verbreitung semi-strukturierter Daten und die Notwendigkeit, diese strukturiert weiterverarbeiten zu können, getrieben. So können diese Ansätze meist nicht auf unstrukturierten Daten, wie z.B. reine Volltexte, eingesetzt werden. Im Gegensatz zu *NLP*-basierten Ansätzen spielt die inhaltliche Analyse kaum eine Rolle. So arbeiten Ansätze zur *Wrapper-Generierung* meist auf Oberflächen-Features, wie beispielsweise den verwendeten HTML-Tags und deren Abfolge. Die Sprache des zugrundeliegenden Textes ist meist irrelevant.

Das Ziel beider Ansätze, un- bzw. semi-strukturierte Daten in eine strukturiertere Form zu überführen, ist allerdings gleich.

2.3.1 Wrapper-Generierung

Ein Wrapper bezeichnet in diesem Kontext eine Komponente, die den Inhalt von spezifischen (Web-)Seiten extrahiert und in ein strukturierteres bzw. anwendungsabhängig besser geeignetes Format transformiert. Im Gegensatz zu NLP-basierten Ansätzen ist der Extraktionsvorgang domänenunabhängig, da er sich primär nach der Struktur und nicht nach dem Inhalt richtet. Anstatt den Inhalt zu analysieren, arbeitet die Wrapper-Generierung auf strukturellen Features, insbesondere der Tag-Struktur.

Meist werden Wrapper-Ansätze in manuelle Ansätze (Wrapper-Generierungssprachen), halbautomatische/überwachte Ansätze und automatische Ansätze eingeteilt [ACMM03, BEG+05].

Manuelle Ansätze basieren auf spezialisierten Wrapper-Generierungssprachen. Diese sind hochspezialisierte und komplexe Sprachen, mit deren Hilfe sich der Extraktionsprozess manuell durch Experten spezifizieren lässt (Abbildung 2.4-A). Beispiele hierfür sind *Araneus* [AMM97], *W4F*[SA99] und *JEDI* [HFAN98].

Ziel der semiautomatischen Ansätze ist es Extraktionsmuster (beispielsweise in Form regulären Ausdrücke oder endliche Automaten) ohne manuelle Programmierung zu generieren (Abbildung 2.4-B) . Innerhalb der semiautomatischen Ansätze

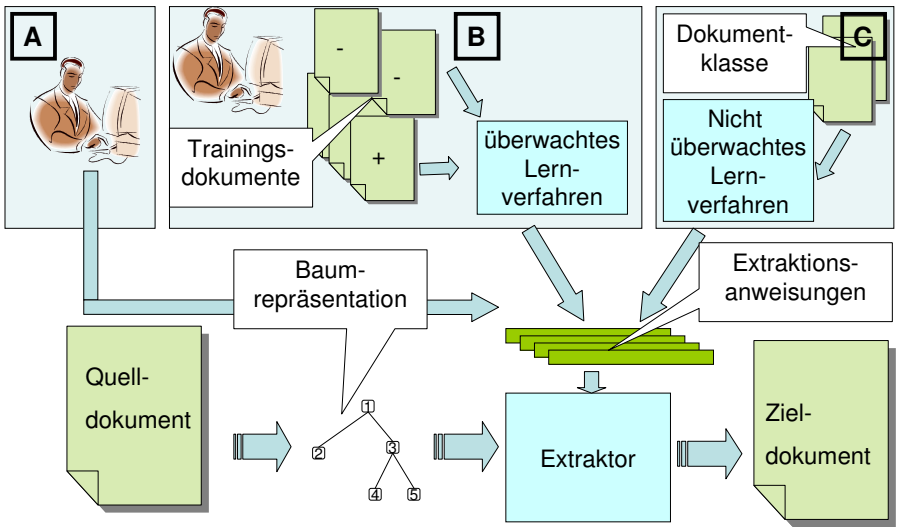


Abbildung 2.4: Wrapper-Generierung

lassen sich interaktive und nichtinteraktive Verfahren unterscheiden. Bei interaktiven Ansätzen wie *XWrap*[LPH00] oder *Lixto*[GKB+04] geschieht die Wrapper-Generierung über ein graphisches Benutzerinterface, das eine visuelle Spezifikation erlaubt. Die nichtinteraktiven Ansätze, wie *Stalker* [MMK99] oder *WIEN* [FK00], beruhen meist auf Lernverfahren, die eine Menge markierter Trainingsdokumente benötigen.

Im Gegensatz zu den semiautomatischen Ansätzen erzeugen die automatischen Ansätze wie *RoadRunner* [CMP02] oder *ExAlg* [AG03] Wrapper nicht mit Hilfe explizit markierte Trainingsdokumente, sondern mit Gruppen ähnlicher, d.h. durch dasselbe Generierungsmuster (*Template*) erzeugter Dokumente (Abbildung 2.4-C). Ziel ist, auf Basis der Gemeinsamkeiten dieser Gruppe strukturell ähnlicher Dokumente das Template zu rekonstruieren (z.B. in Form einer Grammatik), das ihnen zugrunde liegt. Da automatisch erzeugte Wrapper zunächst ohne Benutzerinteraktion erstellt werden, ist manuelle Nachbearbeitung notwendig, um sie den tatsächlichen Anforderungen anzupassen.

Grundsätzlich arbeitet die Mehrheit dieser Ansätze intern auf der Baumrepräsentation von Dokumenten und nicht auf ihrer zeichenbasierten Repräsentation. Nach Durchführung der Extraktion werden die Dokumente einem anwendungsabhängigem Zielformat oder einer Datenbank abgelegt (Abbildung 2.4).

All diesen Ansätzen gemeinsam ist, dass die erzeugten Wrapper, auch wenn sie so robust sind, dass kleinere Änderungen der Web-Seiten keine Auswirkung auf ihre korrekte Funktion haben, für jeweils ein spezifisches Format eines (Web-) Dokuments erzeugt werden. Außerdem ist immer manuelle Arbeit erforderlich, sei es die Auswahl oder Bearbeitung von Trainingsdokumenten, oder die visuelle Erstellung.

2.3.2 Natural Language Processing (NLP)

Information Extraction (IE) ist ein Teil des Forschungsgebiets *Natural Language Processing (NLP)*, welches selbst als Teilgebiet der Computerlinguistik und Informatik (insbes. auch der Künstlichen Intelligenz) angesehen wird. NLP beschäftigt sich mit Problemen, die inhärent bei der Verarbeitung, der Manipulation und dem Verständnis von natürlicher Sprache (sowohl in gesprochener als auch in geschriebener Form) auftreten. Die wichtigsten Teilgebiete des NLP sind:

- Automatische Spracherkennung (*Automatic Speech Recognition* (ASR))
- Informationsextraktion (*Information Extraction* (IE))
- Synthetische Spracherzeugung (*Natural Language Generation* (NLG))
- Maschinelle Übersetzung (*Machine Translation* (MT)).

Das Teilgebiet der Informationsextraktion selbst kann man (dem führenden Forum dieser Forschungsrichtung, der *Message Understanding Conference*, folgend ([MUC98][CMBT02])) in fünf Teilgebiete bzw. Teilaufgaben (*IE tasks*) unterteilen.

- *Named Entity recognition (NE)*(Erkennung von „Namen“): Finden und klassifizieren von Personennamen, Ortsnamen, Organisationen, Terminen und Geldbeträgen etc.. Die Erkennung von Geldbeträgen und Datumsangaben fällt auch unter NE, obwohl es sich hierbei natürlich um keine „Namen“ handelt.
- *Coreference resolution (CO)* (Auflösung von Koreferenzen): Auflösung sprachlicher Ausdrücke, die sich auf dasselbe reale Objekt beziehen. Ein Beispiel hierfür ist die Zuordnung von Pronomen zu *Named Entities*.
- *Template Element construction (TE)* (Konstruktion von Template-Instanzen): Ergänzung von NE-Ergebnissen um beschreibende Informationen in vorher festgelegten Schablonen(*Templates*). So könnte beispielsweise für eine Firma zusätzlich zum Namen, durch CO oder aus anderen Quellen, die Anschrift und Organisationsform hinzugefügt und gebündelt abgelegt werden.
- *Template Relation construction (TR)* (Auffinden von Relationen zwischen Template-Instanzen): Erkennen von Beziehungen zwischen verschieden Templates. Z.B: {Personen_Template} arbeitet_für {Firmen_Template}.
- *Scenario Template production (ST)* (Erzeugung von Instanzen für Szenario-Templates): Zusammenfassung von TE- und TR-Ergebnissen in einem (meist domänenspezifischen) Szenario-Template.

Begriffserklärung *Named Entities*

Der Begriff *Named Entity* bezeichnet eine Entität (beispielsweise eine Person), der ein Name (beispielsweise Jens Graupmann) zugewiesen wurde. Allerdings ist das Verständnis von *Named Entities* ist in der Literatur sehr uneinheitlich (vgl. [PLYM96],[BSAG98]).

Der *MUC-7 Named Entity Task Definition* (im Kontext der Message Understanding Conferences (MUC) [MUC98]) zufolge (die sich an den von Borthwick([BSAG98]) genannten Kategorien orientiert) gibt es die folgenden Kategorien von *Named Entities*:

The Named Entity task consists of three subtasks (entity names, temporal expressions, number expressions). The expressions to be annotated are „unique identifiers“ of entities (organizations, persons, locations), times (dates, times), and quantities (monetary values, percentages).

Dieser Definition folgend werden auch temporale und numerische Ausdrücke als Named Entity angesehen.

Mikheev et al. ([MGM99]) haben aufgrund der nicht eindeutigen Definition die folgende Sichtweise, die in dieser Arbeit geteilt wird, dass: *What counts as a Named Entity depends on the application that makes use of the annotations.*

Somit sind auch alle weiteren domänenspezifischen Erweiterungen der in dieser Arbeit implementierten Annotationskomponente *Named Entites*.

```
The shiny red rocket was fired on Tuesday.
It is the brainchild of Dr. Big Head. Dr.
Head is a staff scientist at We Build Rockets Inc
```

Abbildung 2.5: Beispieltext

Die oben erläuterten Schritte der Informationsextraktion liefern auf dem Beispieltext 2.5 folgende Ergebnisse:

- *Named Entity recognition (NE):*
  „rocket“, „Tuesday“, „Dr. Head“’, „We Build Rockets“
- *Coreference resolution (CO)*:
  „it“ = „rocket“; „Dr. Head“ = „Dr. Big Head“

- *Template Element construction (TE):*
  the rocket is „shiny red“ and Head's „brainchild“.
- *Template Relation construction (TR):*
  Dr. Head works for We Build Rockets Inc.
- *Scenario Template production (ST):*
  rocket launch event with various participants

In dieser Arbeit konzentrieren wir uns ausschließlich auf *Named Entity Recognition*, um vordefinierte Kategorien in Texten zu erkennen und zu annotieren.

Eine Reihe von Systemen und Tools für NLP sind verfügbar und können somit in eigene Applikationen integriert werden. Die folgenden bekannten Tools stellen einen Ausschnitt einiger (prominenter) Vertreter dar.

- Fastus [HAB+96]
- Minorthird [Coh04]
- Connexor [Conn]
- GATE/ANNIE [CMBT02]

Da GATE/ANNIE den größten Funktionsumfang aufweist, in JAVA implementiert ist und sehr intensiv weiterentwickelt und gepflegt wird, wird es in dieser Arbeit eingesetzt. Obwohl Tools wie GATE seit einiger Zeit verfügbar sind, wurden erst kürzlich Anstrengungen unternommen diese in IR-Systeme zu integrieren.

2.3.3 GATE/ANNIE

Ein bekanntes und häufig eingesetztes Tool für NLP ist *GATE*(*General Architecture for Text Engineering*) [CMBT02, Cun02], zusammen mit seinem integrierten Informationsextraktionssystem ANNIE. GATE ist ein Framework zur Erstellung eigener NLP-Applikationen. ANNIE (a Nearly-New Information Extraction System) besteht aus mehreren Modulen für das GATE-Framework, die in einer Pipeline angeordnet werden. In Abbildung 2.6 ist die aus der GATE-Dokumentation ([CMBT02]) entnommene Abbildung der aus ANNIE-Komponenten bestehende Verarbeitungs-Pipeline dargestellt. Im Folgenden werden nur die wichtigsten der dargstellten Module erläutert.

Nach grundlegenden Schritten wie dem *Part-of-Speech-Tagging* durch den *HipHep-Tagger* (dem Erkennen und Annotieren von Satzbestandteilen wie z.B. Nominalphrasen oder Adverbialphrasen) und dem Nachschlagen von Begriffen in spezifischen Listen durch das *Gazetteer Modul* (z.B. in Listen von Ortsnamen, Vornamen,

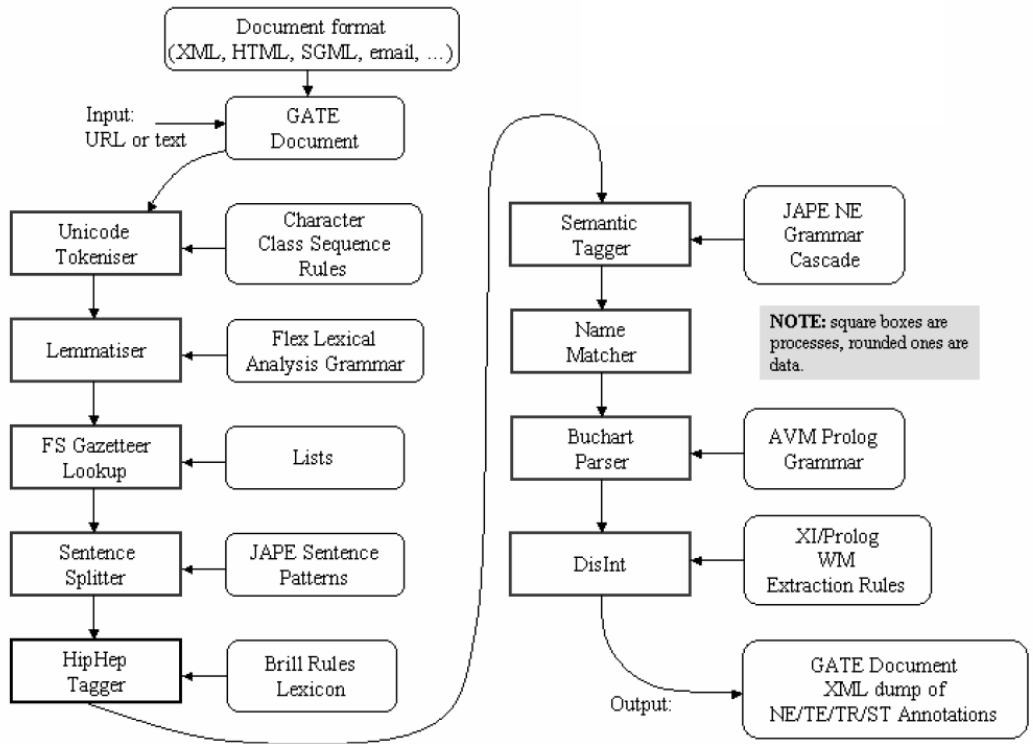


Abbildung 2.6: ANNIE-Pipeline

Währungsangeben etc.) wird der *Semantic Tagger* aufgerufen. Der *Semantic Tagger* benutzt Regeln, die Annotationen ausnutzen, die in vorherigen Stufen generiert wurden. Der *Semantic Tagger* beruht auf dem *JAPE-Transducer*, der Spezifikationen endlicher Automaten, die reguläre Ausdrücke über Annotationen benutzen, interpretiert. Annie kann einfach über zusätzliche Wörterbücher oder auch Regeln erweitert werden. Da GATE/ANNIE in dieser Arbeit verwendet wird, werden im Folgenden einige Details erläutert. Abbildung 2.7 zeigt auf der linken Seite ein Eingabedokument für ANNIE und auf der rechten Seite die visualisierte Annotation im User-Interface von GATE.

2.3.3.1 JAPE

Die *Java Annotation Pattern Engine* (JAPE) kompiliert Grammatiken in (kaskadierte) endliche Automaten. Bei diesen Automaten handelt es sich um *Finite State Transducer* (FST) (Transduktoren), d.h. Automaten, die nicht nur eine Eingabe akzeptieren (wie es ein einfacher endlicher Automat, ein *finite state automaton*(FSA), ausführen würde), sondern auch eine Ausgaben generieren.

Die Grammatik enthält hierbei in Phasen eingeteilte Annotationsregeln, die je-

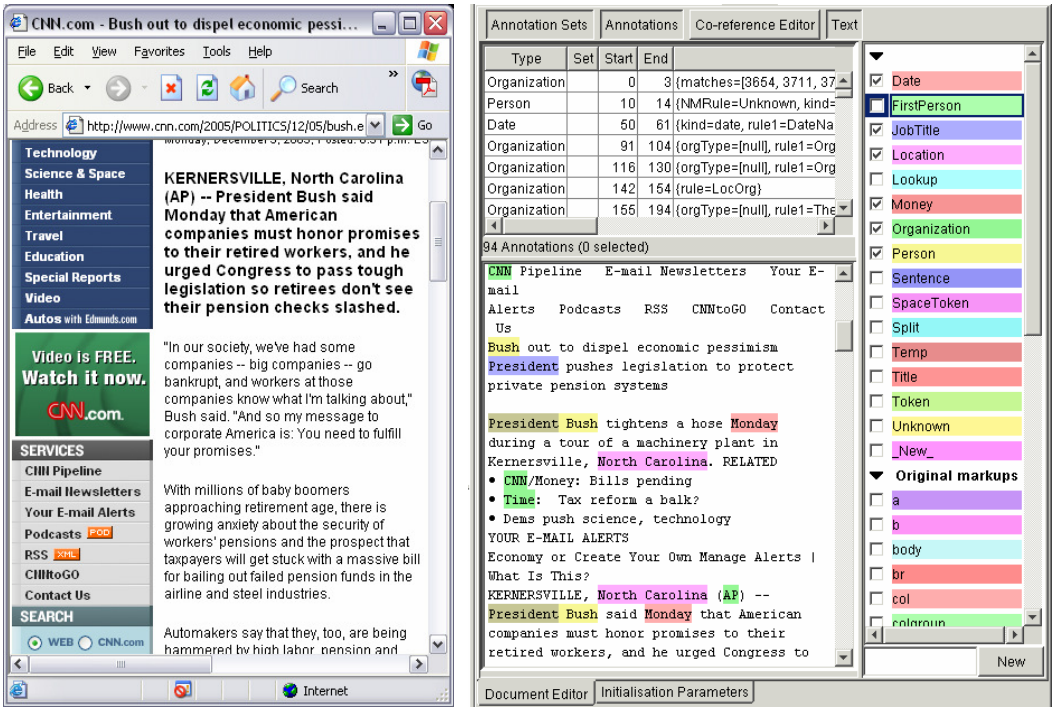


Abbildung 2.7: Annotation mit GATE

weils aus einem Muster (einem regulären Ausdruck über Annotationen) und einer Aktion bestehen, die ausgeführt werden soll, wenn das Muster erfüllt wird. Die Phasen und die aus ihren Regeln erzeugten Automaten werden nacheinander ausgeführt. In den ersten Phasen werden elementare Annotationen hinzugefügt auf denen spätere Phasen aufbauen. Somit arbeitet nur der erste Automat auf dem Ursprungsdokument; jede weitere Phase erhält als Eingabe die um Annotationen erweiterte Ausgaben der vorherigen Phase.

Die Grammatik entspricht weitestgehend der CPSL (*Common Pattern Specification Language*) ([App99]). Ein Beispiel für die linke Seite einer Regel (das Muster) ist das folgende mit Namen `KiloAmount`:

```
Rule: KiloAmount
( ({Token.kind== "containsDigitAndComma"}):number
{Token.string=="kilogram"}):whole
```

Ein einfaches Muster besteht aus `Annotation.attribut=wert` in geschweiften Klammern und wird von einer Annotation mit Namen `Annotation`, Attributnamen `attribut` und Attributwert `wert` erfüllt. Komplexe Muster werden durch runde Klammern gebildet. Hierbei können durch den „:“-Operator Labels zugewiesen

werden, auf die in der rechtes Seite der Regel zugegriffen werden kann. An komplexe Muster können außerdem die Operatoren + (Disjunktion), ? (optionales Vorhandensein) und * (Kleene Star für Wiederholungen) angehangen werden. Eine Folge von von komplexen Regeln repräsentiert eine sequentielle Konjunktion. Eine Disjunktion wird durch den '|'-Operator ausgedrückt. Die obige Teilregel wird also von einem Textteil, der mit *containsDigitAndComma* annotiert ist, gefolgt von einer mit *kilogram* annotierten Zeichenkette, erfüllt. Dem ersten Teil wird hierbei das Label *number*, der gesamten Zeichenkette *whole* zugewiesen.

Ein Beispiel für eine vollständigen Regel ist die Folgende:

```
Rule: NumbersAndUnit
( ( {Token.kind == "number"} )+:numbers {Token.kind == "unit"} )
-->
:numbers.Name = { rule = "NumbersAndUnit" }
```

Ein Beispieltext zur Regel ist in Abbildung 2.8-A dargestellt. Die Regeln werden allerdings nicht auf reinem Text, sondern auf schon durch vorhergehende Bearbeitungsprozesse annotierten Texten durchgeführt. Abbildung 2.8-B zeigt den so vorverarbeiteten Text, auf dem die obige Regel angewandt wird. Annotationen sind als umschließende Tags dargestellt. Die obige Regel wird ausgeführt bei einer Folge von Zahlen, gefolgt von der Angabe einer Einheit (In Abbildung 2.8-A sind die entprechenden Annotationstags fett dargestellt). Als Resultat (rechte Seite der Regel) wird der Bereich der Zahlen (*numbers*) mit einer neuen Annotation „Name“ mit dem Attribut-Wert Paar rule=“NumbersAndUnit“ versehen (siehe 2.8-C).

A) The drink costs 12$

B) <Token kind=“Determiner“>The</Token><Token…>drink</Token><Token…>costs<Token>
<Token>1</Token **kind=“number“**> </Token>2</Token **kind=“number“**>
<Token>$<Tokens **kind=“unit“**/>

C) <Token kind=“Determiner“>The</Token><Token …>drink</Token><Token…>costs<Token>
<Name rule=“numbersAndUnit“>
<Token>1</Token kind=„number“> </Token>2</Token kind=„number“>
<Token>$<Tokens kind=„unit“/>
</Name>

Abbildung 2.8: Beispiel zur JAPE-Regel

2.4 WordNet

WordNet ist eine lexikalische semantische Datenbank für die englische Sprache [Fell98]. Durch ihre Struktur kann WordNet als Ontologie angesehen und eingesetzt werden [Mar03]. WordNet gruppiert Begriffe in sogenannte *Synsets*. Eine Synset steht für ein abstraktes Konzept wie beispielsweise ein Stuhl zum darauf sitzen oder der Lehrstuhl an einer Fakultät. Jedem Synset ist eine Menge von Begriffen zugeordnet, die in natürlicher Sprache für dieses Konzept verwendet werden. Diese Begriffe sind natürlich per Definition Synonyme, da eben diese dieselbe Bedeutung haben.

Da viele Begriffe in natürlicher Sprache mehrdeutig sind, sind sie in WordNet auch in mehreren Synsets enthalten. WordNet liefert für den Begriff „chair“ die folgenden vier Synsets, die zusätzlich mit einem natürlichsprachigem Satz erläutert werden.

Synset 1: **chair** (a seat for one person, with a support for the back) „he put his coat over the back of the chair and sat down“

Synset 2: **professorship, chair** (the position of professor) „he was awarded an endowed chair in economics“

Synset 3: **president, chairman, chairwoman, chair, chairperson** (the officer who presides at the meetings of an organization) „address your remarks to the chairperson“

Synset 4: **electric chair, chair, death chair, hot seat** (an instrument of execution by electrocution; resembles an ordinary seat for one person) „the murderer was sentenced to die in the chair“

Zusätzlich enthält WordNet unterschiedliche Relationen zwischen den Synsets, die ihr semantisches Verhältnis zueinander ausdrücken.

Die wichtigsten semantischen Relationen zwischen Synsets sind:

- Hypernym: Ein Hypernym bezeichnet einem Oberbegriff (Z.B.:Ein KFZ ist ein Fahrzeug).
- Hyponym: Ein Hyponym bezeichnet einen Unterbegriff, ist also die inverse Relation zum Hypernym.
- Holonym: Ein Holonym bezeichnet eine Teil-von-Beziehung (Z.B.:Der Kopf ist ein Teil vom (Holonym zu) Körper).
- Meronym: Ein Meronym ist die inverse Relation zur Holonym-Beziehung.

Mit den Synsets und ihren Relationen bildet WordNet einen Graph, den Ontologiegraph.

Für das Synset „chair (a seat for one person...)“ liefert WordNet die folgenden Hypernyme. Jedes Hypernym ist hierbei ein direktes Hypernym zum Begriff der vorherigen Zeile und transitiv ein Hypernym zu vorhergehenden Begriffen. Somit entspricht die folgende Auflistung einem Pfad im Ontologiegraph (für weitere Details siehe [Theo03]).

chair (a seat for one person,....)
seat (furniture that is designed for sitting on)
furniture, piece of furniture, article of furniture (furnishings that make a room ...)
furnishing ((usually plural) the instrumentalities that make a home (or other area) livable)

Allerdings enthält WordNet keine Information über das Maß der Ähnlichkeit einer Beziehung zwischen zwei Synsets. Mit diesem Problem beschäftigten sich intensiv die Arbeiten zur XXL- und zur Top-X-Suchmaschine ([Theo03],[TW02],[Theo06]). Zur Berechnung der Gewichtung wurde hierbei die Korrelation der Begriffe statistisch über den *Dice*-Koeffizienten zur Berechnung der Korrelation diskreter Zufallsvariablen bestimmt.

Zur Bestimmung der Stärke einer Relation zwischen zwei Begriffen t_1 und t_2 wird hierbei die Anzahl der Dokumente mit Vorkommen von t_1, die Anzahl der Dokumente mit Vorkommen von t_2, die Anzahl der Dokumenten mit Vorkommen von t_1 und t_2 innerhalb eines Dokuments bestimmt und zueinander über den Dice-Koeffizienten in Relation gesetzt:

$$dice(t_1, t_2) = \frac{freq(t_1 \wedge t_2)}{freq(t_1) + freq(t_2)}$$

Hierbei bezeichnet $freq(t)$ die Anzahl der Dokumente mit Vorkommen des Begriffs t im Korpus. Da jedes Synset aus einer Menge von Begriffen besteht, wird als Ähnlichkeit die maximale Ähnlichkeit von Begriffen der beiden Synsets gewählt. Somit wird für eine Relation zwischen zwei Synsets $s1$ und $s2$ ihr Gewicht wie folgt berechnet.

$$sim(s1, s1) = \max_{t_1 \in s1, t_2 \in s2} dice(t_1, t_2)$$

Zu weiteren Details sei auf [Theo06] und [Theo03] verwiesen.

3 Architekturüberblick

Abbildung 3.1 illustriert die Architektur des in dieser Arbeit konzipierten und implementierten Systems. Alle Teilkomponenten des Systems sind in der Abbildung mit Verweisen auf die entsprechenden Kapitel versehen.

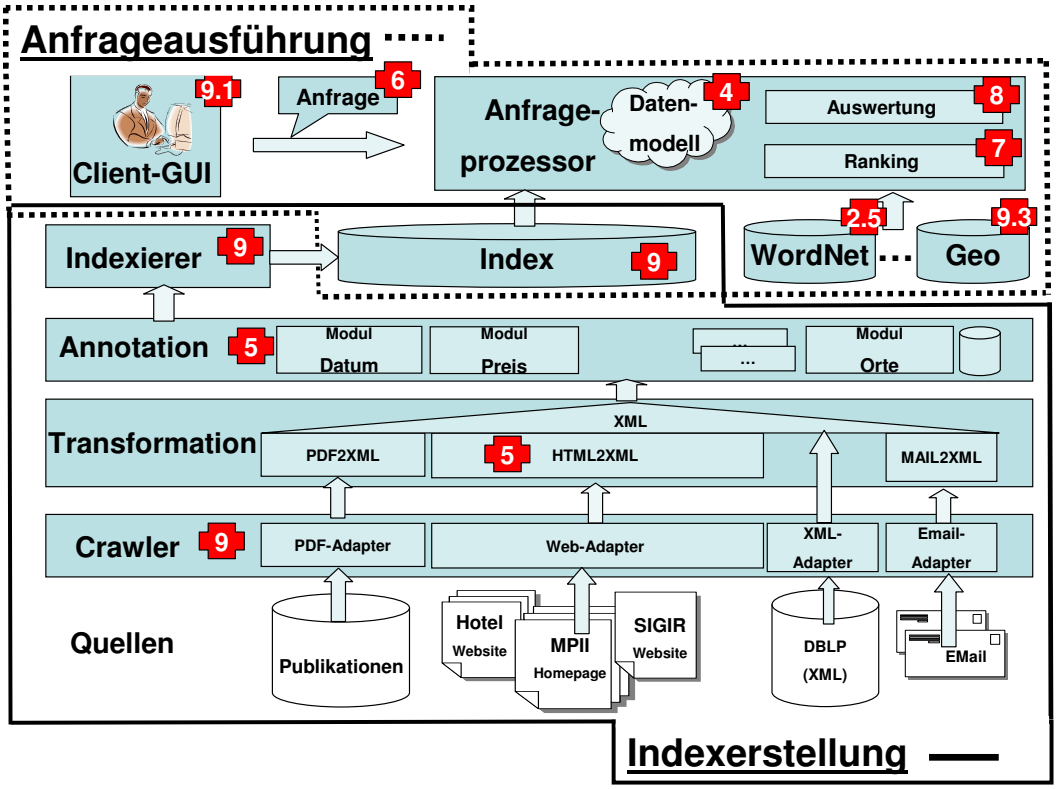


Abbildung 3.1: Systemarchitektur

Die Komponenten lassen sich den Bereichen Indexerstellung und Anfrageausführung zuordnen.

Im Folgenden sind die Bestandteile der Indexerstellung aufgeführt. Sie sind kaskadierend angeordnet, d.h. dass die jeweils nächste Komponente als Eingabe die Ausgabe der vorherigen Komponenten erhält

- Crawler: Der Crawler „sammelt“ die Daten entweder vom lokalen Datei-System oder aus dem Netzwerk. Da sowohl die Extraktion von Linkstrukturen

aus den Dokumenten als auch der Zugang zu den Daten, dokumentformatspezifisch ist, verfügt er über einen entsprechenden Adapter für jedes Format.

- Transformationskomponente: Die Transformationskomponente konvertiert die Ursprungsformate in das interne XML-Format. Jedem Datentyp ist hierbei ein eigenes Transformationsmodul zugeordnet, welches mit Hilfe verschiedener Techniken und Heuristiken Strukturen und Tags der Ursprungsformate in semantisch reichere Tags und Strukturen konvertiert.

- Annotationskomponente: Die Annotationskomponente fügt mit Hilfe von NLP-Technologien (Information Extraction) Annotationen zu den Originaldaten hinzu. Beispiele hierfür sind Personennamen, Orte oder Datumsangaben.

- Indexierer: Der Indexierer zerlegt die Dokumente und speichert sie entsprechend in den relationalen Tabellen des Index.

Für die Anfrageausführung sind die folgenden Komponenten relevant:

- Client: Der Client gestattet die graphische Erstellung und Initiierung von Anfragen aus einem Web-Browser heraus.

- Anfrageprozessor: Die Verarbeitung der Anfrage und Bestimmung der Ergebnisse findet im Anfrageprozessor statt. Folgende Aspekte und Aufgaben müssen im Anfrageprozessor berücksichtigt werden:

  - Datenmodell: Die Daten müssen zur Verarbeitung in das interne Sphere-Search-Datenmodell, das der Auswertung von Anfragen zugrundeliegt, umgewandelt werden.
  - Auswertung: Treffer für die Anfrage müssen effizient in der Datenbank ermittelt werden.
  - Ranking: Basierend auf der, dem internen Datenmodell folgenden, Repräsentation werden die (potentiellen) Treffer entsprechend ihrer Relevanz sortiert.

- Datentypspezifische Auswertungsmodule: Zur Auswertung datentypspezifischer Ähnlichkeitsoperatoren (beispielsweise für Orts- oder Zeitbedingungen) stehen entsprechende Komponenten zur Verfügung.

  - WordNet-basierte Ontologie: Dieses Modul ermöglicht die Ontologiegestützte Auswertung von semantischen Ähnlichkeitsbedingungen.
  - Geo-Server: Dieses Modul dient der Auswertung von ortsspezifischen Ähnlichkeitsanfragen (z.B. „in der Nähe von Berlin“).

4 Internes XML-Zwischenformat

Die Verarbeitung von Dokumenten ist in SphereSearch, bis diese im Index abgelegt werden, XML-basiert. Aus diesem Grund müssen zunächst alle übrigen Formate nach XML umgewandelt werden.

Eine solche Umwandlung in ein anwendungsabhängig besser strukturiertes Format wird meist durch Wrapper-Generierungs-Tools (siehe Abschnitt 2.3.1) durchgeführt. Allerdings setzten sowohl semi-automatische als auch automatische Ansätze voraus, dass die Dokumente durch wenige strukturierte Generierungsmuster (Templates) erzeugt wurden, um Regelmäßigkeiten erkennen zu können. Dies ist in SphereSearch nicht gegeben, da der Dokumentindex i.d.R. aus Dokumenten vieler Quellen und Formate besteht oder die Dokumentquelle über kein striktes Generierungsmuster verfügt (z.B. im Falle der Wikipedia-Enzyklopädie [WIKI]). Aus diesem Grund werden nur Regeln und Heuristiken angewendet, die unabhängig von der Grundstruktur der Seite sind.

Zuständig für diese Umwandlung ist die Transformationskomponente, die über typspezifische Transformationsmodule verfügt. Die Aufgabe eines Moduls ist es, ein spezifisches Ursprungsformat unter Anwendung verschiedener Regeln und Heuristiken nach XML zu konvertieren. Ein Modul wird hierbei durch ein Programm oder eine auf einem XSL-Stylesheet basierende XSL-Transformation umgesetzt. Da XSL-Transformation ebenfalls Turing-vollständig sind [Kep04] haben beide Varianten dieselbe Ausdruckskraft.

Module für zwei Datentypen sind im Prototyp-System implementiert, PDF2XML[1] für die Umwandlung von Dokumenten im PDF Format nach XML, und HTML2XML für die Umwandlung von HTML-Seiten nach XML.

Es handelt sich bei der Umwandlung nicht nur um eine einfache Formatkonvertierung, sondern um eine Transformation der Ursprungsdokumente, die die Dokumentstruktur unter Umständen deutlich ändert. Ziel der Regeln und Heuristiken ist es, die implizit im Ursprungsdokument kodierte Struktur explizit im erzeugten XML zu kodieren. Abbildung 4.2 zeigt dies am Beispiel einer HTML-Seite. Auf der linken Seite der Abbildung ist die Originalseite, wie sie in einem Web-Browser dargestellt wird, zu sehen. In der Mitte der Abbildung ist nun zu sehen wie ein menschlicher Experte den Inhalt der Seite als XML kodieren würde, wenn er diese Seite manuell oder semi-automatisch erstellte. Da SphereSearch viele Dokumen-

[1] Auf PDF2XML wird nicht weiter eingegangen, da dieses Modul in der Prototyp-Implementierung aktuell nur eine sehr einfache Umwandlung durchführt.

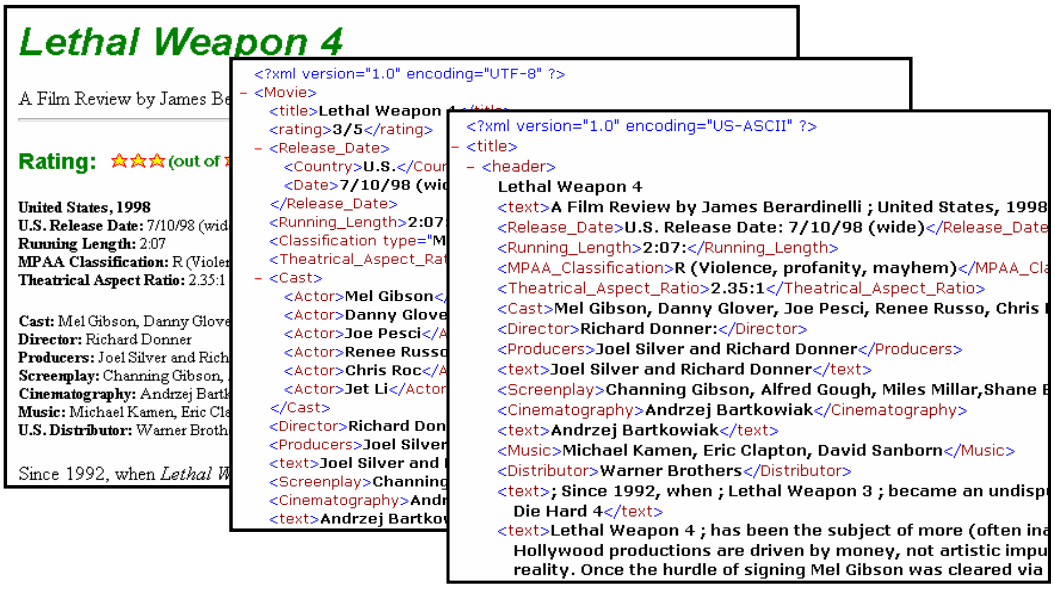


Abbildung 4.1: Transformation von HTML nach XML

te automatisch indexieren soll, ist eine manuelle Umwandlung nicht praktikabel, allerdings können auch viele Aspekte dieser intellektuellen Umwandlung durch Anwendung einzelner Heuristiken automatisiert werden. So hat der menschliche Experte jeweils aus optisch hervorgehobenen Worten zu Beginn einer Zeile ein Tag mit dem folgenden Text als Kindknoten bzw. Inhalt generiert (so wurde beispielsweise aus „**director:**Richard Donner“ das XML-Fragment $<$ *Director* $>$ Richard Donner$<$ /*Director* $>$). Genau solche Aufgaben sollen die Transformationsmodule automatisch durchführen. Auf der rechten Seite der Abbildung ist schließlich die XML-Version zu sehen, die automatisch ohne manuelle Interaktion durch das HTML2XML-Modul erzeugt wurde.

4.1 Von HTML nach XML (HTML2XML)

Da auch die Transformationsregeln selbst auf einem XML-Dokumentbaum arbeiten, muss vor Anwendung eines Transformationsschritts das Dokument in XHTML, eine XML-konforme HTML-Variante, überführt werden. Diese Aufgabe übernimmt die JTidy-Bibliothek [JTIDY]-Bibliothek. Liegt das HTML-Dokument im XML-Format vor, werden nacheinander Transformationsschritte in mehreren Phasen durchgeführt. Die Regeln lassen sich hierbei in zwei Kategorien einteilen:

- **Dokumentregeln**: Ein Dokumentregel erhält als Eingabe den gesamten Dokumentbaum des Dokuments und führt Transformationen auf diesem durch.

Alle nach dieser Regel angewandten Transformationsregeln arbeiten auf dem transformierten Dokumentbaum. Der ursprüngliche Dokumentbaum ist anschließend nicht mehr verfügbar.

- **Tag-Regeln**: Eine Tag-Regel wird immer auf ein spezifisches Tag angewandet. Sie erhält als Eingabe einen Teilbaum, dessen Wurzeltyp dem Typ der Tag-Regel entspricht. Nach der Anwendung einer Tag-Regel wird der der Regel übergebene Teilbaum im Dokumentbaum durch den transformierten Teilbaum ersetzt.

Der Gesamte Transformationsprozess läuft ein drei Phasen ab. In jeder der Phasen wird eine Gruppe von Regeln ausgeführt

1. **Initialtransformationen:**
   Die Dokumenttransformationsregeln der Intitialdokumenttransformation werden in einer festgelegten Reihenfolge nacheinander augeführt. Sie führen einfache Aufgaben zur Vorverarbeitung, wie beispielsweise die Entfernung von Skriptelementen, durch.

2. **Tag-Transformationen:**
   In dieser Phase wird der Dokumentbaum von der Wurzel beginnend traversiert. Wann immer ein Tag, für welches eine Regel existiert, besucht wird, wird die entsprechende Regel auf den mit diesem Element beginnenden Teilbaum angewandt.

3. **Finaltransformationen:**
   Analog zu den Dokumentregeln der Initialtransformationen werden in dieser Phase Dokumenttransformationen in einer festen Reihenfolge aufgerufen. Ziel dieser Stufe ist es, für die Indexierung unnötige Elemente zu entfernen.

Welche Regeln in den entsprechenden Phasen aufgerufen werden, ist konfigurierbar. Tabelle 4.1 zeigt die im SphereSearch-System aufgerufenen Regeln für jede der Phasen.

Da viele dieser Regeln nur sehr einfache Aufgaben erfüllen (beispielsweise das entfernen von „unwichtigen“ HTML-Tag oder von Leerzeichen durch die *CleanerDocumentRule*), wird in den folgende Abschnitten nur auf einige Transformationen eingegangen.

4.1.1 Die HeaderRule-Dokumenttransformation

Eine wichtige Dokumenttransformationsregel ist die Umwandlung von Überschriften in HTML-Dokumenten (repräsentiert durch H1,...,H5-Tags) nach XML. Abbildung 4.2 zeigt auf der linken Seite ein HTML-Fragment mit drei Überschriften und seine

Initialdokumenttransformationen		
Nr.	Regel	Funktion
1	CleanerDocumentRule	entfernt nicht benötigte Tags
2	HeaderRule	erzeugt Schachtelung der Header-Tags
Tag-Transformationen		
Tag	Regel	Funktion
b	BoldRule	wandelt Hervorhebungen um
ul,ol	ListTagRule	transformiert Aufzählungen
img	ImageTagRule	transformiert Bildeinbettungen
table	TableTagRule	transformiert HTML-Tabellen
a	ATagRule	transformiert Links
dl	DLTagRule	transformiert Definitionslisten
Finaldokumenttransformationen		
Nr.	Regel	Funktion
1	CleanerDocumentRule	entfernt nicht benötigte Tags
2	CombineTextDocumentRule	verschmelzt benachbarte Textknoten

Tabelle 4.1: HTML2XML-Regeln

Darstellung in einem Web-Brower. In einem Web-Browser ist die Hierarchie der Elemente ersichtlich: *Settings* und *Results* sind der Hauptüberschrift *Experiments* untergeordnet. Ebenso ist offensichtlich, dass die Textblöcke den Unterpunkten untergeordnet bzw. zugeordnet sind.

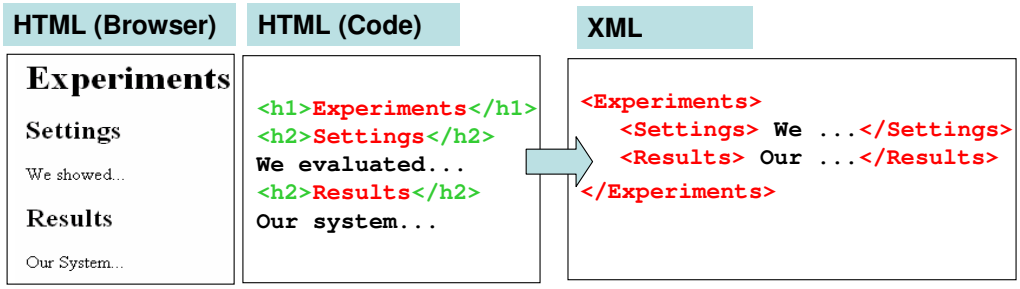


Abbildung 4.2: HeaderRule

Betrachtet man den zugehörigen HTML-Code, sieht man allerdings, dass diese Zuordnung nicht durch eine Schachtelung der entsprechenden Tags widergespiegelt wird. Weder die Hierarchie der Überschriften, noch die Zuordnung der Textblöcke, ist ohne Kenntnis der Semantik der *H*-Tags erkennbar. Die *Header*-Regel restrukturiert das Ursprungsdokument so, dass die intendierte Struktur des

Algorithm 2 HeaderRule

```
1: parent := root
2: parent.level := 0
3: stack.push(parent)
4: while root.hasNextChild() do
5:    n = root.nextChild()
6:    if type(n)! = Header then
7:       parent.append(node)
8:    else if n.type = Header AND n.level > parent.level then
9:       parent.append(n)
10:      stack.push(n)
11:      parent := n
12:   else if n.type = Header AND n.level ≤ parent.level then
13:      while stack.top.level ≥ n.level do
14:         stack.pop()
15:      end while
16:      stack.top().append(n)
17:      stack.push(n)
18:      parent = n
19:   end if
20: end while
```

HTML-Dokuments durch die Schachtelung der XML-Tags des transformierten Dokuments ausgedrückt wird. Der vereinfachte Algorithmus der *HeaderRule* ist in Abbildung *Algorithmus 2* im Pseudo-Code dargestellt. Er geht davon aus, dass alle Überschriften direkte Kindknoten des Wurzelknotens (*root*) sind. Er iteriert über die Liste der Kinder des Wurzelknotens und verwendet einen Stack zur Verwaltung des aktuellen Header-Elements, dem folgenden Knoten als Kindknoten angehangen werden.

Auf der rechten Seite von Abbildung 4.2 ist das Ergebnis der Transformation zu sehen. Es entspricht einer Struktur, in der ein menschlicher Experte das Dokument modelliert hätte.

4.1.2 Die BoldRule-Tag-Transformation

Eine Tag-Transformation, die in der zweiten Phase des Transformationsprozesses aufgerufen wird, ist die *BoldRule*. Falls bei der Traversierung des Dokumentbaums ein $< B >$-Element besucht wird, wird die Regel mit dem in diesem Element wurzelnden Teilbaum aufgerufen.

Die dieser Regel zugrunde liegende Heuristik besagt, dass ein fett dargestellter Begriff ($< B > -Tag$), dem ein Doppelpunkt folgt, eine Art von Überschrift des

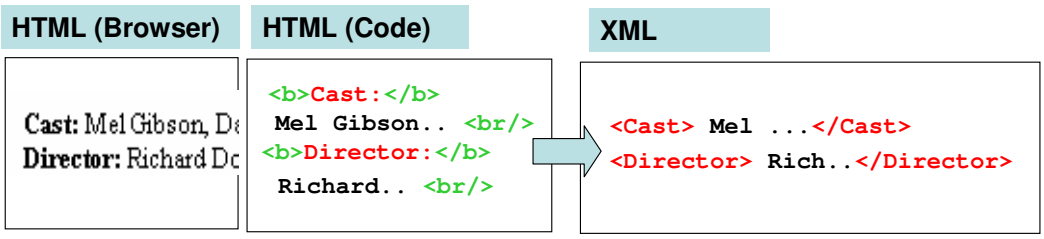


Abbildung 4.3: BoldRule

folgenden Textfragments darstellt. Basierend auf dieser Annahme wird der fett dargestellte Begriffe in ein Tag umgewandelt. Der folgende Text wird ein Kindelement dieses Tag-Knotens. Abbildung 4.3 zeigt ein Beispiel dieser Umwandlung.

Der Begriff *Cast* ist nach der Transformation ein Tag, dass den folgenden Textblock (*Mel Gibson, ...*) umgibt. Dies entspricht ebenfalls (in der Regel) der Art und Weise, wie ein menschlicher Experte dieses Fragment in XML hätte sinnvoll kodieren können.

4.1.3 Link-Transformation

Eine andere Regel wandelt HTML-Links in XML-konforme XLinks um. Aufgrund der unterschiedlichen Link-Konzepte bei HTML und XML erfordert diese Umwandlung mehrere Schritten.

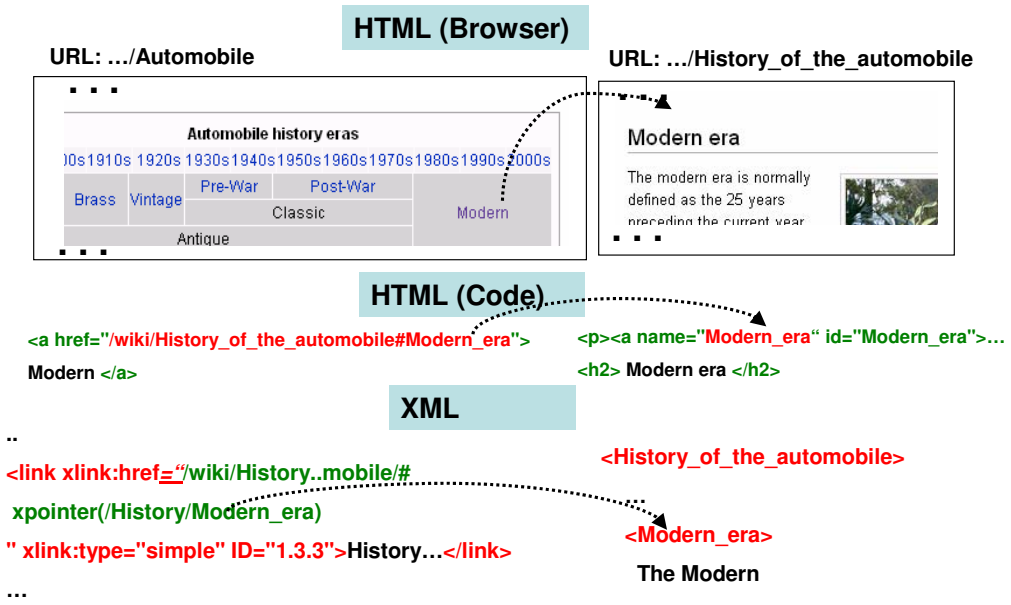


Abbildung 4.4: Link-Transformation

Im Fall von HTML besteht der Link aus einer URL gefolgt von einem Anker-

term. Dieser Ankerterm referenziert ein Element mit identischem Ankerterm im Zieldokument. Es enthält allerdings keinerlei Informationen über die Position dieses Terms im Zieldokument. Zur Auflösung des Links muss das Zieldokument nach diesem Term durchsucht werden. XLinks hingegen bestehen aus einer URL und einem XPath-Ausdruck, der innerhalb des Zieldokuments auf das Zielelement zeigt.

Aus diesem Grund ist eine Umwandlung des HTML-Href-Links in einen XLink ohne Analyse des Zieldokuments nicht möglich, da für die Konstruktion des XPath-Ausdrucks die genaue Position des Ankerterms im Zieldokument benötigt wird. Falls bei der Transformation ein Link gefunden wird, wird zunächst überprüft, ob das Zieldokument schon umgewandelt wurde. Falls dies nicht der Fall ist, wird ein temporäres Symbol erzeugt, das durch den korrekten XPath-Ausdruck ersetzt wird, sobald das Zieldokument umgewandelt wurde. Wurde das Zieldokument schon transformiert, wird die Position des Ankerterms in diesem bestimmt und der entsprechende XPointer erzeugt und dem XLink hinzugefügt.

Abbildung 4.4 zeigt zwei verlinkte Dokumente als HTML-Repräsentation im Web-Browser, als zugehörigen HTML-Code und als transformierte XML-Codierung.

4.1.4 Tabellentransformation

Tabellen sind das Hauptelement in HTML, um Daten strukturiert darzustellen. Allerdings geht die visuelle Struktur, aus der die intendierte Hierarchie für einen Betrachter direkt klar wird, meist nicht aus dem HTML-Quellcode hervor. Aus diesem Grund ist die nichttriviale Problemstellung der automatischen Analyse von HTML-Tabellen in vielen wissenschaftlichen Arbeiten behandelt worden([LN99, YTT01, WH02]). [2]

Um die der Struktur einer Tabelle automatisiert zu erkennen, muss zunächst die Ausrichtung der Tabelle und hiermit verbunden insbesondere die Position des Tabellenkopfs, der die Überschrift enthält, bestimmt werden.

Die HTML-Spezifikation sieht zwar spezielle HTML-Tags für den Tabellenkopf vor, allerdings werden diese, wie in Abbildung 4.5 illustriert, in der die Browser-Darstellung einer HTML-Tabelle und ihr zugehöriger HTML-Code abgebildet sind, kaum genutzt.

Oft enthalten HTML-Tabellen, wie auch der Abbildung zur erkennen ist, Zellen, die sich über mehrere Spalten bzw. Zeilen erstrecken. Damit spätere Algorithmen nicht auf dieser unregelmäßigen Struktur operieren müssen, wird die Tabelle zunächst in einen so genannte *Pseudo-Tabelle* überführt ([WH02]). Eine *Pseudo-Tabelle* hat in jeder Zeile die gleiche Anzahl an Spalten und in jeder Spalte die gleiche Anzahl an Zeilen.

HTML-Tabellen werden mit Hilfe der Tags `TR` für Zeilen und `TD` für Spalten spe-

[2] Aufgrund der Komplexität dieser Aufgabe wurde die Tabellenerkennung und Transformation in dieser Arbeit, im Vergleich zu weitergehenden publizierten Arbeiten, relativ einfach umgesetzt.

Color		engine
interieur	exterieur	
black	grey	2.5 l 140 HP
brown	beige	

```
<TABLE border=1 cellspacing=0 cellpadding=5>
    <TR align=center>
        <TD colspan=2 align=center><b>Color</b></TD>
        <TD rowspan=2 align=center valign=center><b>engine</b></TD>
    </TR>
    <TR>
        <TD><b>interieur</b></TD>
        <TD><b>exterieur</b></TD>
    </TR>
    <TR align=center>
        <TD>black</TD>
        <TD>grey</TD>
        <TD rowspan="2">2.5 l 140 HP</TD>
    </TR>
    <TR align=center>
        <TD>brown</TD>
        <TD>beige</TD>
    </TR>
</TABLE>
```

Abbildung 4.5: HTML-Tabelle: Darstellung und HTML-Code

zifiziert. Erstrecken sich Zellen über mehrere Spalten oder Zeilen wird dies über die Attribute *colspan* und *rowspan* spezifiziert. Zur Konstruktion der *Pseudo-Tabelle* werden nun solche Zellen durch die entsprechende Anzahl Zellen einfacher Höhe bzw. Breite ersetzt. Der Inhalt der ursprünglichen Zelle wird hierbei dupliziert. Abbildung 4.7 zeigt die *Pseudo-Tabelle* zu der in Abbildung 4.5 dargestellten Tabelle.

Color	Color	engine
interieur	exterieur	engine
black	grey	2.5 l 140 HP
brown	beige	2.5 l 140 HP

Abbildung 4.6: Pseudo-Tabelle

Der nächste Schritt ist die Erkennung der Ausrichtung. Hiefür werden die For-

matangaben der ersten Zeile und der ersten Reihe untersucht. Hat die erste Zeile (bzw. Spalte), im Gegensatz zur ersten Spalte (bzw. Zeile), durchgehend identische Formatangaben, wie ein `<B>`-Tag zur Angaben von Fettschrift oder die Angabe der selben Formatklasse, wird von einer vertikalen (bzw. horizontalen) Ausrichtung der Tabelle ausgegangen, d.h. die erste Zeile ist die Tabellenüberschrift und jede Zeile enthält einen Datensatz.

Als letzter Schritt wird die Formatierung der Überschriftszeile (bzw. Spalte) mit der folgenden Zeile (bzw. Spalte) verglichen. Sind diese identisch, werden die beiden Zeilen verschmolzen. Abbildung 4.7 zeigt die Tabelle aus Abbildung 4.5 nach der Verschmelzung der Überschriften.

Color-interieur	Color-exterieur	engine
black	grey	2.5 l 140 HP
brown	beige	2.5 l 140 HP

Abbildung 4.7: Umgewandelte Tabelle

Zusammengefasst läuft die Umwandlung in den folgenden 5 Schritten ab:

1. Erstelle Pseudo-Tabelle
2. Falls TH-Tags zur Spezifizierung des Tabellenkopfes benutzt wurden, springe zu Schritt 5.
3. Bestimme Ausrichtung durch Vergleich der Formatierung der ersten Zeile und Spalte.
4. Verschmelze Zeilen bzw. Spalten, die zum Tabellenkopf gehören.
5. Wandle Tabelle nach XML um.

Kann einer der obigen Schritte nicht durchgeführt werden, wird die Tabellenumwandlung abgebrochen und die Tabelle in ihrer Struktur beibehalten, wobei `TR` und `TD`-Tags durch `Row` bzw. `Cell` Tags ersetzt werden.

Abbildung 4.8 zeigt die Umwandlung nach XML bei erfolgreicher Erkennung und bei nicht erfolgreicher Erkennung. Es ist erkennbar, dass nach erfolgreicher Umwandlung zusammengehörige Informationen „näher“ beieinander stehen, als Informationen verschiedener Datensätze.

Aufgrund des SphereSearch-Ranking-Algorithmus, der die Nähe der Vorkommen von Treffern für Teilbedingungen berücksichtigt, würden diejenigen Ergebnisse bes-

Nicht erkannte Tabelle

```
<TABLE>
    <Row>
        <Cell>Color</Cell>
        <Cell>engine</Cell>
    </Row>
    <Row>
        <Cell>interieur</Cell>
        <Cell>exterieur</Cell>
    </Row>
    <Row >
        <Cell>black</Cell>
        <Cell>grey</Cell>
        <Cell>2.5 l 140 HP</Cell>
    </Row>
    <Row >
        <Cell>brown</Cell>
        <Cell>beige</Cell>
    </Row>
</TABLE>
```

Erfolgreich erkannte Tabelle

```
<TABLE>
    <Row>
        <Color_Interieur>black</Color_Interieur>
        <Color_Exterieur>grey</Color_Exterieur>
        <Engine>2.5 l 140 HP</Color_Exterieur>
    </Row>
    <Row>
        <Color_Interieur>brown</Color_Interieur>
        <Color_Exterieur>beige</Color_Exterieur>
        <Engine>2.5 l 140 HP</Engine>
    </Row>
</TABLE>
```

Abbildung 4.8: Nach XML umgewandelte Tabellen

ser bewertet werden, die die gesuchten Begriffe innerhalb eines Datensatzes beinhalten. Da dies tatsächlich meist die korrekteren Ergebnisse sind, hilft somit eine leistungsfähige Tabellentransformation die Ergebnisqualität einer Suchanfrage zu verbessern.

5 Datenmodell

5.1 Vorüberlegungen

5.1.1 Das klassische IR-Datenmodell

Das am weitesten verbreitete Datenmodell im Information Retrieval ist das Standard-Vektorraummodel ([SG83]). Hierbei werden alle Dokumente in einem hochdimensionalen Raum repräsentiert, der durch die Terme, die nach Vorverarbeitungsschritten (Stopwortelimination und Stammformreduzierung) aus dem Korpus extrahiert wurden, aufgespannt wird. Ein Dokument stellt nun einen Vektor in diesem Raum dar, dessen Komponenten für Terme, die im entsprechenden Dokument vorkommen, ungleich Null sind ([SG83][BR99]). Der Wert einer zu einem Term gehörenden Vektorkomponente ist hierbei im Allgemeinen ein Gewicht (z.B. tf/idf, BM25), das von der Häufigkeit des Vorkommens innerhalb des Textes abhängt. Eine Anfrage ist nun ebenfalls ein Vektor in diesem hochdimensionalen Raum, so dass die Relevanz eines Dokuments bezüglich der Anfrage als Ähnlichkeit seines Vektors zum Anfragevektor bestimmt werden kann. Hierzu werden Ähnlichkeitsmetriken für Vektoren, wie das Kosinusmaß oder die euklidische Distanz benutzt. Anstelle einer Sequenz von Wörtern, die ein Dokument darstellt, wird ein Dokument in dieser Repräsentation als Multimenge von Wörtern, als *Bag-of-Words*, angesehen. Dieses Model hat sich für viele Anwendungen aufgrund seiner einfachen mathematischen Handhabbarkeit und effizienten Umsetzbarkeit durchgesetzt.

Ein Nachteil dieser Darstellung ist allerdings, dass durch die Abbildung auf Vektoren viele Eigenschaften des Ursprungsdokuments verloren gehen. Dies ist zum einen die Reihenfolge von Wörtern, also ihre relative und absolute Position innerhalb des Textes. Zum anderen gehen sämtliche Abhängigkeiten, die z.B. aus der Satzstruktur ersichtlich waren (und durch NLP-Tools hätten bestimmt werden können), und Strukturinformationen, die entweder implizit durch Formatierung oder explizit, z.B. durch (geschachtelte) Tags, enthalten waren, verloren. Somit grenzt dieses Modell die mögliche Ausdruckskraft von Anfragesprachen stark ein, da diese bei zugrundeliegendem Vektorraummodell nur auf Multimengen von Termen, nicht aber auf strukturierten Dokumenten ausgewertet werden können.

In der Einführung dieser Arbeit wurde hinreichend motiviert, dass sich sowohl Dokumentstrukturen als auch der Qualitätsanspruch an gesuchte Information geändert hat, so dass reine Schlüsselwortanfragen nicht mehr ausreichen. Insbesondere

im Kontext von großen strukturierten Dokumenten reicht es nicht mehr aus, dass die entsprechenden Suchterme „irgendwo“ im Dokument zu finden sind. Aus diesem Grund wird in den folgenden Abschnitten ein flexibleres Datenmodell vorgestellt.

5.1.2 Graph- und Intervallbasierte Modelle

Viele Dokumentformate wie z.B. HTML oder XML können als baumstrukturierter Graph dargestellt werden. Abbildung 5.1 zeigt ein einfaches Dokument und seine Baumrepräsentation. Wir bezeichnen die Baumrepräsentation eines Dokuments im Folgenden als *Dokumentbaum*. Wenn baumstrukturierte Dokumente in ihrer Dokumentbaumrepräsentation verarbeitet werden, ist es wichtig, dass die Dokumentreihenfolge von Elementen des Ursprungsdokuments erhalten bleibt, da die Reihenfolge die spätere Relevanzberechnung beeinflusst. Außerdem könnte der Originaltext nicht mehr aus dem Dokumentbaum des Dokuments rekonstruiert werden. Aus diesem Grund werden Dokumentbäume im Folgenden immer als *dokumentreihenfolgeerhaltende Baumgraphen* betrachtet. An die W3C-DOM-Spezifikation [DOM] angelehnt, ist die Dokumentreihenfolge wie folgt definiert :

Definition 5.1.1 (Dokumentreihenfolge)
Die Dokumentreihenfolge ist die auf den Dokumentbaum übertragene Leserichtung des Ursprungsdokuments. Die Reihenfolge der Elemente (bzw. der öffnenden Tags) im Dokument muss mit der Reihenfolge, in der eine Tiefensuche die Knoten des Dokumentgraphs betritt (depth first pre-order traversal), übereinstimmen.

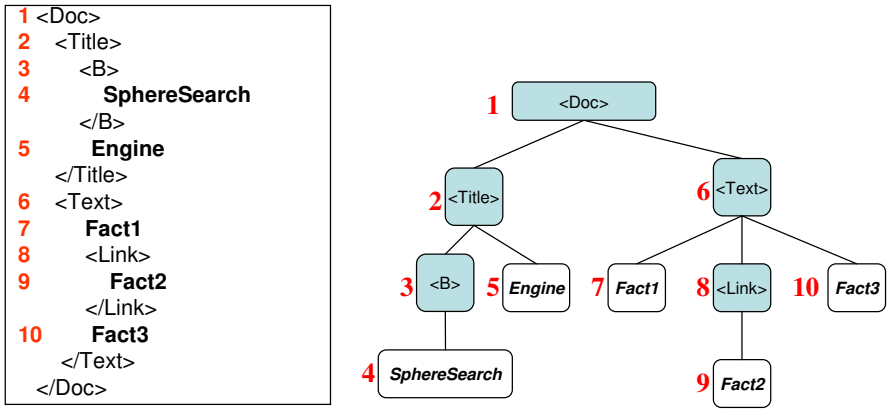


Abbildung 5.1: Dokumentreihenfolge

Falls im Originaldokument ein Textfragment i vor einem Textfragment j gefunden wird, muss der Pre-Order-Rang von i kleiner oder gleich dem Pre-Order-Rang von j sein.

In Abbildung 5.1 sind die Knoten des Dokumentbaums zusätzlich mit ihren Pre-Order-Rängen nummeriert. Da der Dokumentbaum dokumentreihenfolgeerhaltend ist, entsprechen die Nummern im Dokumentbaum der sequentiellen Nummerierung der öffnenden Tags der textuellen Repräsentation.

Betrachtet man nicht nur einzelne Dokumentbäume isoliert, sondern auch die Verlinkung von Dokumenten untereinander und die dokumentinterne Link-Struktur, erhält man (gerichtete) Graphen. Abbildung 5.2 zeigt zwei miteinander über Links verbundene Dokumentbäume und den resultierenden (zyklischen) Gesamtgraph.

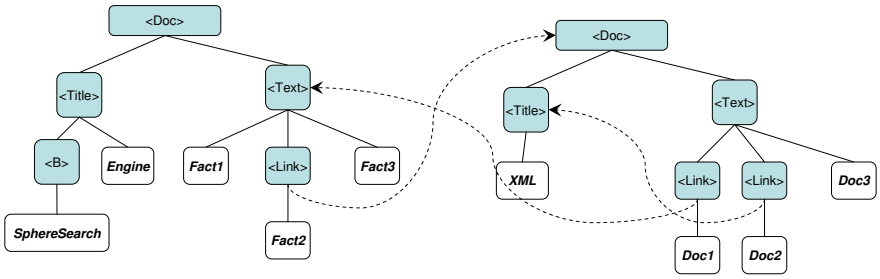


Abbildung 5.2: Dokumentgraph

Arbeiten Annotationstools intern ebenfalls auf der Dokumentbaumrepräsentation, werden Annotationen einfach als zusätzliche Tags eingefügt. Abb 5.3 zeigt ein einfaches XML-Dokument mit und ohne Annotationen [1].

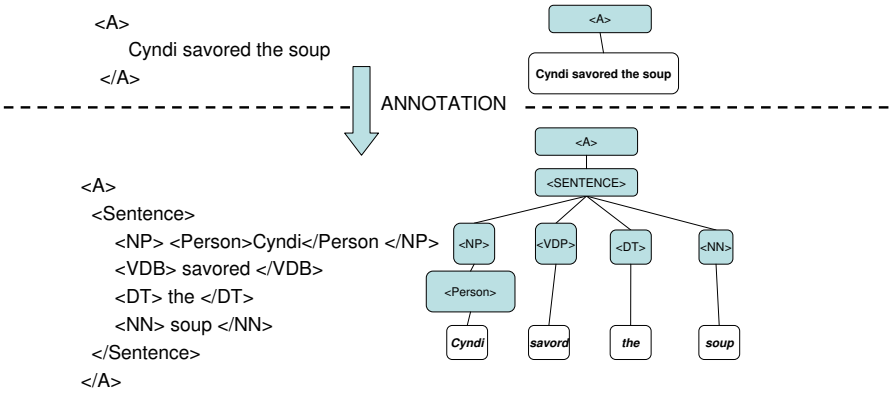


Abbildung 5.3: XML-basierte Annotation

Meist verwenden Annotationstools allerdings keine Graphmodelle, sondern intervallbasierte Modelle ([Gris97]), um Annotationen auf einem Text zu repräsentieren. Jeder Annotation ist hierbei ein Intervall zugeordnet, das durch den Offset der Start- und Endposition des Intervalls relativ zum ersten Zeichen des Gesamttexts

[1] Die eingefügten Tags stehen für *Proper Noun*(NP), *Verb,past* (VBD), *Determiner*(DT) und *Noun* (NN).

spezifiziert wird und somit einen Bereich des Texts beschreibt. Tags werden bei dieser Repräsentation ebenfalls mit Hilfe von Intervallen, analog zu Annotationen, repräsentiert. Das Intervall spezifiziert in diesem Fall den Teil des Gesamttextes, der von dem Tag umschlossen wird. Den Gesamttext, d.h. die Konkatenation aller Textelemente des Ursprungsdokuments, bezeichnen wir als Dokumentinhalt.

Definition 5.1.2 (Dokumentinhalt)
Der Dokumentinhalt ist die Konkatenation der Textelemente eines Dokuments in der Reihenfolge der Leserichtung des Ursprungsdokuments.

Eine alternative zur Dokumentreihenfolgendefinition analoge Definition des Dokumentinhalts über die Reihenfolge, in der eine Tiefensuche die Knoten betritt, ist äquivalent zur der oben gegebenen.

Eine tabellarische Darstellung einer einfachen intervallbasierten Annotation ist in Abbildung 5.4-1 angegeben. Im oberen Teil der Tabelle ist der annotierte Text zu sehen, unter dem sich eine Skala zur Bestimmung der Zeichenposition befindet. Darunter erscheinen zeilenweise die einzelnen Annotationen. Jede Annotation besteht aus der Angabe eines Intervalls, das aus der (absoluten) auf dem Gesamttext zeichenbezogenen Start- (*SpanStart*) und Endposition (*SpanEnd*)besteht, einer ID, dem Typ (*Type*) und zusätzlichen Attributen (*Features*). Die zusätzlichen Attribute sind in der Form *Name=Wert* angegeben.

Im dargestellten Beispiel werden dem Satz *„Cyndi savored the soup“* Annotationen der drei Schritte *Tokenization* (*type=token*) mit *Part-Of-Speech Tagging* (*pos=...*), *Sentence Boundary Recognition* (*type=sentence*), und *Named Entity Recognition* (*type=name*) hinzugefügt. Durch die *Tokenization* wird der Satz in seine atomaren Bestandteile, seine *Tokens*, d.h. Wörter und Satzzeichen zerlegt und durch das *Part-Of-Speech Tagging (POS)* um ihre Satzfunktionen ergänzt. Durch die *Sentence Boundary Recognition* werden Satzgrenzen annotiert und schließlich durch *Named Entity Recognition* Personennamen ergänzt (vgl. Abschnitt 2.3.2).

Werden solche intervallbasierte Methoden um die Möglichkeit der Hierarchisierung erweitert ([CMBT02]), erhält man eine Graphstruktur. Hierbei enthält eine Annotation nicht nur ein Intervall und Attribute, die die Annotation selbst charakterisieren, sondern auch Verweise auf untergeordnete Annotationen bzw. Bestandteile. Die einfache Intervallbasierte Annotation in Abbildung 5.4-1 ist in Abbildung 5.4-2 als um Hierarchisierung erweiterte Annotation dargestellt. In Abbildung 5.4-1 ist die Annotation mit Id 7 (*Sentence*) nicht mit den Annotationen 1-4 (Satzbestandteile, wie *Noun* (NP) oder *Determiner* (DT)) verknüpft, obwohl diese Bestandteil der Annotation 1 sind. Im erweiterten Fall 5.4-2 enthält die Annotation 7 Verweise auf die Annotationen 1-4, die diese als Bestandteile von 7 kenntlich machen.

1

Text

```
Cyndi savored the soup.
|0...|5...|10..|15..|20
```

Annotations

Id	Type	Span Start	Span End	Attributes
1	token	0	5	pos=NP
2	token	6	13	pos=VBD
3	token	14	17	pos=DT
4	token	18	22	pos=NN
5	token	22	23	
6	name	0	5	name_type=person
7	sentence	0	23	

2

Text

```
Cyndi savored the soup.
|0...|5...|10..|15..|20
```

Annotations

Id	Type	Span Start	Span End	Attributes
1	token	0	5	pos=NP
2	token	6	13	pos=VBD
3	token	14	17	pos=DT
4	token	18	22	pos=NN
5	token	22	23	
6	name	0	5	name_type=person
7	sentence	0	23	constituents= [1],[2],[3],[4],[5]

3

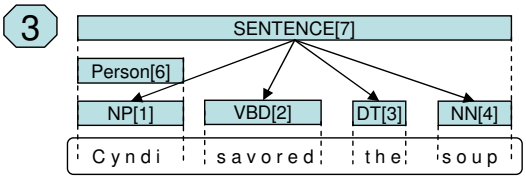


Abbildung 5.4: Intervallbasierte Annotation

Abbildung 5.4-3 zeigt die erweiterte Annotation noch einmal graphisch. Die gestrichelten Linien zeigen hierbei die Bereiche der Annotation bezogen auf den unten in der Abbildung dargestellten Text. Pfeile symbolisieren hierarchische Abhängigkeiten. Es ist direkt ersichtlich, dass die erweiterte intervallbasierte Annotation nicht als reine zusammenhängende Baumstruktur dargestellt werden kann, da nicht zwischen allen Elementen hierarchische Beziehungen bestehen. So ist die Person-Annotation mit keiner der übrigen Annotation, die einen zusammenhängenden Graphen bilden, verknüpft. Außerdem sind prinzipiell auch überlappende Annotationen möglich.

Somit ist die Intervallannotation, obwohl sie eine einfache Zuordnung zu Teilen des Originaltextes ermöglicht, nicht ohne weiteres mit der Graphstruktur semistrukturierter Dokumente vereinbar.

Deswegen wird in dieser Arbeit eine hybride Darstellung verwendet, die sowohl eine graphstrukturierte Dokumentdarstellung ermöglicht als auch die Intervallspezifikation beinhaltet und somit auch überlappende Annotationen zulässt.

Abbildung 5.5 zeigt das hybride Graphmodell (rechts), das die erweiterte intervallbasierte Darstellung (links oben) und die (XML-)graphbasierte Darstellung (links unten) kombiniert und im Abschnitt 5.3 genau spezifiziert und erläutert wird. Jedem Knoten ist in diesem Model zusätzlich zu (potenziell mehreren) Datentypen mit Konfidenzwerten, die Tags oder Annotationen und ihre erwartete Korrektheit repräsentieren, ein Intervall zugewiesen.

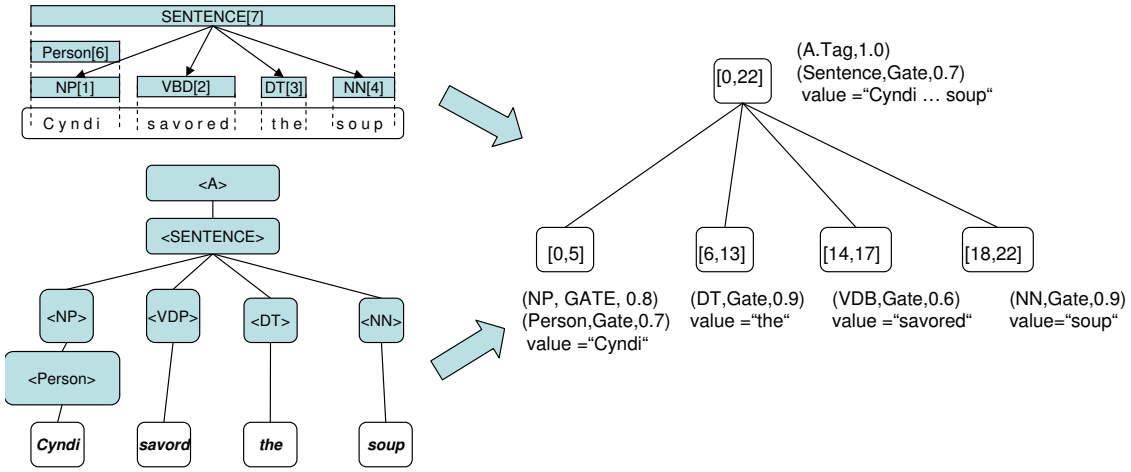


Abbildung 5.5: hybrides Datenmodell

5.1.3 Unterstützung von Ähnlichkeit

Das Konzept der Ähnlichkeit spielt in dieser Arbeit eine zentrale Rolle. Anfragen sollen nämlich nicht nur exakte Treffer zurückliefen, sondern auch „ungefähre“ Treffer, d.h. Ergebnisse, deren Inhalt Teilen der Anfrage ähnlich ist. Aus diesem Grund muss die Auswertung von Anfragen im Hinblick auf Ähnlichkeit explizit im Datenmodell unterstützt werden. Ähnlichkeit ist allerdings ein weiter und nicht klar definierter Begriff.

Selbst innerhalb eines Teilgebietes, z.B. der Mathematik, hat Ähnlichkeit verschiedene Bedeutungen bzw. besitzt verschiedene Ausprägungen. So sind in der Geometrie zwei Objekte zueinander ähnlich, wenn sie durch eine Ähnlichkeitsabbildung (eine geometrische Abbildung, die sich aus zentrischen Streckungen und Kongruenzabbildungen zusammensetzen lässt) ineinander übergeführt werden können, wohingegen in der linearen Algebra zwei quadratische Matrizen A und B ähnlich sind, wenn sie durch eine invertierbare Matrix S ineinander übergeführt werden können.

Grundsätzlich kann man drei Klassen von Ähnlichkeit unterscheiden, die im Information Retrieval besondere Berücksichtigung finden.

- Syntaktische Ähnlichkeit
- Phonetische Ähnlichkeit
- Semantische Ähnlichkeit

Grundsätzlich ist meist die semantische Ähnlichkeit gesucht. In der Suchmaschine XXL[TW02] wurde zur Bestimmung der semantischen Ähnlichkeit eine Ontologie

eingesetzt. In [Theo03] wird eine Ontologie wie folgt definert :

> Eine Ontologie (ein Thesaurus) ist eine Spezifikation von einem repräsentativen Vokabular und Begriffen einschließlich der hierarchischen Beziehungen und der assoziativen Beziehungen zwischen diesen Begriffen auf Basis ihrer Bedeutungen [...]

Auch wenn Ontologie und Thesaurus unterschiedliche Konzepte sind, werden sie in dieser Arbeit als synonym angesehen. Als Ontologie wurde in XXL WordNet (siehe auch Kapitel 2.4) eingesetzt. Da WordNet zum einen umfassend und themenübergreifend und zum anderen frei verfügbar und in zahlreichen Projekten erprobt ist, wird auch in dieser Arbeit WordNet als Basis-Ontolgie eingesetzt.

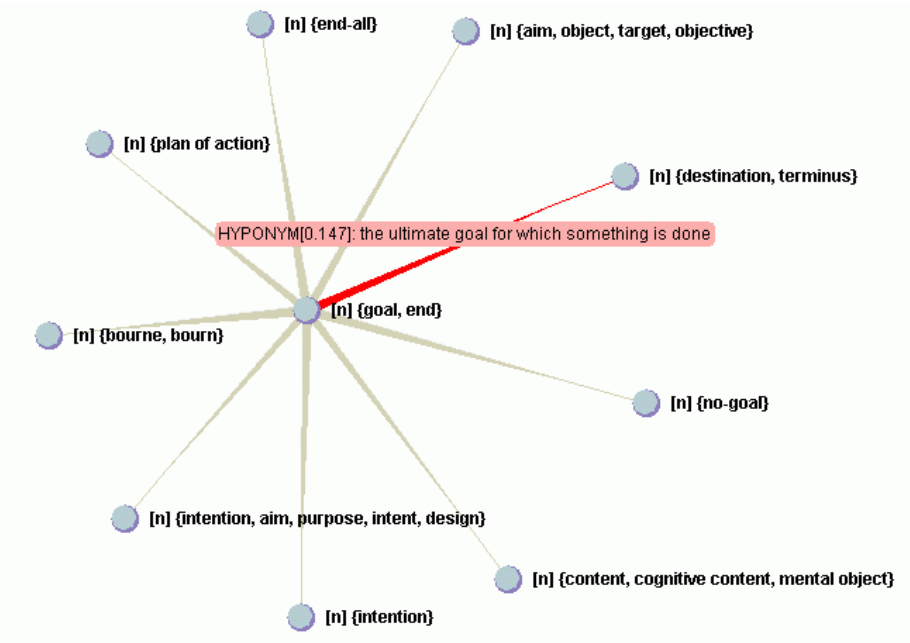


Abbildung 5.6: Die WordNet-Ontologie

Zusätzlich zu der in WordNet enthaltenen Information wurden die Relationen der Begriffe zueinander quantifiziert, d.h. gewichtet (siehe Kapitel 2.4). Abbildung 5.6 zeigt einen Screenshot eines Visualisierungs-Tools für die WordNet-basierte Ontologie, der verwandte Begriffe zum Begriff *Goal* darstellt. Wird eine Relation (im Visualisierungstool entspricht dies einer Kante) selektiert, wird der Typ der Relation und ihre Stärke dargestellt. Die wichtigsten in WordNet enthaltenen Relationen sind Synonyme, Hypernyme (Oberbegriffe) und Hyponyme (Unterbegriffe).

Eine rein WordNet-basierte Lösung zur Bestimmung der Ähnlichkeit, wie sie in XXL eingesetzt wurde, ist aus mehreren Gründen für verschiedene Szenarien und Anwendungen unzureichend:

1. **Fehlendes Instanzwissen:** Bei der Anfrage *(german, car, caravan)* enthält

WordNet zwar Synonyme für *car*, wie zum Beispiel *automobile* oder *motocar*. Was allerdings für eine erfolgreiche Anfrageausführung benötigt wird, ist Instanzinformation, die aussagt, dass ein *BMW 525i Touring* eine Ausprägung der gesuchten Objektklasse ist.

2. **Mangelnde Abdeckung:** Eine Anfrage nach einer Synagoge in der Nähe von Berlin profitiert nicht von einer WordNet-Expansion des Begriffs Berlin durch *Capital of Germany*. Vielmehr werden in diesem Fall Orte in der Nähe von Berlin benötigt, zusammen mit einem sinnvollen Maß der Ähnlichkeit, also in diesem Fall der Entfernung zu Berlin.

3. **Fehlendes Maß für semantische Ähnlichkeit:** Manchmal ist unklar, wie eine semantische Ähnlichkeit zu definieren ist (Wie ist z.B. die semantische Ähnlichkeit der Namen Meyer, Schmidt und Smith [2]) oder notwendige Daten stehen nicht zur Verfügung.

Oft sind allerdings auch nur Rechtschreibfehler oder Wortumordnungen dafür verantwortlich, dass Begriffe bzw. Zeichenketten nicht als gleich oder ähnlich angesehen werden. Aus diesen Gründen wird in SphereSearch ein hierarchisches Ähnlichkeitskonzept verfolgt, das wie folgt aufgebaut ist (vgl. Abbildung 5.7).

Abbildung 5.7: Ähnlichkeitshierarchie

1. Zunächst wird versucht eine **typspezifische semantische Ähnlichkeit** zwischen Objekten zu bestimmen. Für Ortsangaben könnte dies die geographische Entfernung oder auch die administrative Zugehörigkeit (Provinz, Land, Kontinent) sein. Hieraus wird ersichtlich, dass für die Bestimmung der typspezifischen semantischen Ähnlichkeit sowohl die Domäne exakt bestimmt werden als auch ein domänenspezifischer Vergleicher zur Verfügung stehen muss.

[2] Ist Meyer zu Schmidt ähnlicher als Schmidt zu Smith, weil erstere deutschsprachige Namen sind ?

Abstrakt gesehen entspricht eine Domäne einem spezifischen Datentyp. Deswegen wird in dieser Arbeit Ähnlichkeit als ein an einen Datentyp gebundener Operator angesehen, der zwei Instanzen in Relation setzt.

2. Falls diese Voraussetzungen nicht erfüllt sind, kann versucht werden, eine **allgemeine semantische Ähnlichkeit** zu bestimmen. Eine typische Form der allgemeinen semantischen Ähnlichkeit ist Entfernung zwischen zwei Termen in einem Begriffsgerüst, d.h. in einer Ontologie bzw. einem Thesaurus. Falls allerdings eine oder beide zu vergleichenden Zeichenketten in einer solchen Begriffshierarchie zur Bestimmung der allgemeinen semantischen Ähnlichkeit nicht gefunden werden können, ist auch diese Stufe der Ähnlichkeitsbestimmung nicht anwendbar.

3. Dann kann allerdings als letzte Stufe die **syntaktische Ähnlichkeit** bestimmt werden. Da sich IR primär mit Text beschäftigt, 1ist es naheliegend, die syntaktische Ähnlichkeit zwischen Zeichenketten (Editierdistanz) als syntaktisches Basisähnlichkeitsmaß zu verwenden.

Im folgenden Abschnitt wird die Handhabung von Ähnlichkeit im Kontext von Datentypen formalisiert.

5.2 Datentypen

In typischen IR-Systemen gibt es keine bzw. nur einen einzigen Datentyp: Text. Dies ist auch meist ausreichend, da nur das Vorkommen von Suchbegriffen in einem Text überprüft wird; verschiedene Datentypen wären hier oft keine Hilfe.

Aber auch die meisten XML-Retrieval Systeme unterstützen nicht explizit das Konzept verschiedener Datentypen. Dies liegt insbesondere daran, dass die meisten Daten, die von XML-IR Systemen betrachtet werden, dokumentzentriert und schemalos sind. Auch wenn diese Dokumente an ein Dokumentenschema gebunden sind, handelt es sich hierbei meist um eine DTD, die ebenfalls keine Datentypen unterstützt. Datenzentrierte Dokumente hingegen, die an ein XML-Schema, das eine komplexe Datentyp-Unterstützung aufweist, gebunden sind, sind oft typisiert. Durch ihre meist sehr starre Struktur und Verknüpfung mit spezifischen Applikationen sind sie allerdings meist für IR-Anwendungen uninteressant.

Eine Ausnahme bildet die Arbeit von Fuhr [Fuhr02], der auf die Unterstützung von Datentypen und datentypspezifischen Prädikaten eingeht.

In SphereSearch werden Datentypen explizit unterstützt. Ein fundamentaler Unterschied zu anderen Arbeiten ist, dass in SphereSearch Tag-Namen selbst als Datentypen angesehen werden. So werden die drei folgenden Fälle prinzipiell gleich behandelt:

- Ein XML-Dokument enthält folgendes Fragment: `<Date>02.01.1976</Date>`.

- Ein XML-Dokument enthält folgendes Fragment: `<x>02.01.1976</x>`. Ein verknüpftes XML-Schema spezifiziert, dass das Tag `<x>` vom Typ *date* ist.
- Ein XML-Dokument enthält folgendes Textfragment `02.01.1976`. Ein Annotationstool hat dieses Textfragment als Datumsangabe erkannt.

Ein Beispiel für eine Anfrage, die Typinformation zur Auswertung benötigt, ist die folgende Anfrage:

`Welche deutschen Komponisten wurden zwischen 1800 und 1900 geboren?`

Auf einer relationalen Datenbank würde diese Anfrage eine Teilanfrage auf einer numerischen Spalte der Art `1800<date<1900` beinhalten. Ohne Typisierung wäre für ein IR-System nicht klar, dass die Zeichenketten 1800,1801,1802....1899,1900 Treffer für die Suchanfrage darstellen.

Ein flexibles Datenmodell, das eine Auswertung solcher Anfragen ermöglicht, wird im Folgenden entwickelt. Es ist in Teilen an das von Fuhr vorgestellte Modell [Fuhr02] angelehnt.

Definition 5.2.1 (Datentyp)
Ein Datentyp D ist ein Paar (W_D, O_D)*, wobei* W_D *alle zur Domäne des Datentypen gehörenden Werte umfasst und* $O_D = o_1, ..., o_n$ *eine Menge von Operatoren ist. Jeder Operator implementiert eine Funktion* $p_i : W_D \times W_D \rightarrow [0, 1]$

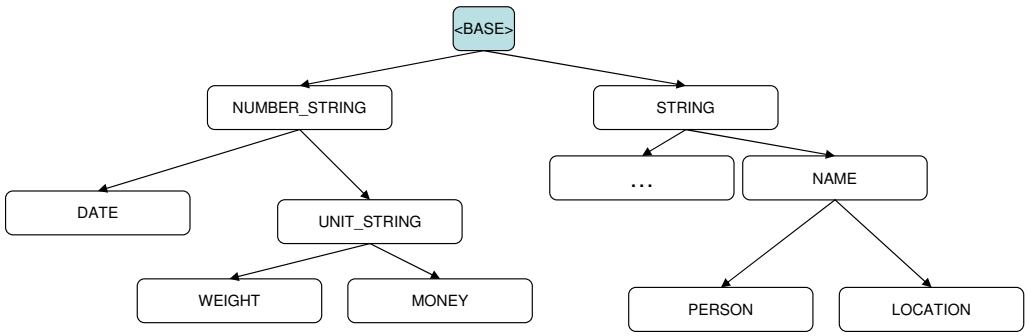


Abbildung 5.8: Datentypen

Beispiele für Funktionen sind $<, >, =, \sim$ etc. Je nach Datentyp wird die Implementierung einer Funktion, d.h. der Operator, unterschiedlich umgesetzt. Für einen =-Operator des Datentyps *Datum* wären die Angaben $17Nov.01$ und 17.11.2001

gleich, wohingegen ein =-Operator des Datentyps Strings Ungleichheit derselben Zeichenketten feststellen würde.

Funktionen von boole'scher Natur, wie z.B. die strikte Gleichheit, haben einen auf {0,1} eingeschränkten Wertebereich. Funktionen, die hingegen eine Ähnlichkeit bestimmen, wie z.B. die ontologische Ähnlichkeit von zwei Begriffen, werden auf das kontinuierliche Intervall $[0, 1]$ abgebildet.

Datentypen sind in der Regel nicht unabhängig voneinander. Insbesondere beim Vergleich von Objekten unterschiedlicher Datentypen ist eine Datentyphierarchie notwendig. So ist in der einfachen in Abbildung 5.8 dargestellten Datentyphierarchie ein Ortsname (*LOCATION*) auch ein *NAME* und ein *NAME* ein *STRING*.

Um Vergleiche von allen Datentypen untereinander zu ermöglichen stehen für jeden Datentyp Operatoren der gleichen Funktionen zur Verfügung. Datentypen implementieren entweder eigene Operatoren oder erben die Operatoren von Supertypen, für deren Funktionen kein typspezifischer Operator zur Verfügung steht.

Abbildung 5.9 zeigt die SphereSearch-Datentyphierarchie[3] mit zugehörigen Operatoren. Die Menge P enthält die Funktionen, die in Form eines geerbten oder implementierten Operators für jeden Datentyp existieren müssen. Die von jedem Datentyp spezifisch implementierten Operatoren sind in der Menge IO enthalten, die inklusive der geerbten Operatorinstanzen für einen Datentyp zur Verfügung stehenden Operatoren sind der Menge O enthalten.

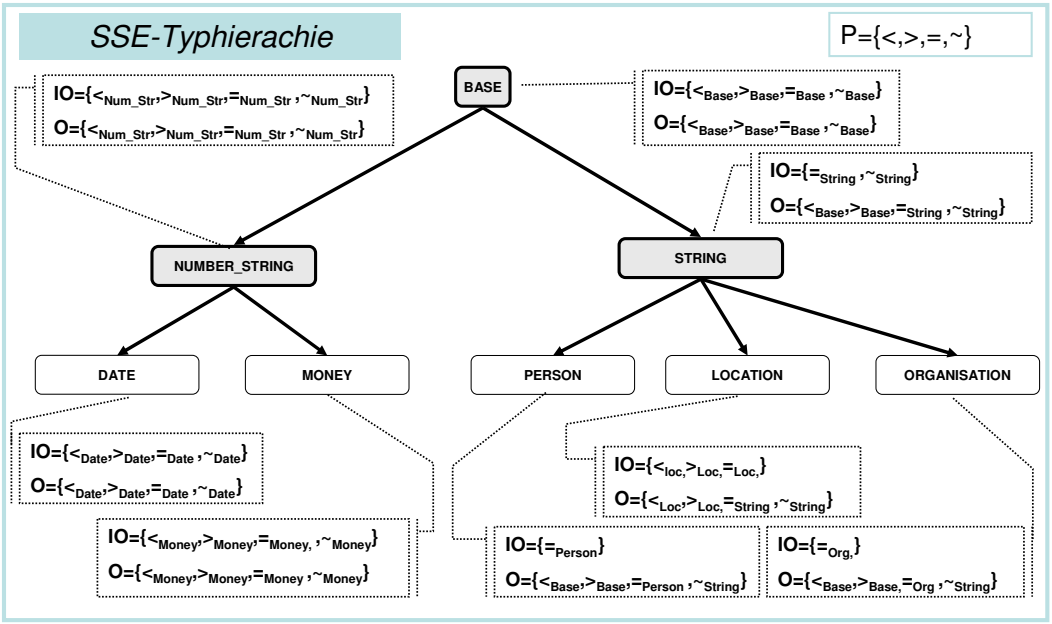


Abbildung 5.9: Datentypen und Operatoren in SphereSearch

[3]Nicht alle der angegebenen Operatoren sind im Prototypsystem tatsächlich umgesetzt

Die transitiven Sub- und Supertyp-Beziehungen von Datentypen untereinander sind wie folgt definiert.

Definition 5.2.2 (Subtyp)
Ein Subtyp D eines Datentyps D' unterscheidet sich von diesem entweder durch seine Domäne, die eine Teilmenge der Domäne von D' ist, oder durch abweichende Operatorinstanzen.
Formal ist die Subtyp-Beziehung $<$ zweier Datentypen D, D' wie folgt definiert:
$D < D' \Leftrightarrow W_D \subseteq W_{D'} \wedge (W_D \neq W_{D'} \vee O_D \neq O_{D'})$.

Die Definition der zur Subtypbeziehung inversen Supertypbeziehung lautet wie folgt.

Definition 5.2.3 (Supertyp)
Ein Typ D ist ein Supertyp des Datentyps D' (Notation: $D > D'$), falls D' ein Subtyp von D ist ($D' < D$).

Aufgrund der inhärenten Komplexität wird Mehrfachvererbung in diesem Modell ausgeschlossen. Aus diesem Grund gilt für die gesamte Typhierarchie die Einschränkung, dass ein Typ zwar beliebig viele direkte Subtypen haben kann, allerdings nur einen direkten Supertyp (Symbol: $>_d$) .

Für jeden Typ D' und seinen direkten Supertyp D (Notation: $D >_d D'$) muss somit gelten:

$D >_d D' \Rightarrow D > D' \wedge \neg\exists D'' : D'' \neq D' \wedge D'' >_d D'$

Die für einen Datentyp D gültigen Operatoren sind wie folgt definiert.

Definition 5.2.4 (Gültige Operatoren)
$P = \{p_1, ...p_n\}$ bezeichnet die Menge der zur Verfügung stehenden Funktionen (Beispiele sind $\equiv, \leq, ...$).

Die Menge der gültigen Operatoren O_D für einen Datentyp D enthält genau eine Implementierung jeder Funktion aus P.

Die Funktion $I : O_D \rightarrow P$ bildet einen Operator o eines Typ D auf die Funktion aus P ab, die er implementiert.

Die Menge $IO_D = \{o_{D,i}, .., o_{D,k}\}$ umfasst alle typspezifischen Operatoren des Datentyps D.

Dann besteht die Menge der gültigen Operatoren für D aus den typspezifischen Operatoren für D (IO_D) und den geerbten Operatoren des Supertyps, für die keine typspezifischen Implementierungen für Datentyp D zur Verfügung stehen, d.h. in IO_D enthalten sind:

$O_D = IO_D \cup \{o | o \in O_{D'} \wedge D' >_d D \wedge I(o) \notin \{I(o_{D,i}), ..., I(o_{D,k})\}\}$
D' ist direkter Supertyp von Datentyp D.

Für die in Abbildung 5.9 dargestellte SphereSearch-Datentyphierarchie müssen nun, den Definitionen von Sub- und Supertypen folgend, die folgenden Bedingungen gelten. Der Basistyp *BASE* muss alle Operatoren implementieren, da er keine Operatoren erben kann. Der Typ *LOCATION* hingegen implementiert beispielsweise nur die Operatoren der Funktionen $<, >$ und erbt die Operatoren für die Funktion $=$ vom Supertyp *STRING*. Da *LOCATION* eigene Operatoren implementiert, muss die Domäne von *LOCATION* nicht mehr zwangsläufig eine echte Teilmenge von *STRING* sein. Die Typen *DATE* und *MONEY* implementieren alle Operatoren.

Die SphereSearch-Datenhierarchie ist um beliebige Typen erweiterbar bzw. veränderbar, nur die Basistypen *BASE*, *STRING* und *NUMBERSTRING* sind statisch.

5.2.1 Anwendung von Operatoren

Falls eine Funktion auf zwei Objekte unterschiedlichen Datentyps angewandt wird, muss ein Operator gewählt werden, der beide Objekte verarbeiten kann. Dies ist nur möglich, falls beide Objekte zur Domäne des Datentyps des gewählten Operators gehören. Dies trifft für alle Operatoren, die zu einem gemeinsamen Vorfahren der verglichenen Objekte gehören, zu. Es wird der Operator des nächsten gemeinsamen Vorfahren gewählt, da dieser der spezifischste Datentyp mit den spezifischsten Operatoren ist. Wird zum Beispiel ein Objekt vom Typ *LOCATION* mit einem Objekt des Typs *PERSON* verglichen (siehe Abbildung 5.9), wird der Operator des Datentyps *STRING* gewählt. Falls ein *LOCATION* mit einem *DATE* Objekt verglichen wird, wird der Operator des Wurzeldatentyps *BASE* gewählt, da dieser der einzige Datentyp ist, der ein gemeinsamer Vorfahre beider Typen ist. Diese Wahl des Operators ist analog zur Handhabung von Vergleichen verschiedener Datentypen in modernen objektorientierten Programmiersprachen ([AC96]).

5.2.2 Die SphereSearch-Typhierarchie

In diesem Abschnitt wird konkret auf die SphereSearch-Datentyphierachie, wie sie in Abbildung 5.9 dargestellt ist, und ihre Abbildung auf die Typhierarchie, die in Abbildung 5.7 dargestellt ist, eingegangen.

Operatoren des *BASE*-Datentyps werden nur angewandt, wenn semantische Operatoren nicht sinnvoll angewandt werden können, wie insbesondere beim expliziten Vergleich von Objekten, bei denen je ein Objekt einen Typ aus dem mit *NUMBER_STRING* wurzelnden Teilbaum und ein Objekt einen Typ aus dem mit *STRING* wurzelnden Teilbaum hat. Ein Beispiel hiefür ist der Vergleich eines Personennamens mit einem Datum. Aus diesem Grund liefern <- und >-Operatoren immer den Wert 0 zurück.

Der Typ *NUMBER_STRING* umfasst Elemente, die aus Zahlen kombiniert mit anderen Zeichen bestehen, wie mit Einheiten verknüpfte Zahlen (10 $US\$$, 5 kg) oder Zeichenketten mit numerischer Interpretation (dritter Mai $\rightarrow$ 3.5.).

Zeichenketten, die nicht explizit zu einem der Datenypen gehören, werden immer als Typ *STRING* behandelt. Dies trifft zu, wenn der Typ nicht in der SphereSearch-Typhierarchie enthalten ist (z.B. bei beliebigen Tags) und bei untypisierten Zeichenketten (z.B. eine Schlüsselwortbedingung $\sim$ *lecture* in einer Anfrage).

Konkret wird die im letzten Abschnitt beschriebene Ähnlichkeitshierarchie auf die Datentyphierarchie bzw. ihre Operatoren wie folgt abgebildet:

1. Wird der Ähnlichkeits- oder Vergleichsoperator eines Typ unterhalb des Typs (NUMBER_)STRING gewählt (LOCATION, PERSON, WEIGHT etc.), handelt es sich hierbei um einen typspezifischen semantischen Operator.

2. Der allgemeine Typ STRING repräsentiert die allgemeine semantische Ebene der Ähnlichkeitshierarchie. Er nutzt die quantifizierte WordNet-Ontologie, um die Ähnlichkeit zu bestimmen. Falls dies nicht möglich ist, wird der syntaktische Ähnlichkeitsoperator des *BASE*-Typs angewandt.

3. Operatorinstanzen des Typs BASE bestimmen ausschließlich die syntaktische Ähnlichkeit.

5.3 Das SphereSearch-Datenmodell

Wir betrachten den gesamten der Suchmaschine bekannten Datenkorpus $\mathcal{X}$ als einen einzigen großen Graph, den Elementgraph $G(\mathcal{X})$, der aus den verlinkten Dokumentgraphen der einzelnen Dokumente des Korpus besteht. Jedes ursprüngliche Dokument wird somit durch einen Teilgraph des Elementgraphs $G(\mathcal{X})$ repräsentiert. Abbildung 5.10 zeigt den Elementgraph eines aus vier Dokumenten bestehenden Korpus.

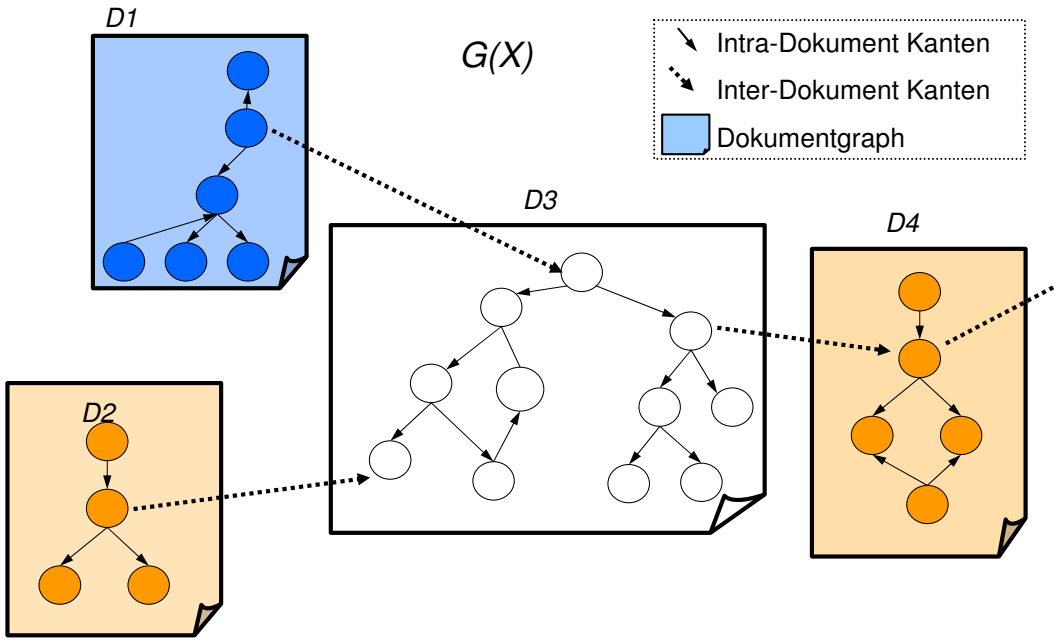


Abbildung 5.10: Elementgraph eines Dokumentkorpus

Ein Dokumentgraph repräsentiert ein Dokument mit seinen zugehörigen Annotationen. Jeder einzelne Knoten eines Dokumentgraphs stellt hierbei den typisierten Teil eines Dokuments oder einer seiner Annotationen dar.

Kanten zwischen einzelnen Knoten innerhalb des Elementgraphs können unterschiedliche Typen haben. Sie verbinden entweder Knoten eines Dokumentgraphs (*Intra-Dokument-Kanten*) oder Knoten verschiedener Dokumentgraphen, also Elemente verschiedener ursprünglicher Dokumente, miteinander (*Inter-Dokument Kanten*).

Zunächst wird der Dokumentgraph eines Dokuments und anschließend, darauf basierend, der Elementgraph eines Dokumentkorpus definiert.

5.3.1 Dokumentgraph

Wie bereits in Kapitel 5.1.2 erläutert bezieht sich jede Annotation auf einen durch ein Intervall spezifizierten Teil des Dokumentinhalts (Definition 5.1.2). Jede Annotation verfügt über einen Typ (z.B. Personenname, Geldbetrag oder Zahl), eine Quelle (z.B. das Annotationssystem GATE) und einen Konfidenzwert (z.B. 0.89). Der Konfidenzwert, eine reelle Zahl zwischen 0 und 1, reflektiert hierbei die erwartete Korrektheit der durchgeführten Annotation.

Falls eine Annotationstechnologie, wie beispielsweise Hidden Markov Models [RC01], einen solchen Konfidenzwert erzeugt, wird dieser übernommen. Falls aber Verfahren eingesetzt werden, die keine Konfidenzwerte generieren, wie beispielsweise die der regelbasierte Ansatz des Jape-Transducers von GATE/ANNIE (siehe Abschnitt 2.3.3), werden statistische Werte über die Korrektheit der Annotation als Konfidenzwert gewählt. So haben veröffentlichte Experimente für ANNIE gezeigt ([MBC03]), dass 89% aller *Person*-Annotationen korrekt sind. Somit wird jeder durch GATE erzeugten Person-Annotation der Konfidenzwert 0.89 zugewiesen.

Jede Annotation eines Teils des Dokumentinhalts betrachten wir als Typzuweisung, d.h. dem durch das Annotationsintervall spezifizierten Teil des Dokuments weisen wir eine *Typisierung* zu. Eine *Typisierung* bezeichnet ein Tripel bestehend aus dem Datentyp, der Quelle der Zuweisung und dem Konfidenzwert.

Einem Intervall können durch Typzuweisungen mehrere Typisierungen bzw. Datentypen zugewiesen werden, da beispielsweise Annotationstools demselben Textfragment verschiedene Annotationen wie z.B. *Wissenschaftler* und *Künstler* zuweisen können. Des weitern können sich in Folge überlappender Annotationen auch die entsprechenden Typzuweisungen überlappen.

Formal definieren wir eine Typzuweisung wie folgt:

Definition 5.3.1 (Typzuweisung)
Eine Typzuweisung $t = (d_i, j, k, T)$ weist dem an Position j beginnenden und an Position k endenden Inhaltsteilstring des Dokumentinhalts von d_i die Typisierung $T = (a, q, c)$ zu. Hierbei bezeichnet a den Typ, q die Quelle und c den Konfidenzwert der Typzuweisung.

Um eine analoge Verarbeitung und Darstellung von Strukturinformation des Ursprungsdokuments (den Tags) und Annotationen zu ermöglichen, werden sowohl Annotationen (z.B. Personennamen oder Geldbeträge) als auch Strukturinformation im selben Typsystem repräsentiert.

In Abschnitt 5.1.2 wurde gezeigt, dass ein Tag auch als Intervallannotation dargestellt werden kann. Somit kann ein Tag auch als Typzuweisung repräsentiert werden. Das Intervall spezifiziert in diesem Fall den Teil des Dokumentinhalts, den

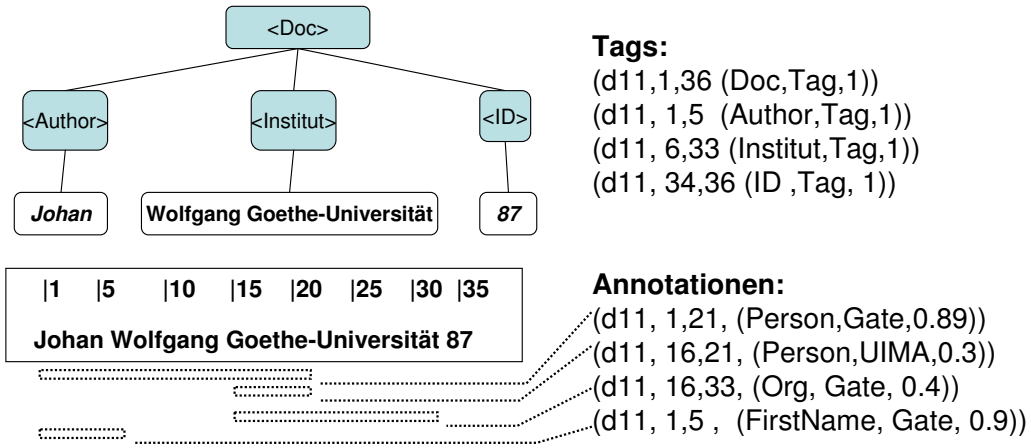


Abbildung 5.11: Dokumentbaum und Typzuweisungen

es umschließt. Der Typ einer solchen Typisierung entspricht dem Tag-Namen und die Quelle wird als „Tag“ spezifiziert. Der Konfidenzwert beträgt immer 1.0.

Abbildung 5.11 zeigt auf der linken Seite den Dokumentbaum eines Dokuments[4] und darunter seinen Dokumentinhalt mit einer Skala zur Bestimmung der Zeichenposition. Auf der rechten Seite der Abbildung sind die Tags des Dokuments als Typzuweisungen angegeben. Zusätzlich sind vier Annotationen angegeben und der Teil des Dokumentinhalts markiert, auf den sich die Annotationen beziehen. Im Dokumentbaum können diese nicht dargestellt werden, da sie sich untereinander überlappen und teilweise auf mehrere Knoten des Dokumentbaums zugleich beziehen.

In Abbildung 5.12 ist der zu den Typzuweisungen zugehörige Dokumentgraph dargestellt, der sowohl das Ursprungsdokument als auch die zugehörigen Annotationen in seiner Graphstruktur berücksichtigt.

Er wird auf Basis der Typzuweisungsmenge T_i , die alle zu einen Dokument D_i gehörenden Typzuweisungen, d.h. sowohl die Tags des Ursprungsdokuments als auch alle Annotationen, umfasst, wie folgt definiert.

[4]Das Beispiel ist bewusst so gewählt, dass es offensichtlich Fehler enthält, allerdings die korrekte Interpretation unklar bleibt.

Definition 5.3.2 (Dokumentgraph)

Der Dokumentgraph $D_{T_i} = (V_i, E_i)$ einer Typzuweisungsmenge T_i eines Dokuments D_i besteht aus einer Knotenmenge V_i und einer Dokumentkantenmenge E_i.

Jedes Intervall aus T_i wird genau durch einen Knoten $x \in V_i$ repräsentiert. $Iv(x)$ bezeichnet das Intervall des Knotens. Der Inhalt eines Knotens ist der Teil des Dokumentinhalts, der durch sein Intervall begrenzt wird. Er wird durch $val(x)$ bezeichnet.

Jede Typzuweisung aus T_i weist dem das Intervall repräsentierenden Knoten die enthaltene Typisierung zu. Einem Knoten können beliebig viele Typisierungen zugewiesen werden. Für einen Knoten x bezeichnet $types(x)$ die Menge aller Datentypen von x. Ferner bezeichnet $conf(x,t)$ den Konfidenzwert (eine reelle Zahl zwischen 0 und 1) und $src(x,t)$ die Quelle der Typzuweisung für Typ t zu x.

Für je zwei Knoten $n_i, n_j \in V_i$ gilt: E_i enthält eine Kante $n_i \rightarrow n_j$ genau dann, wenn das Intervall von n_j vollständig in n_i enthalten ist und es keinen Knoten n_k mit einem Intervall gibt, das im Intervall von von n_i enthalten ist und in dem das Intervall von n_j enthalten ist. Formal:
$(n_i, n_j) \in E_i \Leftrightarrow Iv(n_j) \subset Iv(n_i) \wedge \neg\exists n_k : Iv(n_j) \subset Iv(n_k) \wedge Iv(n_k) \subset Iv(n_i)$

Abbildung 5.12 zeigt den Dokumentgraph D_{T_i} zur Typzuweisungsmenge T_i. Die Annotationen sind als weiß hinterlegte Knoten dargestellt. Es wird nun gezeigt, dass diese Definition des Dokumentgraphs keinesfalls willkürlich gewählt wurde. Dokumentbaum und Dokumentgraph weisen nämlich immer eine starke strukturelle Ähnlichkeit auf, insbesondere existiert meist eine isomorphe Abbildung oder eine Einbettung des Dokumentbaums in den Dokumentgraph.

Im Folgenden wird zunächst davon ausgegangen, dass die betrachteten Dokumente keine unmittelbar geschachtelten Tags, d.h Tags, die den selben Text umschließen[5], beinhalten.

Falls das Dokument keine Annotationen beinhaltet ist ersichtlich, dass der die Tag-Struktur repräsentierende Dokumentbaum isomorph zum Dokumentgraph ist. Dies folgt direkt aus der Definition des Dokumentgraphs.

Abbildung 5.13 zeigt, dass der auf seine Tags beschränkte Dokumentbaum B_1 (grüne Knoten), der keine unmittelbar geschachtelten Tags und keine Annotationen enthält, isomorph zu seinem Dokumentgraph G_1 ist.

[5]Im Beispiel $< a > 1 < b >< c > 2 < /c >< /b >< /a >$ trifft dies für die Tags $< b >$ und $< c >$ zu, da beide den Text im Intervall $[2, 2]$ umschließen.

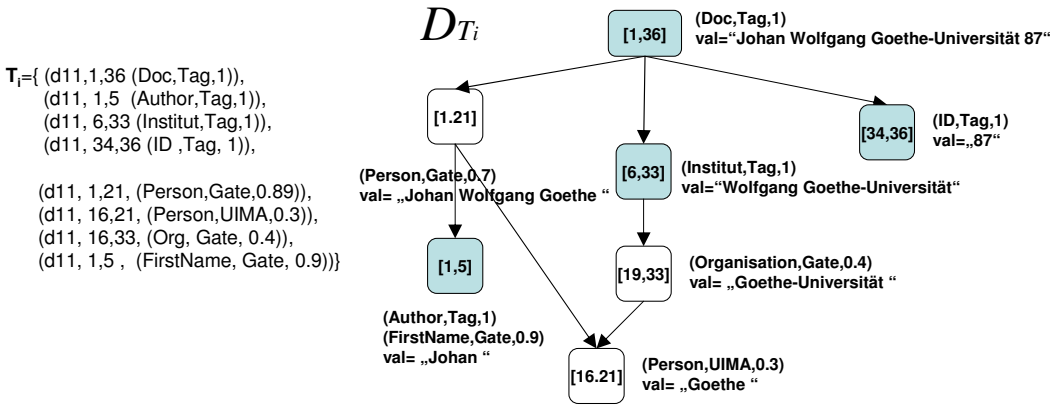


Abbildung 5.12: Dokumentgraph

Falls unmittelbar geschachtelte Tags im Dokument enthalten sind, ist diese Isomorphie nicht mehr gegeben, da Tag-Knoten des Dokumentbaums, die denselben Text umschließen, auf denselben Knoten des Dokumentgraphs abgebildet werden.

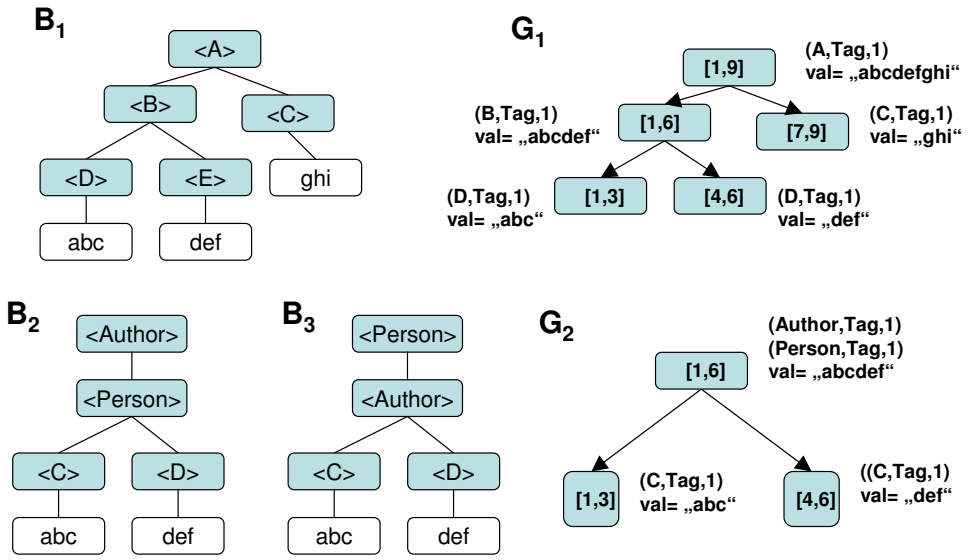


Abbildung 5.13: Abbildung des Dokumentbaums auf seinen Graph

Dies ist ebenfalls in Abbildung 5.13 dargestellt. Die zwei geringfügig unterschiedlichen Dokumentbäume B_2 und B_3 mit unmittelbar geschachtelten Tags werden auf den selben Dokumentgraph G_2 abgebildet. Diese nicht isomorphe Abbil-

dung verschiedener Dokumentbäume auf einen identischen Dokumentgraph ist sogar wünschenswert, da die Reihenfolge direkt geschachtelter Tags, wie in diesem Beispiel *Author* und *Person*, meist willkürlich und somit nicht sinntragend ist.

Für ein Dokument mit Annotationen enthält der Dokumentgraph zusätzliche Knoten und kann somit nicht mehr isomorph zu seinem Dokumentbaum sein. Aber auch in diesem Fall kann man zeigen, dass es eine starke strukturelle Ähnlichkeit zwischen Dokumentbaum und -Graph gibt. Für Dokumente mit Annotationen und ohne unmittelbar geschachtelte Tags existiert eine Einbettung des Dokumentbaums in den Dokumentgraph.

Da die Ordnung, die im Dokumentbaum durch die Abfolge der Kindelemente gegeben ist, im Dokumentgraph durch die den Knoten zugeordneten Intervallen bewerkstelligt wird, ist die Reihenfolge der Kinder im Dokumentgraph nicht relevant. Somit genügt es zu zeigen, dass es eine ungeordnete, d.h. eine nicht die Reihenfolge der Kindknoten erhaltende, Einbettung gibt.

Hierzu erweitern wir dass bei Kilpeläinen ([Kil99]) definierte Problem der Einbettung eines Baums in einen anderen Baum auf das Problem der Einbettung eines Baums, dem Dokumentbaum, in einen gerichteten Graph, den Dokumentgraph. Wir betrachten hierbei den (Dokument-)Baum als gerichteten Graph mit Kanten vom Vater- zu Kindknoten.

Ein Knoten x ist hierbei Vorfahre von Knoten y, falls im zugehörigen gerichteten Graph ein Pfad von x nach y existiert.

Definition 5.3.3 (Einbettung) *Sei $T = (V, E)$ die gerichtete Graphdarstellung eines Baums mit Knotenmenge V und Kantenmenge E und $G = (W, F)$ ein gerichteter azyklischer Graph mit Knotenmenge W und Kantenmenge F. Eine injektive Abbildung der Knoten von T nach G ist dann eine Einbettung von T in G, wenn sie die Vorfahrenbeziehung erhält.*
Für alle Knoten muss gelten:

1. $f(u) = f(v) \Leftrightarrow u = v$
2. *Falls u Vorgänger von v in T ist, dann ist $f(u)$ Vorgänger von $f(v)$ in G*

Wir zeigen nun, dass folgendes Theorem für Dokumente ohne unmittelbar geschachtelte Tags gilt:

Theorem 5.3.1 (Dokumentbaumeinbettung) *Es gibt immer eine Grapheinbettung des Dokumentbaums eines Dokuments ohne unmittelbar geschachtelte Tags in seinen zugehörigen Dokumentgraph.*

Beweisskizze: Alle Knoten des Dokumentbaums umspannen einen unterschiedlichen Teil des Dokumentinhalts. Somit enthält der Dokumentgraph per Definition exakt einen Knoten für jeden Knoten des Dokumentbaums. Die Funktion f bildet den Knoten des Dokumentbaums B auf den entsprechenden Knoten des Dokumentgraphs G ab. Somit ist Bedingung 1 erfüllt.

Falls ein Knoten u in B Vorgänger von v ist, umschließt v einen Teil des von u umschlossenen Dokumentinhalts. Somit ist das $f(v)$ zugeordnete Intervall eine echte Teilmenge des $f(u)$ zugeordneten Intervalls.

Im Dokumentgraph gibt es per Definition eine Kante von Knoten a nach b, wenn das Intervall von b in a enthalten ist und es keinen Knoten c „zwischen“ a und b gibt, d.h. dessen Intervall in a enthalten ist und in dem das Intervall von b enthalten ist: $\neg\exists c : Iv(c) \subset Iv(a) \wedge Iv(b) \subset Iv(c)$

Somit gibt es auch immer einen Pfad von einem Knoten $f(u)$ nach $f(v)$, falls das Intervall von $f(u)$ in $f(v)$ enthalten ist und ein Pfad von u nach v in B existiert [6]. Somit ist auch Bedingung 2 erfüllt.

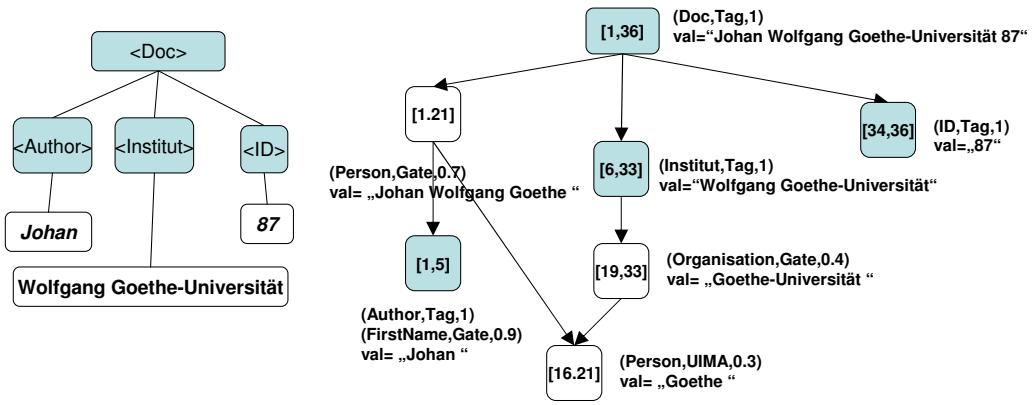


Abbildung 5.14: Dokumentbaumeinbettung

Abbildung 5.14 zeigt noch einmal den Dokumentbaum und Dokumentgraph des letzten Beispiels. Es ist zu sehen, dass der Dokumentbaum in Graph eingebettet ist.

[6]Dies kann leicht über einen Induktionsbeweis über die Pfadlänge gezeigt werden

Im Falle von unmittelbar geschachtelten Tag werden diese, da sie dasselbe Intervall umspannen, analog zum Fall von Dokumenten ohne Annotationen auf den selben Knoten abgebildet. Dies ist, wie schon zuvor erwähnt, sogar wünschenswert, da somit Dokumente mit willkürlich unterschiedlicher Schachtelung ohne semantischen Unterschied auf den selben Dokumentgraph abgebildet werden.

5.3.1.1 Redundanzbereinigter Dokumentgraph

Die inhaltliche Redundanz im Dokumentgraph folgt aus der Tatsache, dass der Inhalt jedes Knotens ein Teil des Inhalts seiner Vorgängerknoten ist. Die Vorkommenshäufigkeit des Inhalts eines Terminalknotens (einem Knoten ohne ausgehende Kante) entspricht somit mindestens der Länge des Pfades zum das Wurzelelement des Dokuments repräsentierenden Knoten.

Da bei der Auswertung von Anfragen diese Redundanz das Ergebnis verfälschen würde (siehe Kapitel 7) ist es erforderlich, dass der Dokumentgraph zunächst redundanzbereinigt wird.

Der *redundanzbereinigte Dokumentgraph* wird nun wie folgt konstruiert. Beginnend von allen Terminalknoten wird der Inhalt der Knoten aus den Vorgängerknoten entfernt.Abstrakt gesehen wird jeder Teil des Inhaltes, der sich im Inhalt eines Kindes wiederholt, durch eine Referenz auf diesen ersetzt. Abbildung 5.15 zeigt den zugehörigen redundanzbereinigten Dokumentgraph des in Abbildung 5.12 dargestellten Dokuments.

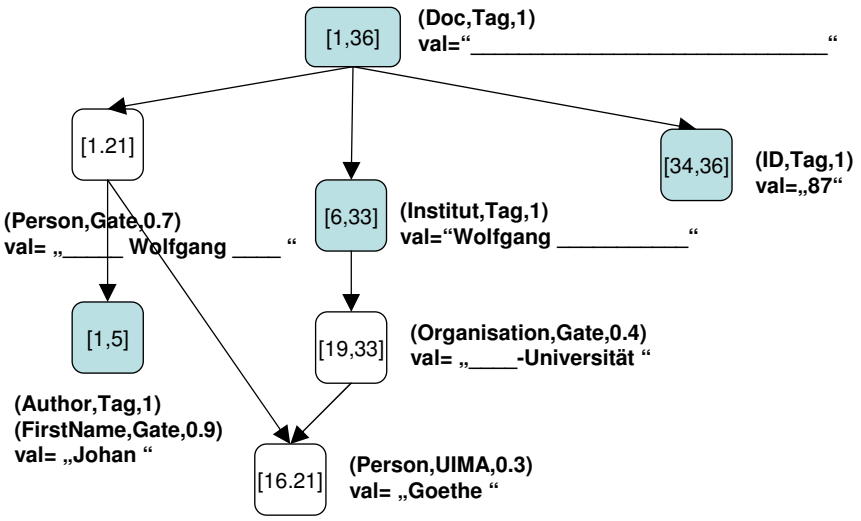


Abbildung 5.15: Redundanzbereinigter Dokumentgraph

5.3.2 Elementgraph

Auch wenn wir für die Konstruktion des Dokumentgraphs die Richtung der Kanten benötigen, wird diese bei der Auswertung von Anfragen nicht berücksichtigt. Dies liegt darin begründet, dass für den Informationssuchenden die Linkrichtung innerhalb von Ergebnissen nicht relevant und somit ihre Bewertung in Hinblick auf die Ergebnisauswertung unklar ist.

So ist bei einer Anfrage nach einem Schauspieler, der in einem bestimmten Film mitgespielt hat, ein Treffer, bei dem ein Link vom entsprechenden Filmdokument zum Schauspielerdokument zeigt, nicht schlechter als ein Treffer mit einem Link vom Schauspieler zu dem Film, in dem er mitgespielt hat.

Selbst auf Dokumentebene kann diese Information zentriert auf den Film oder auf den Schauspieler modelliert sein. Im ersten Fall wäre der Schauspieler ein Kindelement des Films, im zweiten Fall der Film ein Kindelement des Schauspielers. Auch dieser Fall zeigt, dass eine qualitative Bewertung der Kantenrichtung nicht sinnvoll ist.

Aus diesem Grund betrachten wir den Elementgraph, der sich aus den einzelnen Dokumentgraphen zusammensetzt, als ungerichteten Graphen. Der Elementgraph ist wie folgt definiert.

Definition 5.3.4 (Elementgraph)

Eine Dokumentsammlung $\mathcal{X} = (\mathcal{D}, \mathcal{L})$ besteht aus einer Menge von Dokumenten $\mathcal{D}$ und einer Menge von Links $\mathcal{L}$ zwischen den Dokumenten.

Der zur Dokumentsammlung $\mathcal{X}$ zugehörige Elementgraph $G(\mathcal{X}) = (V(\mathcal{X}), E(\mathcal{X}))$ ist ein ungerichteter Graph, bestehend aus der Knotenmenge $V(\mathcal{X})$, der Vereinigung der Knoten aller Dokumentgraphen für Dokumente in $\mathcal{D}$, und der ungerichteten Kantenmenge $E(\mathcal{X})$, die alle Kanten aller Dokumentgraphen und alle Links $\mathcal{L}$ enthält.

Für jede Kante $e \in E(\mathcal{X})$ bezeichnet weight(e) das Gewicht der Kante (eine nichtnegative reelle Zahl zwischen 0 und 1).

5.3.2.1 Kantengewichte

Wie in Definition 5.3.4 festgelegt, verfügen alle Kanten über ein spezifisches Gewicht. Dieses Kantengewicht ist insbesondere für die Anfrageauswertung wichtig. Es kann sowohl die physische Distanz (Pfadlänge) als auch die semantische Ähnlichkeit seiner Endpunkte reflektieren. Beim Aufbau des Graphs haben alle Intra-Dokument-Kanten, die direkt aus dem Ursprungsdokument stammen, Gewicht w_i sowie Links zwischen Dokumenten Gewicht w_d. Kanten, die durch eine

Einfügeoperation eines Annotationsprozesses eingefügt werden, haben stets Gewicht 0, da eine Annotation nicht die Struktur ändert, sondern nur Information ergänzt. Die Gewichte w_i und w_d sind implementations- bzw. anwendungsabhängig zu wählende Parameter. Das Standardgewicht, das in Experimenten genutzt wurde, ist für Intra-Dokument-Kanten 1 und für Links 2.

6 Anfragesprache

6.1 Anforderungen an die Anfragesprache

Einerseits weisen semi-strukturierte, teilweise verlinkte Dokumente vielfältige Strukturinformationen auf (Tags, Attribute, Links), die explizit in einer Anfragesprache unterstützt werden können, andererseits kann bei einem Benutzer, der Information in einem heterogenen Dokumentenkorpus sucht, keinerlei Wissen über die tatsächliche Struktur der Dokumente vorausgesetzt werden.

Aus diesem Grund sind sowohl XML-IR-Anfragesprachen, da sie exaktes Strukturwissen voraussetzen, als auch klassische IR-Keyword-Anfragen, da diese keinerlei Strukturinformation in der Anfrage berücksichtigen, nicht angemessen.

Aus diesem Grund musste eine einfache, neuartige Anfragesprache konzipiert werden. Dabei sollten folgende Aspekte und Anforderungen berücksichtigt werden:

- Eine Anfrage soll als reine Keyword-Anfrage formuliert werden können. Alle weiteren Sprach-Features sollen optional sein.
- Falls der Benutzer Wissen über Struktur und Kontext seiner Anfrage hat, soll er dies spezifizieren können (Kontextbewusstsein).
- Die Anfragesprache soll eine konzeptspezifische [1] Formulierung und Auswertung von Anfragen ermöglichen (Konzeptbewusstsein).
- Semantische Ähnlichkeitsbedingungen, deren Interpretation domänenabhängig ist, sollen unterstützt werden (Abstraktionsbewusstsein).

Die hier vorgestellte Sprache soll nicht unbedingt direkt von einem Endbenutzer zur Anfrageformulierung genutzt werden. Vielmehr soll sie durch ihre einfache Struktur auch die Möglichkeit geben, Anfragen intuitiv über ein graphisches Benutzer-Interface zu spezifizieren.

6.2 Überblick

Die SphereSearch-Anfragesprache SSL (**S**phere **S**earch **L**anguage) kombiniert konzeptbewusste Keyword-Anfragen mit Erweiterungen für abstraktionsbewusste Ähnlichkeitsanfragen und eine kontextbewusste Anfrageformulierung.

[1] Ein Konzept bezeichnet hierbei einen spezifischen Typ, wie z.B. Ort, Person, Datum.

Das folgende Beispiel einer Anfrage dient dazu alle Sprachelemente der SSL anschaulich zu erläutern. Hierzu wird eine zunächst einfache Anfrage nach und nach um weitere Sprachkonstrukte ergänzt. Die Anfrage

„ Welcher deutsche Professor hält eine Vorlesung über Datenbanken und ist an einem Projekt über XML beteiligt“

kann mit einer reinen Schlüsselwortanfrage wie folgt formuliert werden:

```
Q(professor, germany, course, databases, project, xml)
```

Die Anfrage ist eine zulässige SSL-Anfrage und kann vom System ausgewertet werden. Allerdings ist offensichtlich, dass diese Anfrage nicht optimal ist, da dem System Information darüber fehlt, in welcher Relation die Begriffe zueinander stehen. Somit wird es Treffer über XML-Vorlesungen und Datenbankprojekte genauso gut bewerten wie die eigentlich gesuchten Ergebnisse über XML-Projekte und Datenbankvorlesungen.

Aus diesem Grund können Anfragebedingungen, die semantisch zueinander gehören bzw. sich auf dasselbe Objekt beziehen, in mehrere Anfragegruppen gruppiert werden. Die gruppierte, und somit jetzt kontextbewusste Anfrage, die dem System mehr Informationen zu einer der Anfrage angepassteren Ergebnisbewertung gibt, sieht aus wie folgt:

```
A(professor, Germany)
B(course, databases)
C(project, XML)
```

Die einzelnen Gruppen werden hierbei mit einem Buchstaben benannt, um eine spätere Referenzierung zu ermöglichen. Nun repräsentieren die Anfragegruppen drei Objekte: (einen Professor aus Deutschland), (eine Vorlesung über Datenbanken) und (ein Projekt über XML).

Für die Beispielanfrage gibt es allerdings noch viele potentielle Ergebnisse, die nicht exakt die Begriffe Professor, sondern Dozent, nicht XML, sondern „semistructured data“ und nicht Germany, sondern eine Stadt in Deutschland enthalten.

Mit Hilfe des Ähnlichkeitsoperators $\sim$ kann die Anfrage wie folgt formuliert werden:

```
A(professor, ∼Germany)
B(course, ∼databases)
C(project, ∼XML)
```

Die Benutzung des Ähnlichkeitsoperators in Kombination mit den Begriffen `XML`, `database` und `Germany` veranlasst die Expansion der Anfrage mit ähnlichen, aus einer quantifizierten Ontologie stammenden, Begriffen. Eine so erweiterte Anfrage liefert auch Ergebnisse über eine Vorlesung über Informationssysteme und ein Projekt über semistrukturierte Daten mit einer der Ähnlichkeit der Begriffe (vgl. Kapitel 5.2.2) angepassten Ergebnisbewertung zurück. Die Expansion von `Germany` erweitert die Anfrage um Begriffe, wie z.B. „european country“ und „european nation“, die zwar korrekt sind, aber im Fall dieser Anfrage nicht weiterhelfen.

Hat der Benutzer entweder latentes Wissen über die zugrunde liegende Struktur der Daten oder den Typ der gesuchten Information, kann er dies über eine Konzept-Wert Bedingung spezifizieren. So kann die Bedingung $\sim$ *Germany* erweitert werden zu einer mit dem Ähnlichkeitsoperator kombinierten Konzept-Wert-Bedingung *location* $=\sim$ *Germany*.

Die Angabe von *location* $=\sim$ *Germany* an Stelle von $\sim$ *Germany* gibt dem System in diesem Fall die Möglichkeit die domänenspezifische Auswertung für das Konzept *location* an Stelle der generischen Ähnlichkeitsauswertung anzuwenden um zu ermitteln, ob Ortsangaben in potentiellen Treffern in Deutschland liegen.

Im Falle solcher Datentypen, für die das System eine besondere Unterstützung aufweist, sind auch Bereichsanfragen möglich. Möchte der Benutzer die Beispielanfrage auf den Bereich zwischen Saarbrücken und Kaiserslautern einschränken, kann er dies über eine Bereichsanfrage für die Konzept-Wert-Bedingung erreichen:

```
A(professor, location=Saarbrücken-Kaiserslautern)
B(course, ∼databases)
C(project, ∼XML)
```

In diesem Fall würde das System die Koordinaten für Saarbrücken und Karlsruhe in einer internen Geo-Datenbank nachschlagen und um alle Orte, die im in Abbildung 6.1 dargestellten Bereich enthalten sind, expandieren.

Obwohl diese modifizierte Anfrage deutlich mehr Treffer mit einen besseren Ranking zurückliefert, bleiben nach wie vor viele potentielle Ergebnisse unentdeckt. Ein Grund hierfür ist eine fehlende Verknüpfung der Teilergebnisse untereinander. So sind bei vielen potentiellen Ergebnissen für diese Anfrage Projektseiten oft nicht mit den Homepages beteiligter Personen verknüpft. Da SphereSearch zunächst nur die physikalische Linkstruktur ausnutzt werden solche Treffer nicht gefunden. Allerdings kann dem System weiteres Wissen gegeben werden um „semantische Links“ auszunutzen. Hierzu kann eine Join-Bedingung spezifiziert werden, die angibt, dass eine Person (bzw. ein erkannter Personenname), die in der Nähe von Treffern für Anfragegruppe A gefunden wurde, identisch zu einer Person, in der Nähe von Anfragegruppe C sein soll. Das System berücksichtigt nun auch „virtuelle Links“, die

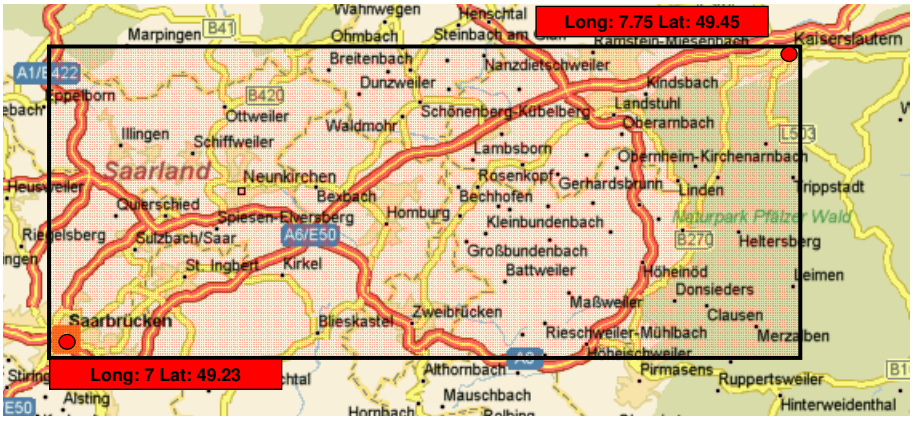


Abbildung 6.1: Ortsbereichsanfrage

auf der Ähnlichkeit der Join-Attribute basieren, zurückgeben. Die vollständige Anfrage unter Ausnutzung aller Sprachkonstrukte lautet wie folgt:

```
A(professor, location=Saarbrücken-Kaiserslautern)
B(course, ~databases)
C(project, ~XML)
A.person=C.person
```

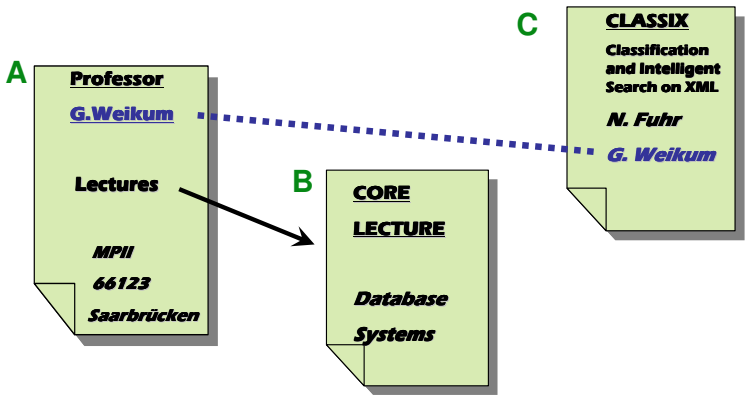


Abbildung 6.2: Beispielanfrage

Abbildung 6.2 zeigt ein Beispiel mit drei Web-Seiten, die sich für die Anfragegruppen A, B und C qualifizieren. Der Treffer für A beinhaltet einen Link zur Vorlesungsseite B. Die drei Seiten sind allerdings zunächst kein Treffer für die Anfrage, da sie nicht miteinander verknüpft sind. Wird nun zusätzlich die Bedingung

`A.person=C.person` hinzugefügt, werden der Graphstruktur weitere Links, und zwar zwischen Person-Objekten mit gleichem Inhalt, hinzugefügt Da die Graphstruktur nun einen Link zwischen den Treffern für Gruppe A und B aufweist, sind die drei Seiten eine potentielle Antwort für die Anfrage.

6.3 Die Sprachkonstrukte

Die Grammatik ist in der *Erweiterten Backus-Naur-Form* (EBNF) angegeben. Die Bedeutung der in den Produktionsregeln verwendeten EBNF-Sprachkonstrukte ist in Tabelle 6.1 angegeben.

Konstrukt	Bedeutung
'...'	Terminalsymbol
(...)	Klammerung
[...]	Optionalität
{...}-	beliebig viele, mindestens jedoch ein Vorkommen
{...}	beliebig viele (auch kein) Vorkommen
|	Alternative
,	Konkatenation
;	Abschluss einer Regel
=	Definitionssymbol

Tabelle 6.1: EBNF-Konstrukte (Ausschnitt)

```
A = ['1'],B, {'3'|'4'|'5'}, {'6'|'7'|'8'}-;
B = '2';
```

$\mathcal{X} = \{12345678, 2887\}$
$\mathcal{Y} = \{12345, 123444\}$

Abbildung 6.3: EBNF-Beispiel

Die durch die in Abbildung 6.3 dargestellte, aus den zwei Produktionsregeln A und B bestehenden, Beispielgrammatik erzeugten Zeichenketten beginnen optional mit einer 1, gefolgt von einer 2 (Regel B), gefolgt von beliebig vielen Vorkommen der Zahlen 3,4 oder 5 und enden mit beliebig vielen, aber mindestens einem Vorkommen der Zahlen 6,7 oder 8. Die Mengen $\mathcal{X}$ bzw. $\mathcal{Y}$ enthalten Zeichenketten, die durch die Grammatik erzeugt bzw. nicht erzeugt werden können. Zur detaillierten Informationen zur EBNF sei auf die ISO-Norm 14977 ([EBNF]) verwiesen.

6.3.1 Anfrage

Eine Anfrage besteht aus einer oder mehreren Anfragegruppen und optionalen Join-Bedingungen. Jede Anfragegruppe setzt sich aus elementaren Bedingungen (Schlüsselwort oder Konzept-Wert-Bedingungen) zusammen. Eine Variable kennzeichnet jede Gruppe.

```
Query = QGroup, {' ', QGroup}, {' ', JCond};
```

6.3.2 Anfragegruppen

Anfragegruppen sind ein optionales Sprachmittel, um Schlüsselwort und Konzept-Wert Bedingungen, die sich auf dasselbe Objekt beziehen, zu gruppieren. Eine Gruppierung durch den Benutzer drückt seine Erwartung aus, dass Terme aus einer Anfragegruppe kompakter [2] repräsentiert sind, als Terme unterschiedlicher Gruppen. Das folgende Beispiel ist eine Anfrage nach einem Film, der in Italien spielt, und einem Gouverneur aus Kalifornien.

```
A(Governor, California)
B(Movie, Italy)
```

Durch die Gruppierung wird ausgedrückt, dass ein Treffer, in dem Governor und California, sowie Movie und Italy in unmittelbarer Nähe voneinander gefunden werden, besser ist als ein vergleichbares Ergebnis mit einer anderen lokalen Häufung der Suchbegriffe (z.B. Governor und Italy, sowie Movie und California). Anfragegruppen werden durch einen Buchstaben benannt und können somit in anderen Teilanfragen, z.B. einer Join-Bedingung, referenziert werden.

```
QGroup = VAR, '(', ElemCondition, {',', ElemCondition}, ')';
```

6.3.3 Der Ähnlichkeitsoperator

Der Ähnlichkeitsoperator $\sim$ kann mit allen elementaren Bedingungen (Schlüsselwort-, Konzept- und Wertbedingungen) kombiniert werden. Wird er spezifiziert, versucht das System zusätzlich zu exakten auch semantisch ähnliche Treffer zu ermitteln.

[2] Die Interpretation von Kompaktheit in diesem Kontext ist in Kapitel 7.2.1 beschrieben. Kompakte Treffer weisen eine geringe Ausdehnung im Hinblick auf ihren Durchmesser in der internen Graphrepräsentation auf.

Das Maß der Ähnlichkeit wird in der Bewertung des Treffers und somit im Ranking berücksichtigt.

Das System wählt kontext- und typabhängig (vgl. Kapitel 5.2 und 5.2.2)eine Auswertungstrategie, um „ähnliche“ Treffer zur ermitteln. Beispiele und Details für den Ähnlichkeitsoperator im Kontext verschiedener Sprachkonstrukte sind in den folgenden Abschnitten angegeben.

6.3.4 Schlüsselwortbedingungen

Schlüsselwortbedingungen in SSL unterscheiden sich nicht von Schlüsselwortbedingungen in Standard-Web-Suchmaschinen. Sie können aus einzelnen Worten oder Phrasen beliebiger Länge, optional kombiniert mit dem Ähnlichkeitsoperator, bestehen. Die folgende Anfrage besteht aus drei Schlüsselwortbedingungen, zwei Phrasen und einer mit dem Ähnlichkeitsoperator kombinierten Schlüsselwortbedingung.

```
Q(Saarland University, Institute for computer science, ∼ XML)
```

Durch den Ähnlichkeitsoperator wird die Teilanfrage $\sim XML$ um ähnliche Begriffe erweitert und intern in die folgende Anfrage expandiert:

```
XML, semistructured data, markup language, DTD.
```

```
KWCond = ExactValue | SimilarValue;
ExactValue = String;
SimilarValue = '∼', String;
String = {'a'|..|'z'|'A'|..|'Z'|'0'|..|'9|' '}-;
```

Somit wird im Kontext von Schlüsselwortbedingungen nur der allgemeine semantische WordNet-basierende Ähnlichkeitsoperator eingesetzt, da keine weitere Typinformation zur Verfügung steht. Falls der semantische Ähnlichkeitsoperator nicht angewandt werden kann, wird keine Expansion durchgeführt[3].

6.3.5 Konzept-Wert-Bedingungen

Eine Konzept-Wert Bedingung (KW-Bedingung) besteht aus einer Konzept- und einer Wertteilbedingung. Wie auch die für die Schlüsselwortbedingung können für

[3]Syntaktische Ähnlichkeit (Editierdistanz) kann in diesem Kontext nicht sinnvoll zur Anfrageexpansion genutzt werden.

Teilbedingungen Terme oder Phrasen, kombiniert mit dem Ähnlichkeitsoperator, angegeben werden. Beispiele für KW-Bedingungen sind:

```
author=weikum
location=∼Cologne
∼location=∼Cologne
related work=unsupervised clustering
```

Auf das Datenmodell bezogen, bezieht sich die Konzeptteilbedingung auf den Typ eines Knotens und die Wertteilbedingung auf dessen Inhalt.

Zusätzlich kann für eine Wertbedingung auch ein Intervall angegeben werden. Voraussetzung hierfür ist, dass das System für das spezifizierte Konzept (den Datentyp) Intervallauswertung explizit unterstützt. Beispiele für KW-Bereichsanfragen sind:

```
price=50−80
location=Salvador - Rio de Janairo
date=08/20/05-08/26/05
```

Wird der Konzeptteil einer KW-Bedingung mit dem Ähnlichkeitsoperator kombiniert, wird zunächst analog zu Schlüsselwortbedingungen eine allgemeine semantische Expansion durchgeführt. Die Anfrage $\sim location =\sim Cologne$ würde somit auch einen Treffer $place = Bonn$ (Knotentyp:place;Inhalt:Bonn) zurückliefern, falls die Expansion von *location* den Begriff *place* beinhaltet.

Wird der Ähnlichkeitsoperator auf die Wertbedingung angewandt, wird der mit dem Typ der Konzeptbedingung verbundene Ähnlichkeitsoperator angewandt.

Wichtig in diesem Zusammenhang ist, dass immer der Ähnlichkeitsoperator der spezifizierten Konzeptbedingung angewandt wird, auch wenn durch die Expansion der Konzeptbedingung weitere Typen hinzukommen. Im obigen Beispiel würde somit der objektspezifische Ähnlichkeitsoperator für *location* für den Vergleich von *Bonn* und *Cologne* benutzt.

```
CVCond = (SimilarValue|ExactValue), '=', (ExactValue | SimilarValue
| RangeValue);
RangeValue = (SimilarValue|ExactValue), '=', ExactValue, '-',
ExactValue;
```

6.3.6 Join-Bedingungen

Join-Bedingungen können in SSL analog zu SQL Join-Bedingungen Informationen, die nicht direkt miteinander verbunden sind auf Basis eines Attributs miteinander verknüpfen. Eine Join-Bedingung ist symmetrisch und besteht aus zwei Join-Partnern, die durch eine Vergleichsoperator miteinander verbunden sind. Als Vergleichsoperatoren stehen Gleichheit(=) und Ähnlichkeit(∼) zur Verfügung. Ein Join-Partner besteht aus einem Anfragegruppennamen und einem Join-Attribut.

Beispiele für Join-Bedingungen sind:

```
A.person=B.person
C.ort=d.stadt
e.ort f.datum
```

Das Join-Attribut bezeichnet hierbei den Datentyp, dessen Inhalt abhängig vom spezifizierten Vergleichsoperator verglichen werden soll. Wird auf das Join-Attribut der Ähnlichkeitsoperator angewandt, wird analog zur KW-Bedingung eine Expansion durchgeführt und anschließend der Vergleichsoperator des spezifzierten Typs angewandt.

Typischerweise handelt es sich beim Join-Attribut um einen ursprünglichen Tag-Namen oder einen Datentyp, der durch einen Annotationsprozess erzeugt wurde. Die folgende Anfrage vergleicht somit paarweise die Vorkommen von Personennamen auf Seiten, die jeweils ein Treffer für Anfragegruppe A bzw. ein Treffer für Anfragegruppe C sind.

```
A(professor, location=Saarbrücken-Karlsruhe)
C(project, ∼ XML)
A.person=C.person
```

Falls der Ähnlichkeitsoperator ∼ als Vergleichsoperator benutzt wird, wird den verglichenen Datentypen entsprechend ein typspezifischer Vergleichsoperator gewählt (vgl. Kapitel 5.2).

```
JCond = Var, '.', (SimilarValue|ExactValue), ('∼'|'='), Var, '.',
(SimilarValue|ExactValue);
```

6.4 Sprachsyntax

Formal ist eine SphereSearch-Anfrage spezifiziert wie folgt:

Definition 6.4.1 (SphereSearch-Anfrage)

Eine SphereSearch-Anfrage $S = (Q, J)$ *besteht aus einer Menge* $Q = \{G_1, \ldots, G_g\}$ *von* Anfragegruppen *und einer (möglicherweise leeren) Menge* $J = \{J_1, \ldots J_m\}$ *von Join-Bedingungen.*

Jede Anfragegruppe Q_i *selbst besteht aus einer (möglicherweise leeren) Menge von Schlüsselwortanfragen* $t^i_1 \ldots t^i_{k_i}$ *und einer (möglicherweise leeren) Menge von Konzept-Wert-Bedingungen* $c^i_1 = v^i_1 \ldots c^i_{l_i} = v^i_{l_i}$.

Schlüsselwortanfragen und die Bestandteile von Konzept-Wert-Bedingungen sind atomare Bedingungen. Eine atomare Bedingung besteht aus einer Zeichenkette (die Leerzeichen enthalten kann) und einem vorangestellte optionalen Ähnlichkeitsoperator ($\sim$*).*
Eine Wert-Bedingung kann außerdem als Bereichsbedingung der Form $v_1 - v_2$*, wobei* v_1 *und* v_2 *atomare Bedingungen sind, spezifiziert werden.*

Zusätzlich können exakte oder Ähnlichkeits-Join-Bedingungen zwischen Anfragegruppen spezifiziert werden. Eine Join-Bedingung hat die Form $Q_i.v = Q_j.w$ *im Falle einer exakten Join-Bedingung und* $Q_i.v \sim Q_j.w$ *im Falle eines Ähnlichkeits-Joins.* Q_i,Q_j *sind hierbei Anfragegruppen und v,w Begriffe (typischerweise Tag-Namen oder Annotationensbezeichner, die durch den Annotationsprozess hinzugefügt wurden).*

Die vollständige Grammatik der SphereSearch-Anfragesprache SSL ist in Abbildung 6.4 angegeben.

```
(* Anfrage *)
Query = QGroup, {' ', QGroup}, {' ', JCond};
(* Anfragegruppen *)
QGroup = VAR, '(', ElemCondition, {',', ElemCondition}, ')';

(* Elementare Bedingungen *)
ElemCondition = KWCond | CVCond;
KWCond = ExactValue | SimilarValue;
CVCond = (SimilarValue|ExactValue), '=', (ExactValue | SimilarValue
| RangeValue);
(* Join-Bedingungen *)
JCond = Var, '.', (SimilarValue|ExactValue), ('∼'|'='), Var, '.',
(SimilarValue|ExactValue);
(* Exakte Bedingung *)
ExactValue = String;
(* Ähnlichkeitsbedingung *)
SimilarValue = ''∼'', String;
(* Bereichsbedingung *)
RangeValue = (SimilarValue|ExactValue), '=', ExactValue, '-',
ExactValue;

(* Variablen *)
Var = 'A' | ... | 'Z';
(* Zeichenketten *)
String = {'a'|..|'z'|'A'|..|'Z'|'0'|..|'9|' '}-;
```

Abbildung 6.4: SSL Grammatik

7 Semantik und Ranking

Das Ergebnis einer SphereSearch-Anfrage ist ähnlich wie bei den meisten Suchmaschinen eine nach (berechneter) Relevanz sortierte Resultatsliste. Allerdings ist jedes einzelne Resultat einer SphereSearch-Anfrage kein Dokument, wie z.B. bei Web-Suchmaschinen, oder ein einzelnes XML-Element, wie in den meisten XML-Suchmaschinen, sondern ein Tupel, das aus je einem Element pro Anfragegruppe besteht.

Abbildung 7.1 zeigt den Elementgraph von sieben teilweise verlinkten Web Seiten.

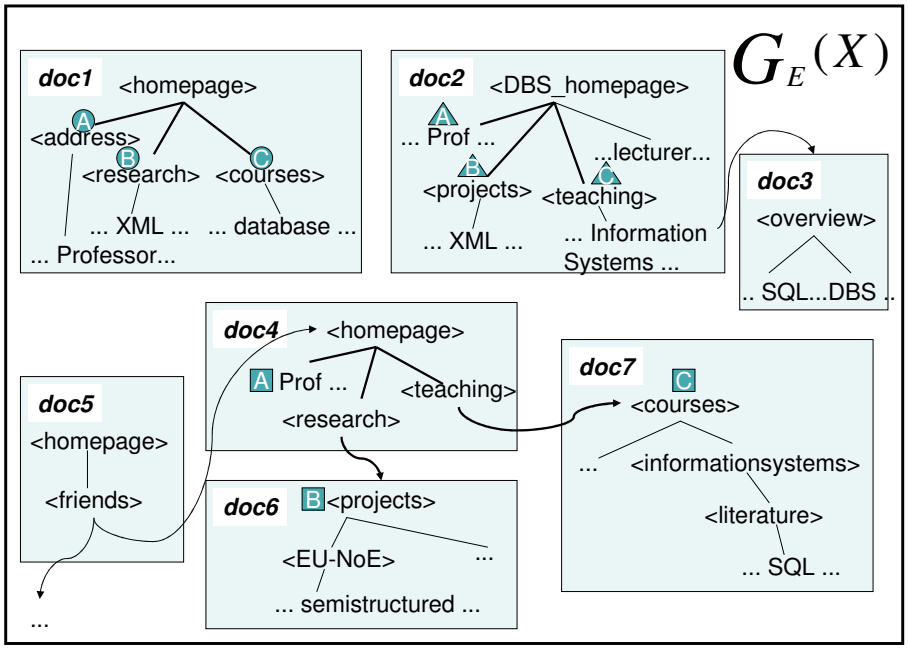


Abbildung 7.1: Dokumentkorpus

Für die Anfrage: $A(professor)\ \ B(research, \sim XML)\ \ C(course, \sim databases)$ würde eine herkömmliche Suchmaschine nur Dokument *doc1* zurückliefern, da nur dieses alle gesuchten Begriffe enthält. SphereSearch hingegen liefert eine ganze Liste von Treffern, die jeweils aus einer Menge von Knoten, die über Elementgraphkanten miteinander verbunden sind, besteht. Die Anzahl der Knoten pro Ergebnis entspricht der Anzahl der Anfragegruppen. In der Abbildung sind drei dieser Treffer

mit Symbolen markiert (Kreis, Dreieck und Rechteck) und der durch sie aufgespannte Teilgraph durch fett dargestellte Kanten hervorgehoben. Die Buchstaben innerhalb der Symbole geben die zugehörige Anfragegruppe an.

Diese Ergebnisse weisen die folgenden Eigenschaften auf:

- Es sind zusammenhängende Teilgraphen des Elementgraphs (vgl. Kapitel 5.3).
- Die Teilgraphen erstrecken sich über Dokumentgrenzen hinweg.
- Treffer für Suchterme befinden sich nicht nur innerhalb der Knoten eines Ergebnisses, sondern auch in deren „näheren“ Umfeld.

Formal ist das Ergebnis einer Anfrage wie folgt definiert:

Definition 7.0.2 (Ergebnis einer SphereSearch-Anfrage)
Das Ergebnis einer SphereSearch-Anfrage $S = (Q, J)$ mit g Anfragegruppen ist eine Liste von g-Tupeln $(e_1, \ldots, e_g)$, bestehend aus Knoten der Menge $V(\mathcal{X})$, wobei jeder Knoten e_i ein Ergebnis für Anfragegruppe G_i ist. Die Tupel sind nach erwarteter Qualität, basierend auf ihrem Relevanzwert, sortiert. Der Relevanzwert setzt sich aus lokalen Komponenten, den Knoten- und Sphären-Scores, und einer globalen Komponente, der Kompaktheit des Ergebnisses, zusammen.

In den folgenden Abschnitten wird die Berechnung des Relevanzwertes eines Ergebnisses beginnend mit den lokalen Komponenten (lokaler Knoten-Score, Sphären-Score) definiert. Anschließend wird auf die globale Scoring-Komponente, die Kompaktheit, eingegangen und schließlich die Berechnung des Gesamt-Scores erläutert.

7.1 Lokales Scoring

Bisherige Ansätze berücksichtigen meist nur den Inhalt einzelner Elemente oder des Dokuments als Ganzes (z.B. in einer Vektorrepräsentation), um seine Relevanz bezüglich der Anfrage zu beurteilen. Hierzu wird in der Regel ein Scoring-Modell benutzt, das Elementen einen hohen Score zuweist, die viele Terme der Anfrage beinhalten.

Bei SphereSearch ist der Score einzelner Knoten (Knoten-Score) nur ein Teil-Score, der als Vorstufe für spätere Berechnungen, die Sphären-Scores, dient.

7.1.1 Knoten-Scores für Schlüsselwortbedingungen

Für exakte Schlüsselwortbedingungen t wird der Knoten-Score $ns(n,t)$ eines Knotens n durch das weit verbreitete Okapi BM25 Scoring-Modell [RW94]) berechnet. Diese Scoring-Metrik hat sich sowohl in Benchmarks für unstrukturierte Daten (TREC [TREC]), als für semi-strukturierte Daten, insbesondere XML, bewährt (INEX [INEX04]).

Wir verwenden Okapi BM25 in der für XML angepassten Form (vgl. [ACR04, VPG04]).

Der Knoten-Score $ns(n,t)$ eines Knotens n für eine Schlüsselwortbedingungen t wird berechnet als:

$$ns(n,t) = w_t \frac{(k_1+1)tf}{K+tf}$$

Hierbei berücksichtigen die Bestandteile w_t das Gewicht des Anfrageterms, t und K die Dokument- bzw. Elementlänge. Sie sind definiert wie folgt:

$$w_t = log(\frac{N - r_t + 0.5}{r_t + 0.5})$$

$$K = k_1((1-b) + b\frac{dl}{avdl})$$

Die Parameter sind hierbei definiert wie folgt:

- k_1 und b sind Parameter, die abhängig von der zugrundeliegenden Datenbasis angepasst werden. Typische Werte für k_1 und b sind 1.2 und 0.75 (TREC oder OKAPI).
- $tf(term\ frequency)$ ist die Häufigkeit des Vorkommens des Terms im Knoteninhaltes.
- N ist die Anzahl der Dokumente im Korpus.
- r_t ist Anzahl der Dokumente, die den Term enthalten.
- dl ist die Länge des Inhalts des betrachteten Knotens, adl die durchschnittliche Länge von Knoteninhalten.

Im Unterschied zum originären BM25-Modell beziehen sich dl und adl nicht auf die Gesamtdokumente, sondern nur auf Elemente. In der aktuellen Implementierung ist adl konstant auf die durchschnittliche Länge aller Knoten gesetzt. Der Grund hierfür ist, dass viele Tag-Namen während des Umwandlungsprozesses der Dokumente (HTML2XML) automatisch erzeugt werden und eine durchschnittliche Tag-abhängige Länge somit nicht mehr hilfreich wäre.

Für eine Ännlichkeitsbedingung t der Form $\sim$k wird zunächst die Menge $\exp(k)$ aller Terme mit einer Ähnlichkeit, die größer als ein Schwellwert ϵ zu k ist, bestimmt.
Für $\exp(k)$ gilt:

$$x \in exp(k) \Leftrightarrow sim_{dt}(x,k) \geq \epsilon$$

Der Knoten-Score $ns(n,t)$ eines Knoten ist dann definiert als:

$$\sum_{x \in \exp(k)} sim_{dt}(k,x) * ns(n,x)$$

$sim_{dt}(x,k)$ bezeichnet hierbei die datentypabhängige Ähnlichkeit von x und k.

7.1.2 Knoten-Scores für Konzept-Wert-Bedingungen

Der Knoten-Score eines Knoten n für eine Konzept-Wert Bedingung der Form $C = V$ ist definiert als das Produkt seiner Teil-Scores für die Konzeptbedingung C, nsc, und die Wertbedingung V, nsv:

$$ns(n, C = V) = nsc(n, C = V) \cdot nsv(n, C = V)$$

Die Konzeptbedingung C kann die Form $(C =)c$ oder $(C =) \sim c$ haben. Für eine Konzeptbedingung der Form c entspricht ihr Teil-Score, falls der Knoten den geforderten Typ hat, dem Konfidenzwert der Typzuweisung.

$$nsc(n, c = V) = \begin{cases} conf(n,c) & \text{Falls } c \in types(n) \\ 0 & sonst \end{cases}$$

Für eine Konzeptbedingung mit dem Ähnlichkeitsoperator der Form $\sim c$ wird der Teil-Score wie folgt berechnet:

$$nsc(n, \sim c = V) = \max_{x \in \text{types}(n)} (sim_{String}(c,x) \cdot conf(n,x))$$

In diesem Fall wird die Ähnlichkeit zwischen c und jedem der Datentypen des Knotens n bestimmt und schließlich das maximale Produkt aus Ähnlichkeit und Konfidenzwert der Typszuweisung verwendet. Für die Ähnlichkeitsbestimmung wird der Ähnlichkeitsoperator des Typs *String* gewählt[1].

Der Teil-Score für die Wertbedingung mit optionalen $\sim$-Operator ist wie folgt definiert. Für eine Wertbedingung V der Form v entspricht ihr Teil-Score dem Knotenscore für die Schlüsselwortbedingung v:

$$nsv(n, C = v) = ns(n, v)$$

[1]Der String-Ähnlichkeitsoperator (vgl. Kapitel 5.2.2) benutzt zunächst, falls möglich, die WordNet-basierte Ähnlichkeit, ansonsten syntaktische Ähnlichkeit, d.h. Editierdistanz.

Für eine Wertbedingung V der Form $\sim v$ wird ihr Teil-Score zusätzlich mit dem Ähnlichkeitswert des Knoteninhalts zu v multipliziert:

$$nsv(n, C =\sim v) = ns(n, v) \cdot \text{sim}_c(\text{value}(n_i), v)$$

Hierbei ist sim_c die datentypspezifische Implementierung des Ähnlichkeitsoperators *sim* für den Typ c der Konzeptbedingung.

7.1.3 Sphären-Scores

Die Berechnung der lokalen Knoten-Scores berücksichtigt weder die Verlinkung von Dokumenten, noch die Verteilung von Treffern über mehrere Elemente (z.B. in Artikeln mit Kapiteln und Abschnitten).

Deswegen wird auf Basis der lokalen Knoten-Scores der Sphären-Score berechnet, der nicht nur Knoten isoliert betrachtet, sondern auch das „Umfeld“ eines Knotens.

Im Hinblick auf ihre Relevanz für das Ergebnis und damit ihren Score „gute“ Knoten zeichnen sich nicht nur durch das lokale Vorkommen der gesuchten Begriffe innerhalb eines Elements aus, sondern auch durch ein Häufung der relevanten Begriffen in ihren benachbarten Knoten.

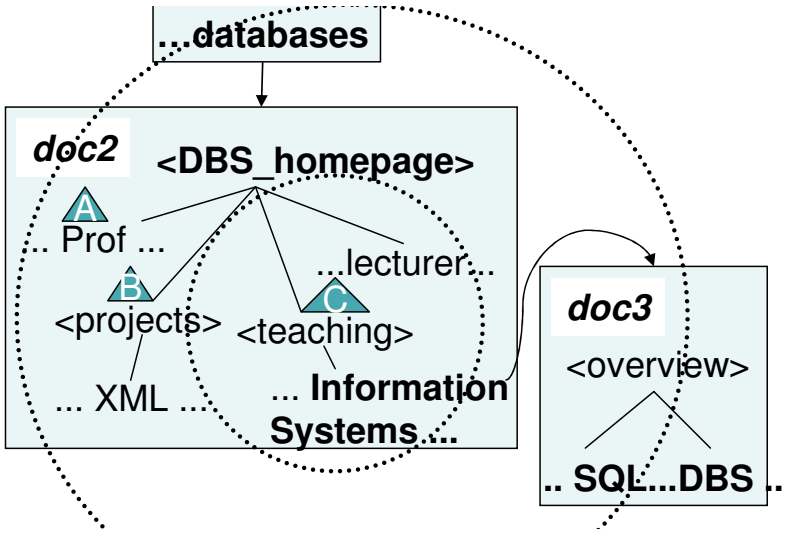


Abbildung 7.2: Sphären

Benachbarte Knoten müssen hierbei nicht zwingend im selben Dokument enthalten sein, sondern können durch (href-, X-)Links verbunden unterschiedlichen Dokumenten angehören. SphereSearch verwendet hierbei das Konzept von Sphären, um diese Art des kontextbewussten Scorings umzusetzten. Eine Sphäre um einen Knoten bezeichnet hierbei die Menge aller Knoten mit einer gegebenen Maximaldistanz zu diesem (zum Zentrum der Sphäre). Die Distanz bezeichnet hierbei die

Länge des kürzesten Pfades zwischen beiden Objekten. Die Länge einer Kante entspricht ihrem Kantengewicht (siehe Kapitel 5.3.2.1).

Abbildung 7.2 zeigt einen Ausschnitt aus Abbildung 7.1, in dem diese Sphären um das Element `teaching`, dem Teilergebnisknoten für die Anfragegruppe C, angedeutet sind. Die Begriffe, die zum Score dieses Knoten beitragen sind fett dargestellt.

Der Sphären-Score eines Knotens ist definiert als der aggregierte Wert aus allen Knoten innerhalb seiner Sphäre, also aus den diesen Knoten umgebenden Knoten bis zu einer Maximalentfernung. Hierbei werden Knoten, die weiter vom Zentrum entfernt sind, schwächer gewichtet.

Die Sphäre eines Knoten n mit Radius D umfasst alle Knoten mit Maximaldistanz D zu n.

Definition 7.1.1 (Sphäre)

Die Sphäre $S_D(n)$ *eines Knotens n des Elementgraphs $G(\mathcal{X})$ mit Radius D bezeichnet die Menge aller Knoten $v \in V(\mathcal{X})$, für die gilt:*

$\delta_{\mathcal{X}}(v,n) \leq D$ *, wobei die Funktion $\delta_{\mathcal{X}}$ die Kosten des billigsten Pfades von v und n in $G(\mathcal{X})$ berechnet. Die Kosten sind die Summe der Kantengewichte.*

Definition 7.1.2 (Sphären-Score einer Bedingung)
Der Sphären-Score $s_D(n,t)$ *mit Sphärengröße D ist für eine (exakte oder Ähnlichkeits-) Bedingung b definiert als:*

$$s_D(n,b) = \sum_{v \in S_D(n)} ns(v,b) * \alpha^{\delta_{\mathcal{X}}(v,n)}$$

$$0 < \alpha \leq 1$$

Der Faktor α ist ein konfigurierbarer Dämpfungskoeffizient zwischen 0 und 1, der den Einfluss von Knoten, die weiter vom Zentrum entfernt sind, reduziert.

Abbildung 7.3 zeigt einen Beispieldatenkorpus bestehend aus drei verlinkten Dokumenten (erkennbar an verschieden Grautönen) und ihren Elementen. Die Anzahl an 't's an den einzelnen Knoten zeigt die Häufigkeit der Vorkommen des Terms 't' in ihrem Inhalt. Für eine einfache Schlüsselwortanfrage nach 't' würde Knoten 2 den höchsten Score, basierend auf seinem Knoten-Score, erhalten; in SphereSearch hat dieser auch den höchsten Knoten-Score [2]. Gesamt gesehen könnte allerdings

[2]Zur Vereinfachung wird im Beispiel angenommen, dass der Knoten-Score direkt von der Anzahl

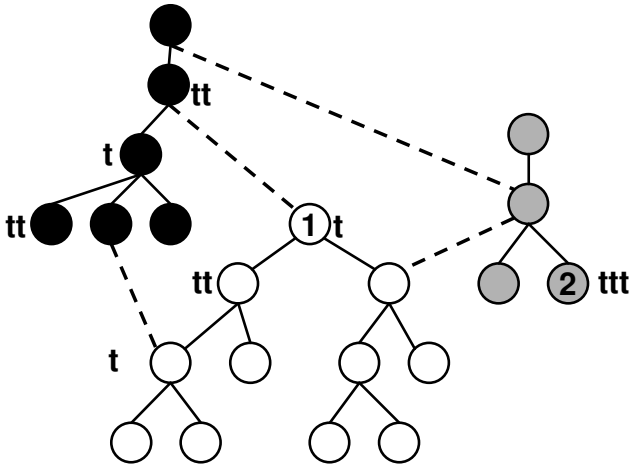


Abbildung 7.3: Verlinkte Dokumente

Knoten 1 ein besserer Treffer sein, da der Term 't' wesentlich häufiger in der direkten Nachbarschaft von Knoten 1, verglichen mit Knoten 2 vorkommt. Aus diesem Grund ist auch der Sphären-Score für Knoten 1 höher als für Knoten 2.

Abbildung 7.4 zeigt dieselben drei Dokumente, die in Abbildung 7.3 dargestellt sind als Sphären mit Größen 1,2,3 und 4 um Knoten 1 herum. Hierbei ist das Gewicht für Dokumentkanten 1 und für Links 2.

Für gewählte Parameter $\alpha = 0.6$ und $D = 4$ erhält man die folgenden Sphären-Scores für die Knoten 1 und 2:

$$s_4(1,t) = 1 + 2 \cdot (0.6) + 3 \cdot (0.6)^2 + 1 \cdot (0.6)^3 + 5 \cdot (0.6)^4 = 4,14$$

$$s_4(2,t) = 3 + 0 \cdot (0.6) + 0 \cdot (0.6)^2 + 0 \cdot (0.6)^3 + 3 \cdot (0.6)^4 = 3,38$$

Somit ist Knoten 1 in unserem Modell ein besserer Treffer für Bedingung t als Knoten 2. Die Sphären-Scores für die beiden Knoten werden hierbei maßgeblich durch Knoten der über Links verbundenen Dokumente beeinflusst und nicht nur durch Knoten des eigenen Dokuments.

Der Sphären-Score eines Knotens für eine ganze Anfragegruppe, bestehend aus Schlüsselwort- oder Konzept-Wert-Bedingungen mit oder ohne $\sim$-Operator, wird als Summe der Sphären-Scores für die einzelnen Bedingungen bestimmt.

---

des Vorkommens des Suchterms abhängt und die übrigen Werte der BM25-Scoring-Funktion konstant sind

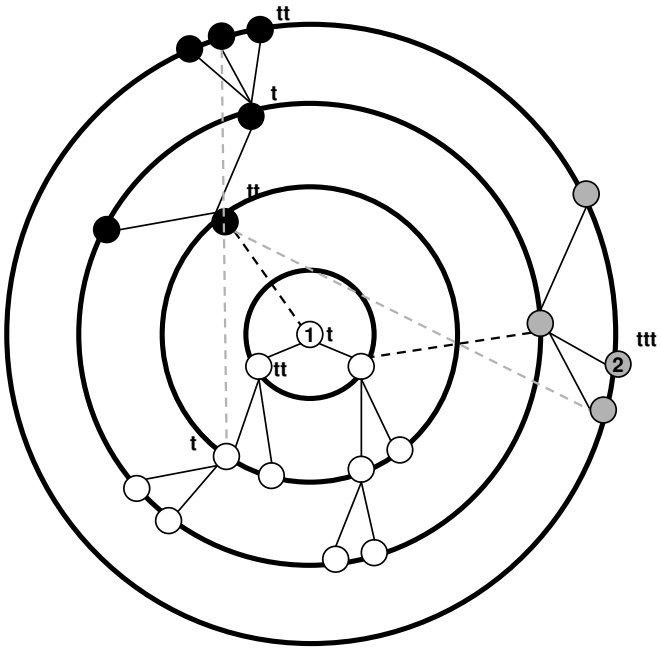


Abbildung 7.4: Sphären um einen Knoten

Definition 7.1.3 (Sphären-Score einer Gruppe)

Der Sphären-Score eines Knotens n für eine Anfragegruppe $G = (b_1, ..., b_k)$ ist definiert als:

$$s_D(n, G) = \sum_{j=1}^{k} s_D(n, b_j)$$

7.2 Globales Scoring

Ein Ergebnis für eine Anfrage mit g Anfragegruppen ist ein aus g Knoten bestehendes Tupel, d.h. einem Knoten pro Anfragegruppe. Da bei einer Anfrage mit mehreren Teilbedingungen diese immer in Beziehung zueinander stehen, müssen auch die Elemente, die diese Bedingungen erfüllen, zueinander in der gewünschten Beziehung stehen. Aus diesem Grund muss ein Ergebnis für eine Anfrage ein zusammenhängender Teilgraph des Gesamtdokumentgraphs sein. Abbildung 7.5 zeigt den Elementgraph eines DokumentenkorpusApproximationalgorithmus mit zwei Tref-

fern für die zu Beginn des Kapitels vorgestellte Anfrage
$A(professor)\ \ B(research, \sim XML)\ \ C(course, \sim databases)$
mit drei Anfragegruppen A, B und C. Ein Treffer ist hierbei mit Sternen, der zweite mit Rechtecken markiert. Der mit Rechtecken markierte Treffer ist ein korrektes Ergebnis, wohingegen der mit Sternen markierte Treffer allerdings kein für den Benutzer im Sinne der Anfrage relevantes Ergebnis ist, da seine Bestandteile zwar mit Links verbunden sind, aber inhaltlich nicht zusammengehören.

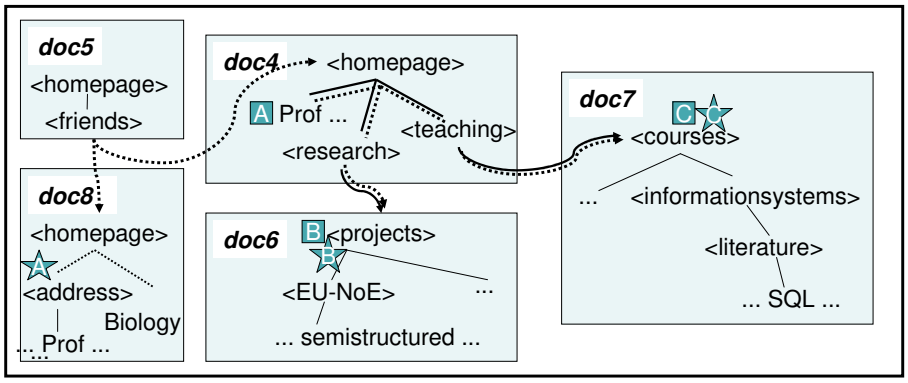


Abbildung 7.5: Kompaktheit

Dies zeigt, dass die reine Aggregation der Scores für die Teilergebnisse unter der Voraussetzung, dass diese im Dokumentgraph miteinander verbunden sind, kein angemessenes Maß zur Berechnung der Relevanz ist. Dann nämlich wären beide Treffer in dem obigen Beispiel gleich gut, denn sie hätten den selben Ergebnis-Score.

Man sieht in der Abbildung, dass für den mit Sternen markierten Treffer der Abstand zwischen den Teilergebnissen für die Gruppen A und B signifikant größer und somit auch der durch dieses Ergebnis aufgespannte Teilbaum deutlich größer ist als bei dem mit Rechtecken markierten Treffer. Verallgemeinert kann man sagen, dass mit steigendem Abstand der Teilergebnisse eines Ergebnisses die Wahrscheinlichkeit sinkt, dass diese sich auf denselben inhaltlichen Kontext beziehen.

Intuitiv sollten deswegen potentielle Ergebnisse bevorzugt werden, deren Knoten im Elementgraph näher beieinander liegen, d.h. deren Repräsentation *kompakter* ist. Meist bedeutet dies, dass sie entweder im selben Dokument zu finden sind, oder in benachbarten Dokumenten und somit nur wenige Kanten entfernt.

Aus diesem Grund ist in SphereSearch der Score einer potentiellen Antwort eine Kombination der Sphären-Scores der einzelnen Knoten und der Kompaktheit des Teilgraphs, der das potentielle Ergebnis repräsentiert.

7.2.1 Kompaktheit

Die Kompaktheit eines Ergebnisses soll ein Maß für die Größe des Teilgraphs sein, der die Knoten eines potentiellen Ergebnisses beinhaltet. Die Bestimmung des kostengünstigsten Teilgraphs der die Knoten einer Knotenteilmenge (hier: die Ergebnisknoten) eines Graphs (hier: der Elementgraph) mit einer Kantenbewertungsfunktion (hier: die Kantengewichte) verbindet, ist bekannt als *Steinerbaumproblem* [Hak71].

Abbildung 7.6 zeigt erneut die drei Dokumente (auf der linken Seite) aus Abbildung 7.3. Es sei eine Anfrage mit drei Anfragegruppen gegeben, wobei für Gruppe G_1 die Knoten A und B, für Gruppe G_2 der Knoten X und für Anfragegruppe G_3 die Knoten 1 und 2 die Ergebnisknoten seien.

Es sind in diesem Beispiel vier zusammenhängende Teilgraphen als Ergebnis möglich, die jeweils eine der Knotenmengen $\{A, X, 1\}$,$\{B, X, 1\}$,$\{A, X, 2\}$,$\{B, X, 2\}$ umfassen. Zur exakten Bestimmung ihrer Kompaktheit müsste für jede Knotenmenge ihr kostengünstigster zusammenhängender Teilgraph innerhalb des Elementgraphs, also ihr kostengünstigster Steinerbaum, bestimmt werden.

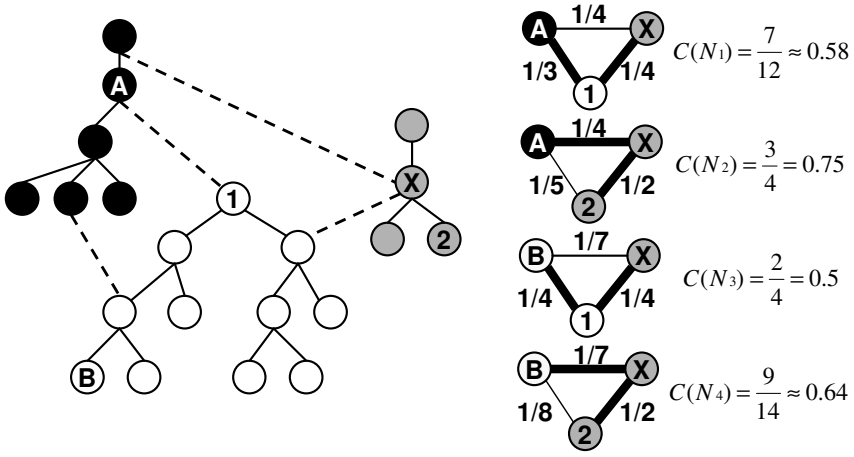


Abbildung 7.6: Kompaktheit auf Basis von minimalen Spannbäume

Das Steinerbaumproblem ist ein NP-vollständiges Problem, aber es sind Approximationsalgorithmen mit Approximationsfaktor < 2 bekannt [Ple81]. Die Approximationsalgorithmen benutzen meist die Abschätzung, dass ein minimaler Spannbaum (MST) auf dem Distanzgraph der betrachteten Knotenmenge höchstens um Faktor 2 teurer ist als die Kosten des optimalen Steinerbaums.

Da wir nicht einen tatsächlichen Steinerbaum benötigen, sondern nur effizient die Kompaktheit eines Ergebnisses beurteilen wollen, ist diese Abschätzung ausreichend[3]. Zunächst muss der Distanzgraph eines Ergebnisses definiert werden.

[3]Die Approximationalgorithmen bestimmen nach dieser Abschätzung noch den approximierten

Definition 7.2.1 (Distanzgraph)

Der Distanzgraph für ein Ergebnistupel N=(e1,...,en) ist wie folgt definiert. Sei $G(\mathcal{X}) = (V(\mathcal{X}), E(\mathcal{X}))$ *der Elementgraph der Dokumentsammlung* $\mathcal{X}$ *und* $D = \{e1, ..., en\} \subseteq V(\mathcal{X})$*, dann heisst der kantengewichtete vollständige Graph* $DIST = (D, E_D, \delta_{\mathcal{X}})$ *mit*

- $E_D = \{\{u, v\} | u, v \in V(\mathcal{X})$ *und* $u \neq v\}$
- $\delta_{\mathcal{X}}(u, v) =$ *Kosten des billigsten Weges zwischen u und v in* $G(\mathcal{X})$

der Distanzgraph von N in $G(\mathcal{X})$*.*

Die Funktion $\delta_{\mathcal{X}}(u, v)$ berechnet hierbei die Kosten des billigsten Pfades von u nach v. Die Kosten eines Pfades sind die Summe der Kantengewichte.

Diese Approximation ist in $0(n^3)$[Ple81] Schritten, wobei n die Anzahl der Knoten ist, berechenbar. In der Praxis liefert der Approximationsalgorithmus oft tatsächlich die Größe des optimalen Steinerbaums. Außerdem erlaubt die Nutzung dieses Approximationsverfahrens die Umsetzung eines effizienten Top-k-Algorithmus.

Da der Score eines Ergebnisses größer sein soll, wenn die Summe der Kantengewichte klein ist, und maximal sein soll, wenn alle Bedingungen im selben Knoten erfüllt werden (Distanz 0), wird die Kompaktheit eines Ergebnisses wie folgt definiert.

Definition 7.2.2 (Kompaktheit)
Die Kompaktheit $C(N)$ *eines Ergebnistupels* $N = (e_1, ...e_n)$ *ist auf Basis der Summe der Kantengewichte* $S(N)$ *eines minimalen Spannbaums auf seinem Distanzgraph definiert als:*

$$C(N) = \frac{1}{S(N)+1}$$

Die Verbindungsgraphen der vier möglichen Ergebnisse sind in Abbildung 7.6 rechts dargestellt. Dokumentkanten haben hierbei Gewicht 1, Links Gewicht 2. Kanten, die zum MST des Distanzgraphen gehören, sind fett dargestellt. Es ist ersichtlich, dass das potentielle Ergebnis N_2, das aus den Knoten A,X und 2 besteht, das kompakteste ist. Es ist zu beachten, dass die Antwortknoten in verschiedenen

Steinerbaum, da der Spannbaum auf dem Distanzgraph oft keinen Baum im Ursprungsgraph repräsentiert.

Dokumenten zu finden sind und somit von einer herkömmlichen Suchmaschine ohne Kontextbewusstsein nicht hätten gefunden werden können.

7.2.2 Join-Bedingungen

Zusätzlich zu Schlüsselwort- und Konzept-Wert-Bedingungen können optional Join-Bedingungen spezifiziert werden. Zwei Knoten sind dann ein Treffer für eine Join-Bedingung der Form `A.v=B.w` bzw. `A.v`$\sim$`B.w`, wenn sie den Typ `v` bzw. `w` haben und ihre Inhalte identisch bzw. ähnlich sind.

Obwohl Join-Bedingungen offensichtlich auf Annotationen wie z.B. Ortsnamen oder Datumsangaben am effizientesten ausgewertet werden können, können Join-Bedingungen prinzipiell für alle Datentypen spezifiziert werden. Da Tag-Namen in unserem Modell analog zu Annotationen als Datentypen behandelt werden, kann somit jeder Tag-Name als ein Join-Attribut spezifiziert werden.

In diesem Fall werden ganze Textblöcke aufgrund entsprechender String-Ähnlichkeitsmaße ([CRF03]) miteinander verglichen. Dies ist inbesondere dann hilfreich, wenn die gesuchte Information nur im Volltext eines Elements enthalten ist, wie z.B.: „Der Regisseur dieses Films ist ...“.

Join-Bedingungen werden fundamental unterschiedlich zu allen anderen Bedingungen ausgewertet, da sie nicht den Score für einzelne Knoten ändern. Als Ergebnis einer Join-Bedingung werden neue Kanten im Elementgraph erzeugt: virtuelle Links. Die Endpunkte dieser virtuellen Links sind die Paare von Knoten, die die Join-Bedingung erfüllen. Das Gewicht dieser Kanten reflektiert die Ähnlichkeit ihrer Endpunkte.

Eine Join-Bedingung *findet* nicht nur neue Ergebnisse, die ohne die Auswertung dieser Bedingung nicht verbunden wären, sondern beeinflusst auch das Ranking von Treffern, die auch ohne Join einen verbundenen Graph repräsentierten. Dies geschieht nämlich dann, wenn eine durch die Join-Bedingung erzeugte Kante einen kürzeren Pfad in einem potentiellen Ergebnis erzeugt und somit der MST dieses Treffers kompakter wird und der zugehörige Score steigt.

Als Beispiel betrachten wir erneut Abbildung 7.6 und nehmen an, dass die Anfrage eine zusätzliche exakte Join-Bedingung zwischen G_1 und G_2 beinhalte. Wir nehmen weiterhin an, dass die Inhalte von Bs Vaterknoten und X Treffer für je eine Seite der Join-Bedingung sind und dass dies (um das Beispiel einfach zu halten) die einzigen Treffer im Graph sind.

Dies führt in diesem Beispiel dazu, dass ein virtueller Link mit Gewicht 2 zwischen den beiden Knoten in den Elementgraph eingefügt wird. Da nun die Distanz von B zu X auf 3 reduziert wurde, erhöht dies die Kompaktheit der potentiellen Antwort $\{B, X, 2\}$ auf 0.2, was ebenfalls Einfluss auf den Gesamt-Score hat und diesen Treffer höher in der Ergebnisliste einstuft.

Formal erweitern wir die Datensammlung $\mathcal{X}$ um neue *virtuelle Links* zur erweiterten Datensammlung $\mathcal{X}' = (\mathcal{D}, \mathcal{L}')$. Für eine Ähnlichkeits-Join-Bedingung

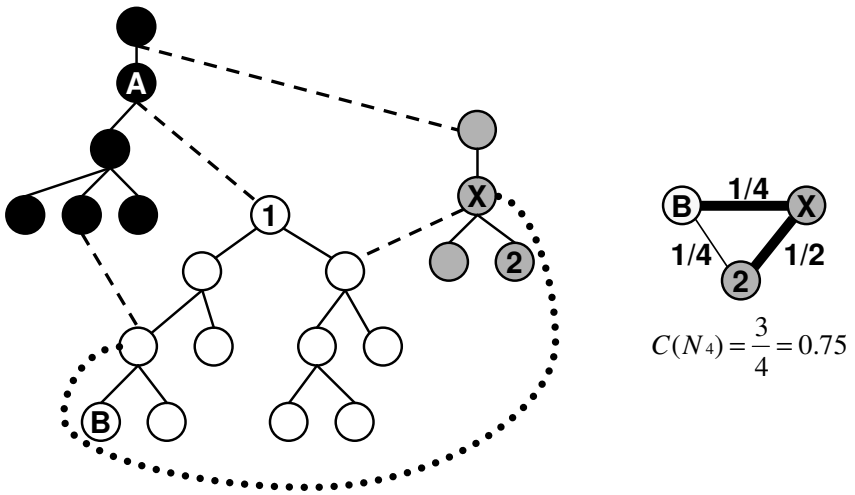


Abbildung 7.7: Virtuelle Links

$Q_i.v \sim Q_j.w$ betrachten wir die Menge $N(v)$ (bzw. $N(w)$) aller Knoten mit Typ v (bzw. w) [4].

Für jedes Paar $x \in N(v), y \in N(w)$ fügen wir eine neue Kante {x,y} mit Gewicht $\frac{1}{sim_{dt}(x,y)}$, das die Ähnlichkeit der Inhalte quantifiziert, hinzu, falls dieses über einem spezifischen Schwellwert ϵ liegt. Ein Wert von 1 repräsentiert hierbei Gleichheit. Für exakte Join-Bedingungen werden aus diesem Grunde nur Kanten mit Gewicht 1 hinzugefügt (falls der Inhalt der Endpunkte identisch ist).

sim_{dt} berechnet hierbei die datentypspezifische Ähnlichkeit für den Datentyp dt. Falls die Datentypen unterschiedlich sind wird der Ähnlichkeitsopertor des nächsten gemeinsamen Supertyp gewählt (vgl. Kapitel 5.2.1).

Es ist Absicht, dass in unserem Modell potentielle Antworten, die kompakt, aber nicht durch kürzeste Pfade unter Ausnutzung virtueller Links miteinander verbunden sind, nicht benachteiligt werden, auch wenn eine Join-Bedingung in der Anfrage spezifiziert ist. Dies basiert auf der Annahme, dass diese Antworten durch ihre Kompaktheit schon in enger Beziehung zueinander stehen und somit ebenfalls gute Antworten für die Anfrage darstellen.

[4] Da dies mit dieser Auswertung verbundenen Berechnungen sehr aufwändig sind, betrachten wir nur Kandidaten, die in der Nachbarschaft von Treffern liegen, die sich für die übrigen Bedingungen einer Gruppe qualifizieren

7.3 Kombiniertes Scoring

Der Score einer potentiellen Antwort N für eine Anfrage S setzt sich zusammen aus den aggregierten Sphären-Scores und der Kompaktheit des Ergebnisses.

Definition 7.3.1 (Relevanzwert eines Ergebnis-Tupels)

Der Relevanzwert $s(N, S)$ eines Ergebnis-Tupels $N = (n_1, ..., n_g)$ für eine Anfrage S mit g Anfragegruppen ist definiert als:

$$s(N, S) = \beta C(N) + (1 - \beta) \sum_{i=1}^{g} s_D(n_i, G_i)$$

mit

- *$C(N)$: Kompaktheit des Ergebnisses N*
- *$s_D(n_i, G_i)$: Sphären-Score von Knoten n_i für Gruppe G_i*
- *$0 \leq \beta \leq 1$: Gewichtungsfaktor*

Der Parameter β (mit einem Wert zwischen 0 und 1) regelt die Gewichtung von Kompaktheit und Summe der Sphären-Scores im Ergebnis. Je größer β, desto größer wird der Einfluss der Kompaktheit auf das Ergebnis, d.h. desto mehr werden kompakte Ergebnisse mit geringeren Sphären-Scores bevorzugt.

8 Algorithmen der Anfrageausführung

Eine naive Auswertungsstrategie würde sowohl bei der Auswertung und der Score-Berechnung von Teilanfragen, also den Sphären-Scores für Teilbedingungen und der Auswertung der Join-Bedingungen, als auch der Bestimmung und Score-Berechnung des Gesamtergebnisses, d.h. den das Ergebnis repräsentierenden Spannbäumen, deutlich zuviel Berechnungsaufwand verursachen.

Im Falle der Sphären-Scores kann durch einfache Überlegungen die Menge der zu betrachtenden Elemente deutlich reduziert werden. Für die MST-Berechnung kann durch Adaption von bekannten Top-k-Algorithmen der Berechnungsaufwand deutlich reduziert werden.

In den folgenden Abschnitten wird auf die Berechnung der Anfrageergebnisse im Detail eingegangen.

8.1 Anfrageausführung

Die Auswertung einer Anfrage läuft (vereinfacht) in den folgenden Schritten ab:

1. Die Trefferknoten für alle Teilanfragen (aller Anfragegruppen) werden ermittelt.
2. Sphären-Scores für alle Anfragegruppen werden berechnet
3. Join-Bedingungen werden ausgewertet und evtl. virtuelle Links generiert.
4. Die Kompaktheit wird für eine Teilmenge der potentiellen Ergebnisse berechnet, um die besten Ergebnisse zu ermitteln.

8.2 Berechnung der Sphären-Scores

Für jede Anfragegruppe werden zuerst die elementaren Bedingungen (Konzept-Wert- und Schlüsselwortbedingungen) ausgewertet und die entsprechenden Knoten-Scores errechnet. Potentiell würde ein (maximal) naiver Algorithmus nun die Sphären-Scores für alle Knoten bestimmen.

Allerdings können Knoten, die Teil eines Ergebnisses für eine Anfragegruppe G_i sind, sich maximal in Distanz D, der Sphärengröße, zu mindestens einem Knoten befinden, dessen Knoten-Score für mindestens eine elementare Bedingung der Anfragegruppe G_i größer null ist.

Wäre nämlich ein Knoten n_i eines Teilergebnisses einer Gruppe mehr als D Knoten von jedem Knoten mit lokalem Score ≥ 0 für eine elementare Bedingung dieser Gruppe entfernt, hätte er einen Sphären-Score von 0. Dies steht aber im Widerspruch zur Definition eines potentiellen Ergebnisses.

Somit müssen die Sphären-Scores nur maximal bis zur Distanz D von Knoten mit positiven Knoten-Scores für einer Anfragegruppe berechnet werden.

Für alle Kandidatenknoten werden nun die Sphären bzw. die Knoten, die auf diesen liegen, bis zur Distanz D bestimmt und ihre Knoten-Scores, multipliziert mit dem entsprechenden Dämpfungsfaktor α, zum Sphären-Score des Knotens addiert.

Die Ergebnisliste R_i einer Anfragegruppe G_i umfasst dann alle Knoten mit einem Sphären-Score ungleich null für Gruppe G_i

8.3 Berechnung der Join-Bedingungen

Um die Effizienz der Berechnung von virtuellen Links (durch Auswertung einer Join-Bedingung) zu verbessern, wird nur eine begrenzte Menge an Endpunkten berücksichtigt. Es werden nur Knoten berücksichtigt, die sich maximal in Distanz D zu einem Knoten mit Sphären-Score > 0 befinden, da ein weiter entfernter Knoten maximal $\frac{1}{2(D+1)+1}$ zur Kompaktheit beitragen kann, was vernachlässigbar klein für typische Sphärengrößen D ist. Experimente haben gezeigt, dass diese Heuristik die Ergebnisgüte kaum beeinflusst, die Berechnung aber deutlich beschleunigt. Insbesondere konnte nie eine Beeinflussung der Top-10-Treffer beobachtet werden.

Für eine Join-Bedingung $i.x = j.y$ zwischen den Anfragegruppen G_i und G_j seien R_i und R_j Listen der Kandidaten für x in G_i bzw y in G_j, die für die die Auswertung der Join-Bedingung in Frage kommen. Falls apriori nicht weiter eingeschränkt werden kann, welche Elemente aus G_i mit Elementen aus G_J verglichen werden müssen, muss das kartesische Produkt der beiden Listen bzw. Mengen auf Ähnlichkeit hin untersucht, d.h. entsprechend der Join-Bedingung miteinander verglichen werden.

Falls die Domäne der verglichenen Objekte eine (partielle) Ordnung aufweist, die es ermöglicht die Menge der zu vergleichenden Objekte aufgrund ihrer Ordnung einzuschränken, wird der Vergleich als *Sort Merge Band Join* [DNS91][LT95] durchgeführt. Ein Bereichs-Join (Band-Join) bezeichnet einen Nicht-Equi-Join, falls das Join-Prädikat es erfordert, dass Werte des Join-Attributs der einen Relation in einen bestimmten Wertebereich der anderen Relation fallen.

Dies trifft z.B. für die SSL-Join-Bedingung $A.date \sim B.date$ zu. Da Datumsangaben intern normalisiert als Zahlenwert gespeichert werden, können diese einfach sortiert werden. Da wir bei Ähnlichkeitsbedingungen grundsätzlich nur an Treffern

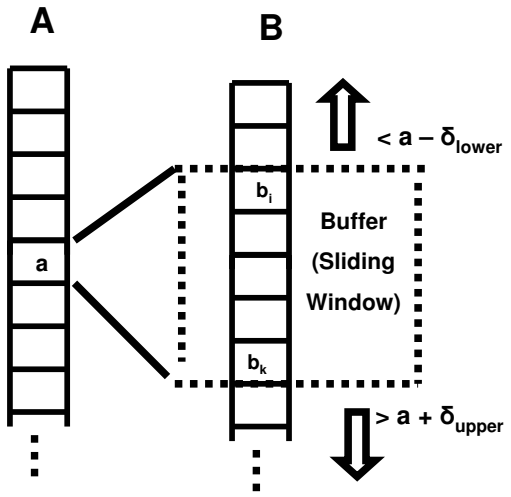


Abbildung 8.1: Sort-Merge-Band-Join

bis zur einem gewissen Ähnlichkeitsschwellwert ϵ interessiert sind, interessieren nur Vergleiche von Elementpaaren, für die gilt: $sim_{date}(A.date, B.date) > \epsilon$.

Da die Ähnlichkeit der Join-Attribute, d.h. in diesem Beispiel der Datumswerte, mit wachsenden Abstand zueinander sinkt bis der Ähnlichkeitsschwellwert unterschritten wird, gibt es zwei Konstanten c_{lower} und c_{upper}, so dass gilt:
$a - c_{lower} \leq b \leq a + c_{upper} \Leftrightarrow sim_{date}(a, b) > \epsilon$, wobei a ein Treffer für den linken Join-Partner (A.date) und b für den rechten Join-Partner(B.date) sei. Dabei kann $c_{lower} = c_{upper}$ oder einer der Werte null sein. Dies ist ein Bereichs-Join, der im Wesentlichen eine Abwandlung eines Sort-Merge Equi-Joins darstellt. Abbildung 8.1 illustriert einen Sort-Merge-Band-Join. Zunächst müssen beide Relationen A und B aufsteigend sortiert werden. Anschließend wird jedes Element von A mit einer Teilmenge der Elemente von B verglichen, für die gilt $a - c_{lower} \leq b \leq a + c_{upper}$
Da nun jedes Element von A nur mit den Elementen von B innerhalb eines *Sliding Windows* verglichen werden muss, sinkt die Anzahl der notwendigen Vergleiche von $|A||B|$ auf $|A|s$ wobei s die durchschnittliche Anzahl an Elementen innerhalb des *Sliding Window* ist.

Allgemein muss eine Domäne d und ihre zugehörige Ähnlichkeitsfunktion $sim_d(a, b)$ folgende Eigenschaften aufweisen, damit die Join-Berechnung als *Sort Merge Band Join* durchgeführt werden kann:

1. d muss eine Ordnung aufweisen
2. $|a - x| \geq |b - x| \Rightarrow sim(b, x) \geq sim(a, x)$ für $a, x, b \in dt$ [1]

[1] Intuitiv: Falls ein Element b näher an einem Element x als ein Element a liegt, muss b zu x ähnlicher als a zu x sein

Algorithm 3 Sort-Merge-Band-Join

```
1: buffer_b := new Buffer()
2: while A.next() ≠ NULL do
3:    a := A.getNext()
4:    while B.next() < a + δ_upper do
5:       buffer.append(B.getNext())
6:    end while
7:    current_b := buffer.getFirst()
8:    repeat
9:       if current_b < a − δ_lower then
10:         buffer.dropFirst()
11:      else
12:         compare(a, current_n)
13:      end if
14:      current_b := buffer.getNext()
15:   until current_b is NULL
16: end while
```

3. Es muss zwei Konstanten c_{lower} und c_{upper} geben, so dass für zwei beliebige Elemete a und b der Domäne gilt:
   $a - c_{lower} \leq b \leq a + c_{upper} \Leftrightarrow sim_{date}(a, b) > \epsilon$

In Abbildung 3 ist ein Algorithmus zur Berechnung im Pseudo-Code angegeben. In dem angegebenen Algorithmus wird für die Elemente innerhalb des *Sliding Window* ein Puffer benutzt, der das wiederholte Lesen von Elementen innerhalb des Fensters minimiert.

8.4 Ein Top-k-Algorithmus zur MST-Berechnung

Ein naiver Algorithmus zur Berechnung der endgültigen sortierten Ergebnisliste würde wie folgt vorgehen. Er würde alle potentiellen Antworten aus Treffern für die einzelnen Anfragegruppen generieren, würde ihren Verbindungsgraphen erzeugen, ihre Kompaktheit bestimmen und anschließend ihre Scores errechnen und die Ergebnisse entsprechend sortieren. Selbst bei nur 3 Anfragegruppen und 1000 Treffern pro Gruppe müssten somit 1.000.000.000 potentielle Antworten untersucht und hierfür ihre Verbingungsgraphen erzeugt werden.

Abgesehen von der Tatsache, dass aufgrund dieses erheblichen Aufwandes die Laufzeit der Resultatsberechnung in vielen Fällen nicht akzeptabel wäre, sind Benutzer bei Anfragen dieser Art typischerweise nicht am vollständigen Ergebnis, sondern nur den besten k, den *Top-k*-Treffern, interessiert.

Eine Auswertung, die nur die Top-k-Ergebnisse bestimmt, würde eine Menge nutzloser Berechnungen sparen, zumal viele der untersuchten Ergebnisse nicht einem über einen verbundenen Spannbaum verfügten.

Um diesen Aufwand zu minimieren und somit den mit dieser Berechnung verbundenen Performance-Flaschenhals zu beseitigen, benutzt SphereSearch einen Top-k-Ansatz, der sich an Fagins Threshold-Algorithmus ([FLN03][NR99][GBK00]) orientiert.

Der Threshold Algorithmus (TA)erfordert als Eingabe nach Score sortierte Listen und eine monotone Aggregationsfunktion. Jede Liste repräsentiert Teilergebnisse bezüglich einer Dimension, bzw. einem Attribut der Anfrage, zusammen mit ihren Teil-Scores. Inkrementell werden Elemente von den Köpfen der Listen gelesen und Scores bzw. Schranken für Scores berechnet, so dass im Idealfall nur ein kleiner Prefix der Listen bearbeitet werden muss, um die Top-k-Elemente zu bestimmen.

Fagins Threshold-Algorithmus gibt es in zwei Varianten. Die erste Variante TA benutzt zusätzlich zu den sequentiellen Zugriffen auf die Listen auch wahlfreie Zugriffe, während NRA (no random access) ausschließlich sequentielle Zugriffe benutzt. In SphereSearch benutzten wir einen an NRA angelehnten Algorithmus.

Die benutzte Aggregationsfunktion ist die Funktion zur Berechnung des Gesamt-Scores (siehe Kapitel 7.3).
Eine Aggregationsfunktion t mit i Parametern ist nach [FLN03] mononton, wenn $t(x_1, ..., x_i) \leq t(x'_1, ..., x'_i)$, falls für alle j gilt: $x_j \leq x'_j$. Da die Score-Berechnungsfunktion eine gewichtete Summe ist, ist dies erfüllt.

Als Eingabe erhält der Algorithmus zwei Sorten von sortierten Listen:

1. *Gruppenlisten:* Für jede Anfragegruppe G_i gibt es eine Liste L_i, die die Ergebnisknoten der Gruppe sortiert nach absteigenden Sphären-Scores enthält.

2. *Kantenlisten:* Für jedes Paar von Anfragegruppen $G_i, G_j, G_i \neq G_j$ mit ihren Ergebnislisten R_i, R_j gibt es eine Liste L_{ij} mit allen ungeordneten Paaren $\{\{r_i, r_j\} | r_i \in R_i, r_j \in R_j\}$, absteigend sortiert nach ihrem zugehörigem Score $\delta_{\mathcal{X}'}(r_i, r_j)$.

Die Kantenlisten enthalten die Kanten der Verbindungsgraphen aller potentiellen Ergebnisse. Der Algorithmus konstruiert nun inkrementell Kandidaten für verbundene MSTs, indem er nacheinander von jeder Liste immer das Kopfelement auswählt und mit bereits existierenden Ergebniskandidaten kombiniert. Er arbeitet solange die Listen ab, bis die Top-k-Treffer bestimmt sind.

Falls zum Beispiel ein Eintrag $\{r_i, r_j\}$ aus L_{ij} gelesen wird und bereits ein Kandidat mit demselben r_i, allerdings ohne Kante zu einem Knoten aus R_j existiert, wird dieser Kandidat um $\{r_i, r_j\}$ erweitert.

Der Algorithmus pflegt eine nach Score sortierte Liste aus k verbundenen MSTs mit den höchsten Scores. Wenn ein Kandidat, der um eine Kante erweitert wird, einen vollständigen verbundenen MST bildet, wird er in diese Liste aufgenommen

Algorithm 4 Top-k-Algorithmus

```
1: for d = 1 to max_depth do
2:   for n = 1 to num_lists do
3:     new_el = List_1.getTopElement()
4:     E = new_el ∪ getExtensibleCandidates(new_el)
5:     for all e ∈ E do
6:       new_cand = copy(e)
7:       new_cand[n] := new_el
8:       min − k = min{score(d)|d ∈ top − k}
9:       if complete(new_cand) ∧ (|top − k| < k) then
10:        top − k = top − k ∪ new_cand
11:      else if complete(new_cand) ∧ score(new_cand) > min − k then
12:        replace d ∈ top − k with lowest score by new_cand
13:      else if bestscore(new_cand) > min − k then
14:        candidates = candidates ∪ new_cand
15:      end if
16:    end for
17:  end for
18:  if max{bestscore(d)|d ∈ candidates} < min − k then
19:    return top-k
20:  end if
21: end for
22: —
23: bestscore(cand) = Σ_{cand[i]!=null} s_i + Σ_{cand[i]==null} high_i
```

und verdrängt den MST mit dem niedrigsten Score, wenn sein Score höher ist oder die Liste noch nicht k MSTs umfasst.

In jeder Runde berechnet der Algorithmus für die nicht vollständigen Kandidaten ihren *Bestscore*. Der *Bestscore* eines Kandidaten bezeichnet den Score, den dieser im optimalen Fall erreichen kann. Dieser berechnet sich als die Summe der Teil-Scores der ausgewerteten Dimensionen des Kandidaten und den Scores der Kopfelemente der Listen der noch nicht ausgewerteten Dimensionen.

Der Algorithmus terminiert, wenn kein Kandidat mehr im weiteren Verlauf so vervollständigt werden kann, dass sein Score höher ist als das Element mit dem niedrigsten Score der Top-k-Liste, also wenn der maximale *Bestscore* aller Kandidaten kleiner ist als der niedrigste Score innerhalb der Top-k-Liste. Dieser Algorithmus berechnet die Top-k potentiellen Antworten mit den besten Scores. Abbildung 4 zeigt den Algorithmus der Top-k-Berechnung als Pseudocode, der im Anschluss im Detail erläutert wird.

Erläuterung des Algorithmus

Der Algorithmus iteriert bis zu einer maximalen Tiefe (dies ist typischerweise die Länge der kürzesten Liste) über die Liste aller Dimensionen (Zeile 1-2). Für das aktuell gewählte Element einer Liste werden alle Kandidaten bestimmt, die mit diesem erweitert werden können, so dass ein neuer Kandidat entsteht (Zeile 3-4). Bevor ein neuer Kandidat konstruiert wird, wird er dupliziert, da der nicht erweiterte Kandidat evtl. auch durch ein anderes Element erweitert werden könnte und somit erhalten bleiben muss (Zeile 6-7). Anschließend wird der Score des schlechtesten Kandidaten der Top-k-Liste bestimmt (8). Wenn ein Kandidat vollständig ist und weniger als k Elemente in der Top-k-Liste enthalten sind, wird er in diese aufgenommen (Zeile 9-10). Falls schon k Kandidaten enthalten sind, aber der Score des neuen Kandidaten besser als der schlechteste der Top-k-Liste ist, wird dieser ersetzt (Zeile 11-12). Falls der neu gebildete Kandidat nicht vollständig ist, aber besser werden kann als der schlechteste der Top-k-Liste, wird er in die Kandidatenliste aufgenommen, ansonsten verworfen (13-14). Der Algorithmus terminiert, falls kein Kandidat der Kandidatenliste mehr so erweitert werden kann, dass er besser wird als das Element mit dem niedrigsten Score in der Top-k-Liste (Zeile 18-19). Der *Bestscore* (Zeile 23) berechnet sich hierbei als die Summe der Scores für seine bereits ausgewerteten Dimensionen und die Scores der Kopfelemente der Listen der noch nicht ausgewerteten Dimensionen.

9 Prototypsystem

Die SphereSearch-Suchmaschine wurde als voll funktionsfähiger Prototyp implementiert. Die folgenden beiden Abschnitte geben einen Überblick über die Implementierung, gefolgt von Abschnitten, die sich den einzelnen Komponenten widmen.

9.1 Überblick

Der SphereSearch Prototyp ist vollständig in Java implementiert. Als Datenbank wurde *Oracle* in der Version *10g Enterprise* (Version 10.2.1.0) und als Application-Server *Tomcat* in der Version 4.0.6 eingesetzt. Die logischen Systemkomponenten wurden in die drei Teilapplikationen *Client*, *Anfrageprozessor* und *SphereSearch-Manager* gekapselt (Abbildung 9.1), die unabhängig voneinander über ein Web-Interface aufgerufen werden können .

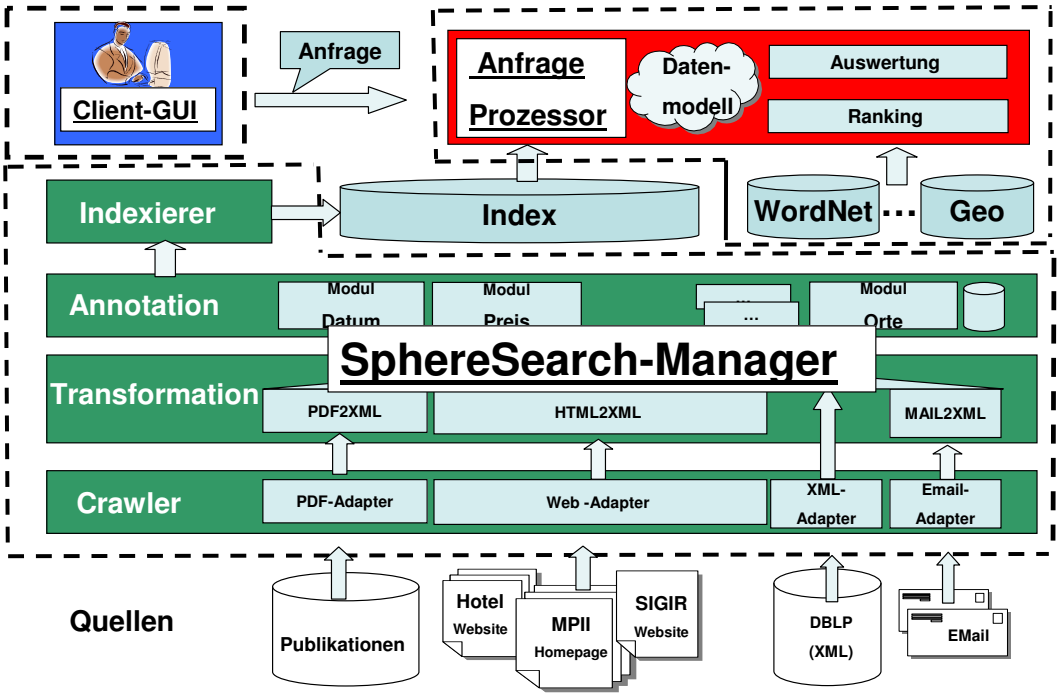


Abbildung 9.1: Systemarchitektur des SphereSearch-Prototyp

Ein Überblick über die logischen Komponenten wurde in Kapitel 3 gegeben.

- **Der Client**
  Der Client gestattet die graphische Konstruktion von Anfragen ohne Kenntnis der zugrunde liegenden Anfragesprache. Er ist als Java-Applet implementiert und kann aus diesem Grund in allen gängigen Web-Browsern ausgeführt werden.

- **Der Anfrageprozessor**
  Der Anfrageprozessor dient zur interaktiven Auswertung von Anfragen, die vom Client übermittelt werden. Des Weiteren generiert er sowohl die Ergebnisse der Anfrage als auch Feedback-Anfragen des Ontology-Service bzw. des Geo-Moduls und stellt sie in einem Web-Browser dar. Er ist als JAVA-Servlet implementiert. Als Servlet-Container kommt ein Tomcat Application-Server in der Version 4.0.6 zum Einsatz.

- **Der SphereSearch-Manager (SSM)**

  Der SphereSearch-Manager ist eine Applikation, die zahlreiche Module für alle zentralen Aufgaben zur Erstellung eines Suchindex umfasst. Er beinhaltet Module für die folgenden Aufgaben:

  - **Crawler-Steuerung**
    Das Crawler-Modul dient zu Steuerung und zur Beoabachtung des Crawling-Prozesses. Der Crawler wird über dieses Modul mit den Start-URLs versorgt, kann gestartet und beendet werden. Zahlreiche Parameter steuern den Crawling-Vorgang. Es können beispielsweise reguläre Ausdrücke definiert werden, um bestimmte Muster in den zu indexierenden URLs aus- oder einzuschließen. Des Weiteren kann der aktuelle Status jedes Prozesses beobachtet werden.
  - **Steuerung und Konfiguration des Annotationsprozesses**
    Über das Annotationsmodul können die indexierten Dokumente um Annotationen erweitert werden. Sowohl die hinzuzufügenden Annotationen als auch die hierfür zu generierenden Tags können konfiguriert werden. Über die integrierte GATE-Applikation können einfach weitere Annotationskomponenten, z.B. durch die Integration weiterer Lexika (Gazetteer-Listen), hinzugefügt werden.
  - **Schemaerstellung**
    Über das Schemamodul können die Datenbankrelationen für einen Suchmaschinenindex erstellt, entfernt, oder Berechnungen auf diesem (z.B. TF/IDF-Statistiken) angestoßen werden. Es können beliebig viele unabhängige Suchmaschinenindizes verwaltet werden.

– **Verwaltung der Einstellungen**
Die zahlreichen Einstellungen und Parameter des Systems (ca.100) können über das Konfigurationsmodul verwaltet werden. Sie werden in Form einer XML-Datei abgelegt, deren Gültigkeit mit Hilfe eines XML-Schemas überprüft wird.

– **Automatisierte Evaluation**
Das Evaluationsmodul dient zur automatisierten Abarbeitung von Anfragen. Diese können in einer hierfür dedizierten Datenbankrelation abgelegt und automatisch ausgeführt werden. Außerdem existiert eine Unterkomponente zur Abarbeitung von Anfragen im NEXI-Format des INEX-Benchmarks.

– **Fehleranalyse**
Das System speichert zahlreiche Ausgaben in einem XML-Format. Ein spezieller Log-Datei-Betrachter dient zur Analyse der generierten Log-Dateien. Zum besseren Debugging kann der Umfang der zur erzeugenden Log-Ausgaben konfiguriert werden.

9.2 Klassen- und Pakethierarchie

Das SphereSearch-Prototypsystem besteht aus insgesamt 418 implementierten Klassen mit zusammen 45.421 Zeilen Code. Die Anzahl der Code-Zeilen ist hierbei als reine Zeilen Code ohne Kommentare und Leerzeilen angegeben[1]. Tabelle 9.1 gibt einen Überblick über die Paket- und Klassenhierarchie. Zur besseren Übersichtlichkeit sind nur die wichtigsten Pakete (inkl. ihrer Unterpakete) zusammen mit der Anzahl der enthaltenen Klassen und Code-Zeilen angegeben.

Die Funktionen der jeweils in Paketen zusammengefassten Klassen werden im Folgenden erläutert. Das Paket `awsl.base` beinhaltet Klassen zur Verwaltung der Konfiguration und benötigt die Data-Binding Bibliotheken `jaxb-*` und die Editor-Bibliothek `XAMPLE` ([XAMP]). Das Paket `awsl.classifier` dient zur Integration des BINGO!-Klassifikators [STSW02]. Für die beiden Benutzeroberflächen des SphereSearch-Managers (SSM) und des Client-Applets sind die Pakete `awsl.client` bzw. `awsl.client.gui` zuständig. Letzteres benötigt zur Darstellung die `JHotDraw`-Bibliothek (siehe Abschnitt 9.3). Web- und File-Crawler-Funktionalität sind in den Paketen `awsl.crawler` und `awsl.file` enthalten, wobei ihre grafische Benutzeroberfläche Teil des SSM ist. Das Paket `awsl.db` kapselt die Datenbankinteraktion. Das Unterpaket `.oracle` beinhaltet in Kombination mit der Oracle-Bibliothek *ojdbc14* hierbei die Oracle-Datenbankanbindung. Bei einer Portierung auf ein anderes Datenbanksystem muss somit nur ein neues Unterpaket für die

[1] Die reine Anzahl an Zeilen inklusive Leerzeilen und Kommentaren ist im Vergleich hierzu mit 87.311 Zeilen fast doppelt so groß.

Paket	# Zeilen	#Klassen	Aufgabe
awsl.base	1505	5	Konfiguration
awsl.classifier	711	13	Wrapper für Klassifikator
awsl.client.ssm	4398	26	Benutzer-Interface des SSM
awsl.client.gui	2328	22	Graphisches Client-Gui
awsl.crawler	1370	25	Web-Crawler
awsl.db	5660	24	Datenbankschnittstelle
awsl.files	261	4	File-Crawler
awsl.gate	1278	12	Gate-Integration
awsl.index	2260	26	Indexierer
awsl.logger	364	8	Logging
awsl.queryprocessing	10070	56	Anfrageverarbeitung
awsl.recover	609	12	Dokumentrekonstruktion
awsl.rules	1338	10	Regeln für HTML2XML
html2xml	6896	96	HTML2XML
de.mpii.graph.mst	1630	11	MST-Berechnung
de.mpii.inex	282	2	INEX-Anbindung
awsl.dbGeoQuery	432	6	Geo-Modul
....	4129	60	übrige Klassen
gesamt	45421	418	

Tabelle 9.1: Klassenstruktur

Bibliothek	Aufgabe
ojdbc14	Oracle-Anbindung (JDBC)
gate	GATE
xmlparserv2	Oracle XML-Parser
jaxb-*	Data-Binding
jhotdraw	Applet-GUI
ontology	Ontology-Modul
xample	XML-Editor
log4j	Fehler-Logging

Tabelle 9.2: Wichtige Bibliotheken

entsprechende Datenbank implementiert und konfiguriert werden. Die Integration der GATE-Bibliothek wird durch das Paket `awsl.gate` bewerkstelligt. Zur besseren Fehleranalyse wurde die Logging-Bibliothek `Log4J` eingesetzt und über das `awsl.logger`-Paket angepasst und eingebunden. Die Anfrageverarbeitung und Auswertung findet im `awsl.queryprocessing`-Paket statt, wobei zusätzlich das Paket `de.mpii.graph.mst` zur Umsetzung des Top-k-Algorithmus zur MST- und Gesamt-Score-Bestimmung benötigt wird.

Zur Darstellung von Dokumentenfragmenten und Dokumenten in der Ergebnisdarstellung, wie sie im internen Index repräsentiert sind (im internen XML-Format inklusive ihrer Annotation), werden die Klassen des Pakets `awsl.recover` benutzt. Die Pakte `html2XML` und `awsl.rules` enthalten die Grundfunktionalität und spezielle Regeln des Transformationsmoduls zur Umwandlung von HTML nach XML (HTML2XML). Das Geo-Modul zur Auswertung geographische Anfragen ist im Paket `awsl.dbGeoQuery` implementiert. Für die INEX-Anbindung ist schließlich das Paket `de.mpii.inex` zuständig. Die erwähnten Bibliotheken sind in Tabelle 9.2 dargestellt und kurz erläutert.

9.3 Der Client

Der Client ist als Java-Applet implementiert, das in jedem gängigen Browser ausgeführt und dargestellt werden kann. Der Vorteil einer als Applet implementierten Lösung ist, dass einerseits keine Client-seitige Installation einer Applikation erfolgen muss, aber auf der anderen Seite die Mittel einer vollwertigen Applikation hinsichtlich Benutzeroberfläche und Funktionalität zur Verfügung stehen. Somit ist eine solche Lösung einer Browser-basierten (D)HTML-Lösung in vielerlei Hinsicht überlegen.

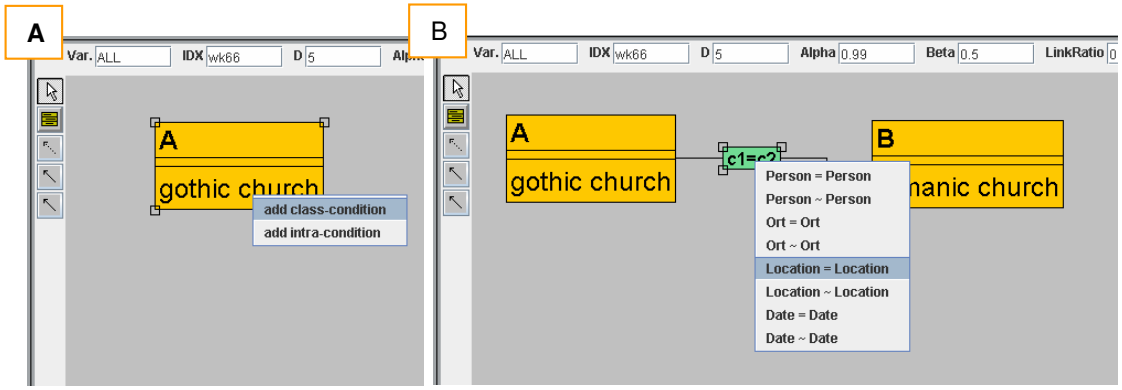


Abbildung 9.2: Das Java-Applet des Client-GUI

Das Applet stellt eine Zeichenfläche (Abbildung 9.2) zur Verfügung, in der intuitiv Anfragen erstellt werden können. Hierzu platziert der Benutzer eine neue Anfragegruppe per Mausklick(Abbildung 9.2.A), der über ein Kontextmenü verschiedene Teilbedingungen hinzugefügt werden können, auf der Zeichenfläche. Nachdem der Benutzer beliebig viele Anfragegruppen erstellt hat, kann er Join-Bedingungen zwischen Anfragegruppen spezifizieren (Abbildung 9.2.B), indem er mit der Maus zwei Anfragegruppen miteinander verknüpft. Anschließend kann die Join-Bedingung spezifiziert werden, indem der Benutzer eine der vorgegebenen Bedingungen auswählt

oder selbst eine beliebige Bedingung angibt. Außerdem können über diesen Client die die Anfrageauswertung beeinflussenden Parameter spezifiziert bzw. modifiziert oder der Index ausgewählt werden, auf dem die Anfrage ausgewertet werden soll. Für alle Parameter sind sinnvolle Werte voreingestellt, so dass diese Anpassungen optional vorgenommen werden können.

Diese Form der Anfrageerstellung hat den Vorteil, dass keinerlei Kenntnis der Anfragesprache erforderlich ist und die Einarbeitungszeit minimiert wird. Zur Implementierung des Clients wurde die JHotDraw [JHOT] Bibliothek genutzt, die die Erstellung solcher Applikationen mit einer Zeichenfläche und beliebig positionierbaren Objekten vereinfacht.

9.4 Der Crawler

Der Crawler ist als Multithread-Applikation implementiert, d.h. dass jedes Dokument von einem eigenen Thread verarbeitet wird. Die Anzahl der Threads und somit die Anzahl der gleichzeitig verarbeiteten Dokumente ist konfigurierbar.

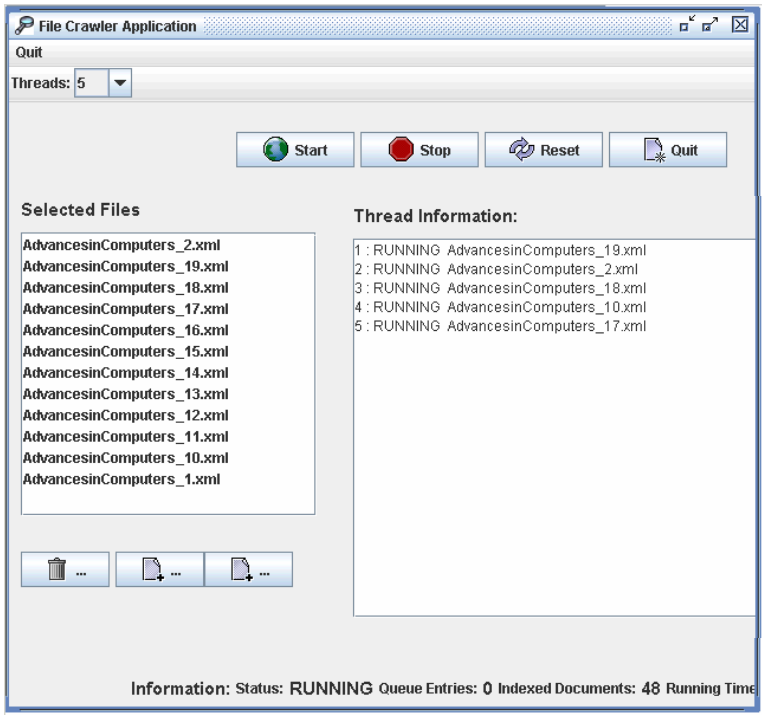


Abbildung 9.3: File-Crawler

Der Crawler verfügt über zwei unterschiedliche Funktionsmodi, die jeweils mit einem eigenen Benutzer-Interface zur Steuerung verbunden sind: den File-Crawler-

und den Web-Crawler-Modus.

Im File-Crawler-Modus können dem Crawler einzelne Dateien oder ganze Verzeichnisse übergeben werden. Der Crawler durchsucht diese Verzeichnisse rekursiv und fügt alle gefunden Dateien mit unterstütztem Datentyp (XML, HTML, PDF) der Crawler-Queue hinzu. Ausschließlich die in diesem Schritt gefundenen Dateien werden vom Crawler gelesen und der Indexierungskomponente übergeben. Links innerhalb der Dokumente werden nicht verfolgt, aber aufgelöst. Die Verarbeitung von Links wird im nächsten Unterabschnitt behandelt.

Der Web-Crawler Modus ist wesentlich leistungsfähiger und es stehen in diesem Modus zahlreiche Optionen zur Verfügung. Im Gegensatz zum File-Crawler werden beim Web-Crawler nur Start-URLs spezifiziert, aus denen der Crawler Links zu weiteren Dokumenten extrahiert und diesen abhängig von der aktuellen Konfiguration folgt. Es ist konfigurierbar, ob der Crawler eine Tiefen- oder Breitensuche beim Crawl-Vorgang durchführt.

URLs können hierbei alle Protokolle beinhalten, die vom System aufgelöst werden können und ohne weitere Schritte (insbesondere eine Authentifikation) zugreifbar sind. So können Dokumente auf Web-Servern (*HTTP://*), lokal zugreifbaren File-Systemen (*FILE://*) oder auch FTP-Servern (*FTP://*) verarbeitet werden.

Abbildung 9.4 zeigt das Interface des Web-Crawler-Moduls. Folgende Einstellungen und Visualisierungen sind der Abbildung zu sehen.

1. Crawler-Implementierung: An Stelle der internen Crawler-Implementierung kann der BINGO!-Crawler [STSW02] ausgewählt werden. Dieser ermöglicht fokussiertes Crawling ([CBD99]) auf Basis einer vorgegebenen Klassenhierarchie [2].

2. Tiefe: Spezifiziert die Tiefe, bis zu der, ausgehend von den Startdokumenten, Links verfolgt und die entsprechenden Dokumente indexiert werden sollen.

3. Anzahl der Crawler-Prozesse: Jeder Prozess verarbeitet ein Dokument. Mit steigender Zahl an parallelen Prozessen steigt der Durchsatz an, bis andere system- und umgebungsspezifische Faktoren eine weitere Steigerung begrenzen. Je nach Anbindung, zur Verfügung stehender Rechenleistung oder Arbeitsspeicher variiert der optimale Wert (typische Werte: 25-50).

4. Steuerung: Über diese Knöpfe wird der Crawler gestartet, manuell beendet, oder wieder in den Initialzustand zurückgesetzt.

5. Start-URLs: Zeigt die Startdokumente des Crawl-Vorgangs an.

[2] Zu weiteren Details von BINGO! und zum Konzept des fokussierten Crawlens sei auf die referenzierte Literatur verwiesen, da dies kein Kernpunkt dieser Arbeit ist.

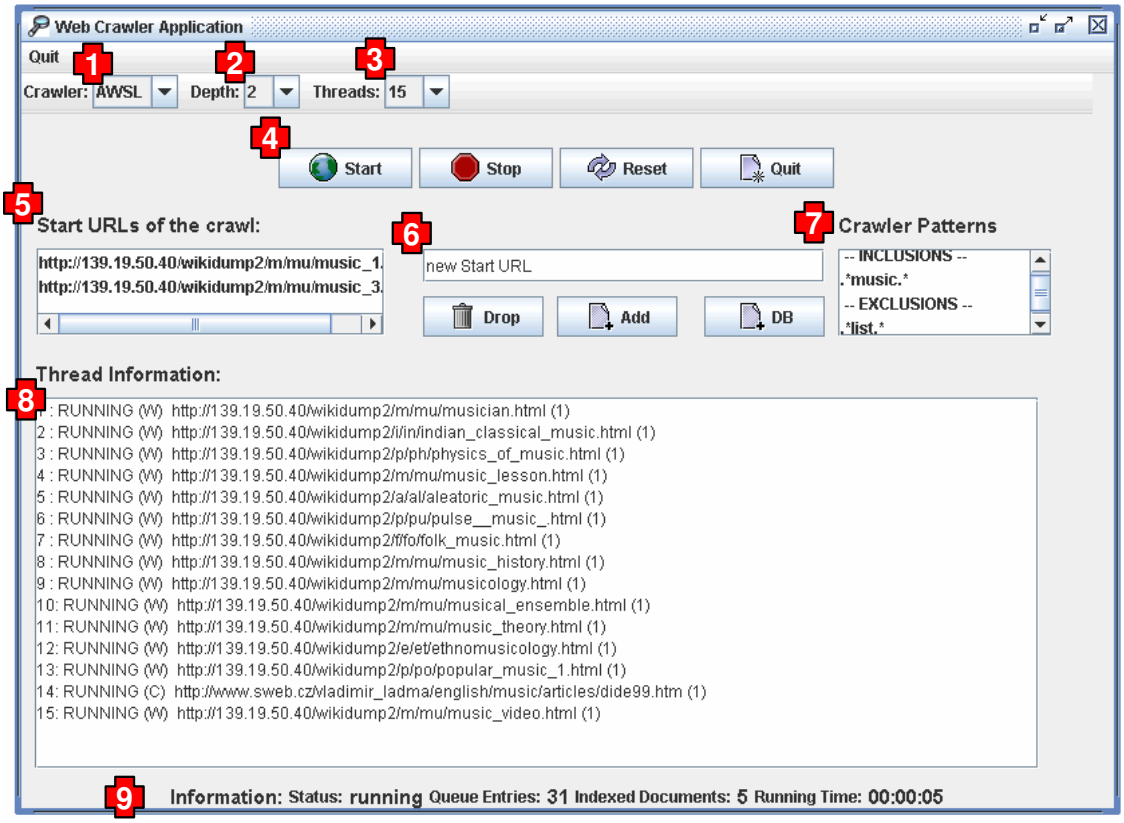


Abbildung 9.4: Web-Crawler

6. Start-URL-Eingabe: Die Start-URLs werden zunächst aus einer Konfigurationsdatei gelesen, können allerdings nachträglich bearbeitet werden. Durch den DB-Knopf können auch die Start-URLs aus einer Datenbank gelesen werden. Das aufgerufene SQL-Kommando wird in der Konfigurationsdatei angegeben.

7. Reguläre Ausdrücke zur Crawl-Steuerung: Es können reguläre Ausdrücke spezifiziert werden um während des Crawl-Vorgangs bestimmte URLs auszuschließen bzw. nur bestimmte URLs zu akzeptieren.

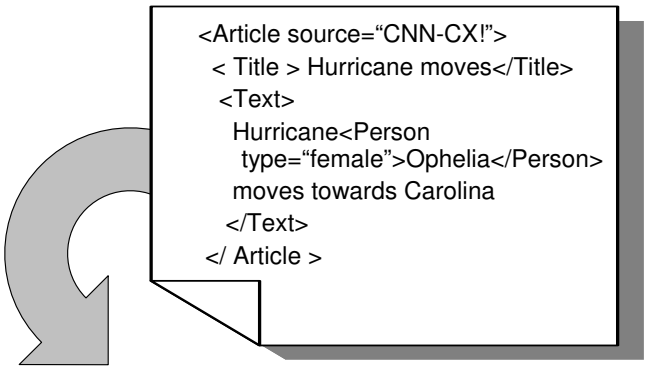
8. Prozessinformation: Für alle Crawler-Prozesse wird das aktuell bearbeitete Dokument sowie der Zustand des Prozesses dargestellt.

9. Crawler-Status: Der aktuelle Status des Crawls (Anzahl verarbeiteter Dokumente, Anzahl der Dokumente in der Queue, aktuelle Laufzeit, etc.) wird an dieser Stelle angezeigt.

Der Crawl terminiert, falls die Crawler-Queue leer ist, der Crawl manuell abgebrochen wurde, oder eine in der Konfigurationsdatei spezifizierte Maximalanzahl an Dokumenten verarbeitet wurde.

9.5 Der Indexer

Da bei SphereSearch die Struktur der Dokumente erhalten bleiben muss, werden die Dokumente verlustfrei indexiert, d.h. die indexierten Dokumente können vollständig im XML-Zwischenformat aus dem Index wiederhergestellt werden[3]. Jeder Knoten wird hierbei als mindestens ein Tupel in der zentralen Dokumentrelation *Objects* gespeichert. Zur Kodierung der Knotenposition jedes einzelnen Knotens im Ursprungsdokument wird im Prototyp das Dewey-Labeling verwendet (siehe Abschnitt 2.2.2). Abbildung 9.5 zeigt ein einfaches XML-Dokument und seine Repräsentation im Index [4].



Object_ID	Attribute_ID	Token_ID	Object_Type	Token	Content
1	0	0	1	Article	
1	1	0	3	Source	CNN-CX!
1	1	1	3	CNN	
1	1	2	3	CX	
1.1	0	0	1	Title	
1.1.1	0	0	2		Hurricane moves
1.1.1	0	1	2	Hurricane	
1.1.1	0	2	2	Moves	
1.2	0	0	1	Text	
1.2.1	...				
...					

Abbildung 9.5: Indexiertes Dokument

Im Attribut *Object_ID* der Relation wird das Dewey-Label des Knotens gespeichert. *Object_Type* spezifiziert den Typ des indexierten Elements: 1 für Tags, 2 für

[3]Es werden allerdings oft nicht die Ursprungsdokumente exakt wiederhergestellt, da der Transformationsprozess verlustbehaftet und nicht umkehrbar ist. XML-Dokumente allerdings können, weil für diese keine Transformation stattfindet, exakt wiederhergestellt werden.

[4]Zur besseren Übersichtlichkeit wurden einige Spalten der Datenbankrelation an dieser Stelle weggelassen. Ein vollständige Dokumentation der wichtigsten Relationen ist in Abschnitt 9.8 zu finden.

Text-Elemente und 3 für Attribute. Für Elemente, die längere Textabschnitte (auch mit Sonderzeichen) enthalten, also für Attributwerte und Textelemente, wird redundant zunächst der gesamte Inhalt unverändert als *Character Large Object*(CLOB) in der Spalte *Content* und zusätzlich die einzelnen Bestandteile als weitere Tupel, durchnummeriert über das Attribut *Token_ID*, im Attribut *Token* gespeichert. Das Attribut *Token* enthält hierbei das vorverarbeitete Element (z.B. nach Wortstammreduktion), auf dem die eigentlichen Suchoperationen durchgeführt werden. Das Attribut *Content* ermöglicht die Rekonstruktion des Originaldokuments und wird für Operationen ausgewertet, die den originären textuellen Inhalt inklusive aller Sonder- und Satzzeichen erfordern.

Auch wenn komplexere Teilanfragen ausgewertet werden, werden zunächst Knoten ermittelt, die die in der Anfrage spezifizierte Terme enthalten. Dies geschieht in SphereSearch, wie auch in den meisten Suchmaschinen, über invertierte Listen. Eine invertierte Liste ordnet jedem Schlüsselwort die Dokumente bzw. Elemente zu, die diesen Term enthalten. In SphereSearch werden die invertierten Listen durch Index-Strukturen der zugrundeliegenden Datenbank erzeugt. Es handelt sich hierbei um B*-Bäume, die Standard-Indexstrukturen relationaler Datenbanken.

Einer der oft vom System genutzten Indizes auf der zentralen *OBJECTS*-Relation, die die indexierten Dokumente beinhaltet, ist der *TOKEN_ALL*-Index. Das SQL-Statement zu seiner Erstellung sieht aus wie folgt:

```
CREATE INDEX IndexName_TOKEN_ALL ON IndexName_OBJECTS
(TOKEN, IU_ID, OBJECT_ID, TOKEN_ID, ATTRIBUTE_ID, OBJECT_TYPE);
```

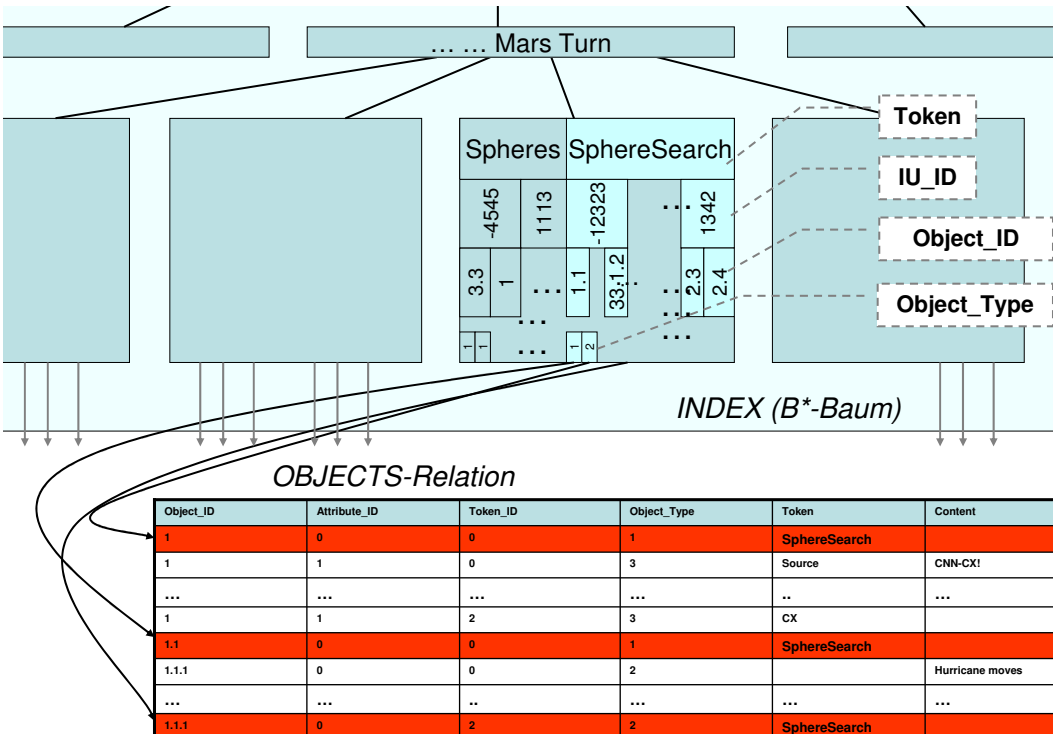
Eine typische Anfrage, bei deren Ausführung dieser Index genutzt wird, ist die folgende, die nach allen Elementen mit dem Vorkommen des Terms „spheresearch“ sucht:

```
SELECT IU_ID, Object_ID, TOKEN_ID, ATTTRIBUTE_ID, OBJECT_TYPE
FROM index
WHERE TOKEN = 'spheresearch'
```

Auch wenn die Filter-Bedingung in der *WHERE*-Klausel der Anfrage nur das Attribut *TOKEN* nutzt, so enthält der Index zusätzlich auch alle Attribute der *SELECT*-Klausel der Anfrage. Abbildung 9.6 veranschaulicht dies. Sie zeigt einen Teil der Index-Blattknoten mit ihren Verweisen auf die *OJBECTS*-Relation.

Diese Abstimmung von Indizes und Anfragen ermöglicht es, dass die Anfragen ausschließlich mit Hilfe des Index, ohne Zugriffe auf die Relation selbst, beantwortet werden können. Die meisten Anfragen und Indizes im System sind in dieser Art aufeinander abgestimmt.

So kann die obige Anfrage durch nur einen *Range-Scan* auf der Blatt-Ebene des



Object_ID	Attribute_ID	Token_ID	Object_Type	Token	Content
1	0	0	1	SphereSearch	
1	1	0	3	Source	CNN-CX!
...	...	...	...	..	...
1	1	2	3	CX	
1.1	0	0	1	SphereSearch	
1.1.1	0	0	2		Hurricane moves
...	...	..	...	...	...
1.1.1	0	2	2	SphereSearch	

Abbildung 9.6: Index und Relation

zum $TOKEN\_ALL$-Index gehörenden B*-Baums ausgewertet werden, also als reine *Index-Only-Anfrage*. Somit müssen keinerlei wahlfreie Zugriffe auf der *OBJECTS*-Relation, sondern nur sequentielle Zugriffe auf dem Index selbst durchgeführt werden. Eine solche Auswertung ist um ein vielfaches effizienter als ein Anfrage mit Indexbenutzung und anschließenden Zugriff auf die zugrundeliegende Relation.

9.5.1 Indexierung von Links

Links zwischen Dokumenten werden in der Relation *Link_Structure* gespeichert. Diese enthält für Quelle und Ziel eines Links die ID des Dokuments (ein aus der URL-Repräsentation generierter Hash-Wert) und die ID des Knotens selbst.
Falls es sich bei dem Link um einen einfachen (X)Link wie den folgenden handelt:

```
<article xlink:type=''simple'' xlink:href=''jg-02.xml''/>
```

kann dieser sehr einfach wie folgt in der Relation *Link_Structure* abgelegt werden.

SOURCE_URI_ID	SOURCE_NODE_ID	TARGET_URI_ID	TARGET_NODE_ID	XPATH
-24434545	1.12.3	-12395877	1	

Als Zielknoten wird bei solchen einfachen Links der Wurzelknoten des Zieldokuments (Dewey-Label 1) eingetragen. Es ist zu beachten, dass in diesem Fall das Zieldokument noch nicht indexiert sein muss. Die Indexierung gestaltet sich bei komplexeren Links, die nicht nur ein Dokument referenzieren, sondern einen spezifischen Knoten innerhalb eines Zieldokuments, deutlich aufwändiger.
In folgenden Beispiel referenziert der Link im Dokument `jg-02.xml` den Knoten, der durch den XPATH `//pub[conf='VLDB']` spezifiziert ist[5].

```
<article xlink:type='simple'
xlink:href='jg-02.xml#xpointer(//pub[conf='VLDB'])'/>
```

Falls in diesem Beispiel das Zieldokument noch nicht indexiert wurde, ist auch das Dewey-Label des Zielknotens nicht bekannt. Aus diesem Grund wird zunächst ein temporärer Eintrag erzeugt und über das Attribut *XPath* als noch auszuwertender XPath-Ausdruck markiert.

SOURCE_ID	SOURCE_NODE_ID	TARGET_ID	TARGET_NODE_ID	XPATH
-24434545	1.12.3	-12395877	pub[conf='VLDB']	X

Falls nun das Zieldokument des Links indexiert wird, wird dieser und alle anderen auf dieses Dokument verweisenden Links mit noch nicht ausgewerteten XPath-Ausdrücken ausgewertet und die Einträge in der Link-Relationen entsprechend aktualisiert.

Da für die Auswertung von Anfragen nicht nur direkt miteinander verbundene Dokumente sondern u.U. auch transitiv verbundene Dokumente betrachtet werden, wird die transitive Hülle bis zu einer konfigurierbaren Tiefe (meist 2 oder 3 Dokumente) nach erfolgter Indexierung vorberechnet und in der Relation *Transitive_Closure* abgelegt.

9.6 Annotation

Annotationen können Dokumenten erst nach ihrer Indexierung hinzugefügt werden. Eine Annotation der Dokumente während des Crawl-Vorgangs und der Indexierung hätte diese Vorgänge zu sehr verlangsamt. Durch die nachgelagerte Annotation können außerdem gezielt Dokumente mit bestimmten Eigenschaften (Begriffe in ihrem Inhalt, Länge etc.) zur Annotation selektiert werden.

Die Annotationen werden dem Index mit Hilfe der integrierten Informationsextraktionskomponente ANNIE des NLP-Frameworks GATE hinzugefügt. Die zugrundeliegende Bibliothek musste hierzu so angepasst werden, dass Dokumente

[5] Prinzipiell kann dieser Link beliebig viele Knoten referenzieren. Im SphereSearch-Prototyp wird allerdings immer nur der erste referenzierte Knoten berücksichtigt.

direkt als Objekte, die aus dem Index generiert werden, zur Annotation übergeben werden können. Die GATE-Bibliothek selbst gestattet nur die Übergabe der URL eines Dokuments.

Zusätzlich zu den schon integrierten Annotationskategorien (Personen, Orte, Datumsangaben, Geldbeträge, Organisationen) können neue Kategorien erstellt werden. Dies geschieht über die Benutzeroberfläche von GATE, die in den SSM integriert ist und über diesen aufgerufen werden kann. Über die Oberfläche von GATE können neue Gazetteer-Listen oder JAPE-Grammatiken hinzugefügt werden (siehe auch Abschnitt 2.3.3). Abbildung 9.7 zeigt das über den SSM aufgerufenen Annotationsmodul von GATE. In der Abbildung ist die Ansicht zur Bearbeitung der Gazetteer-Listen zu sehen.

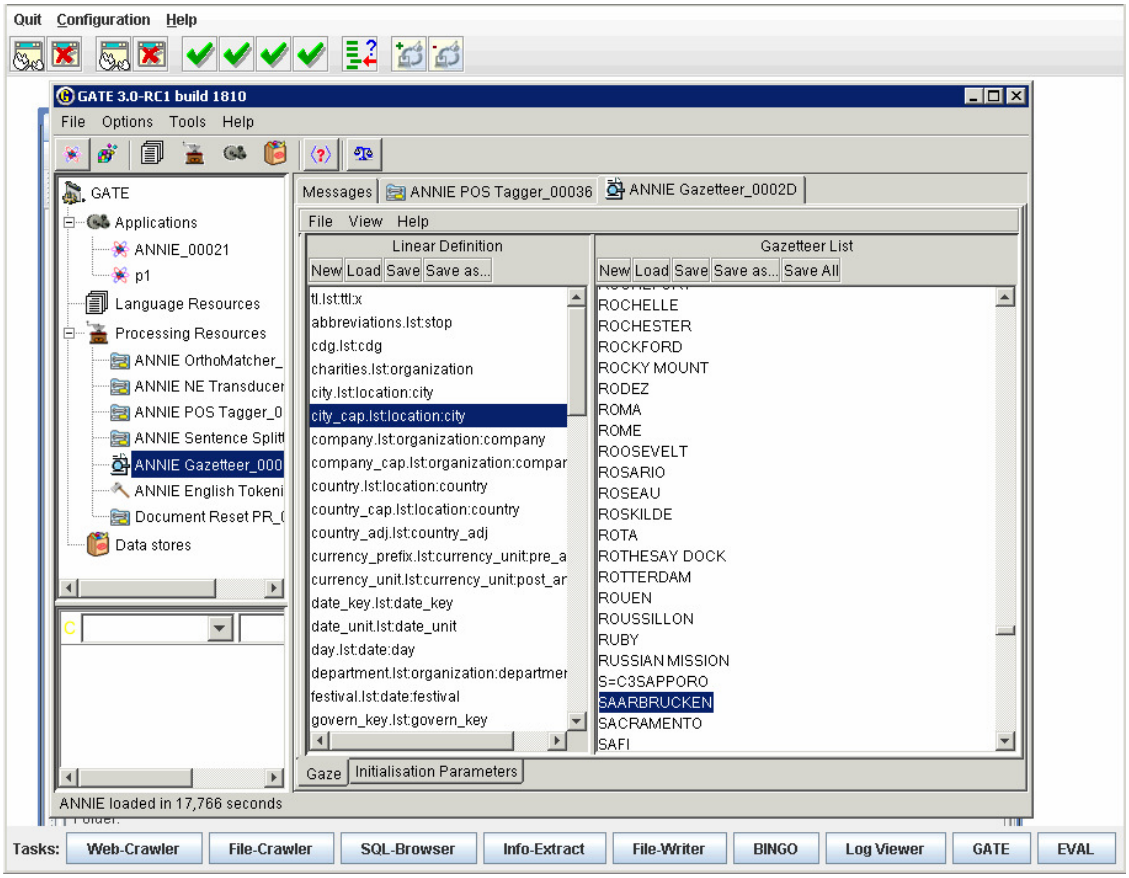


Abbildung 9.7: GATE-Einbettung im SSM

Annotationen können auf zwei Arten erzeugt werden:

- *Inline*: Annotationen werden als Tags in die indexierten Dokumente eingefügt. Anschließend ist eine Unterscheidung zwischen Tags des Ursprungsdokuments

und durch den Annotationsprozess eingefügten Tags nicht mehr möglich.

- *Extern*: Annotationen werden mit Verweisen auf die annotierten Positionen im Originaldokument in typspezifischen Datenbankrelationen gespeichert.

Abbildung 9.8 zeigt den Steuerdialog des Annotationsprozesses. Für jeden Annotationstyp kann die Abbildung auf einen Tag-Namen definiert werden. In der Liste, die in Abbildung 9.8 rechts zu sehen ist, wird zum Beispiel die Annotation *cp_c/loc/city* auf das Tag *LOCATION* abgebildet.

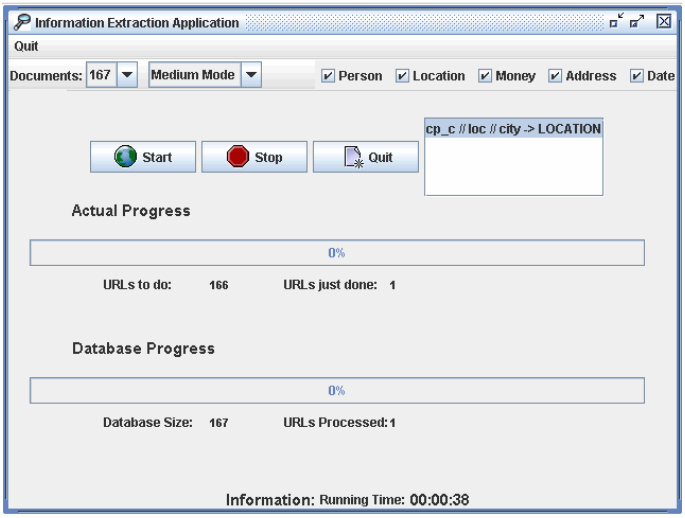


Abbildung 9.8: Benutzeroberfläche des Annotationsprozesses

Die externe Verwaltung von Annotationen ermöglicht insbesondere für numerische Objekte durch ihre optimierte Speicherung eine effizientere typspezifische Auswertung (vgl. Kapitel 5.2). Beispielsweise werden Datumsangaben in einer dedizierten Relationen als numerischer Datentyp (*Number*) gespeichert [6]. Zunächst werden hierfür alle Datumsangaben in eine kanonische Form überführt: $date\_value = year * 10000 + month * 100 + day$. Aufgrund dieser Speicherung kann eine Teilbedingung wie zum Beispiel $08/20/05 <= date <= 08/26/05$ einfach und effizient auf die SQL-Anfrage:

```
SELECT objects FROM date_annotations
               WHERE 20050820< date_value < 20050826
```

abgebildet werden. Details zur Annotation und Auswertung geographischer Angaben sind in Kapitel 9.7.1 zu finden.

[6] Es werden keine datenbankspezifischen Datentypen wie beispielsweise *Date* eingesetzt, damit sämtliche Annotationen in einer generischen Zahlen- oder Zeichenkettenspalte abgelegt werden können. Außerdem vereinfacht dies die Portierung auf ein anderes Datenbanksystem.

Gesamteinträge	4.334.146	Features USA	987.331
Verwaltungseinheiten	2.126.610	Features Europa	1.211.812
Flüsse	480.921	Features Deutschland	271.017
Berge	362.194	Features Saarland	37.915
Vulkane	2.262		
Ölfelder	4.838		

Tabelle 9.3: Statistische Daten zum ADL-Gazetteer

9.7 Typ-Module

Für die Auswertung datentypspezifischer Operatoren (vgl. Kapitel 5.2) sind dedizierte Module zuständig. In den folgenden beiden Unterabschnitten wird auf das Modul zur Auswertung von geografischen Operatoren (Typ *Location*) sowie das Modul zur Auswertung des die allgemeine semantische Ähnlichkeit repräsentierenden Ähnlichkeitsoperator des Typs *String*, der die quantifizierten WordNet-Ontologie nutzt (vgl. Kapitel 5.2.2), eingegangen.

9.7.1 Das Geo-Modul

Das Geo(graphie)-Modul dient zur Auswertung von Anfragen mit geographischen Anfragebedingungen. Es arbeitet auf den Daten des ADL Gazetteer ([Alex]) der Alexandria Digital Library, einem Projekt zum Aufbau einer digitalen Bibliothek der University California (UCSB).

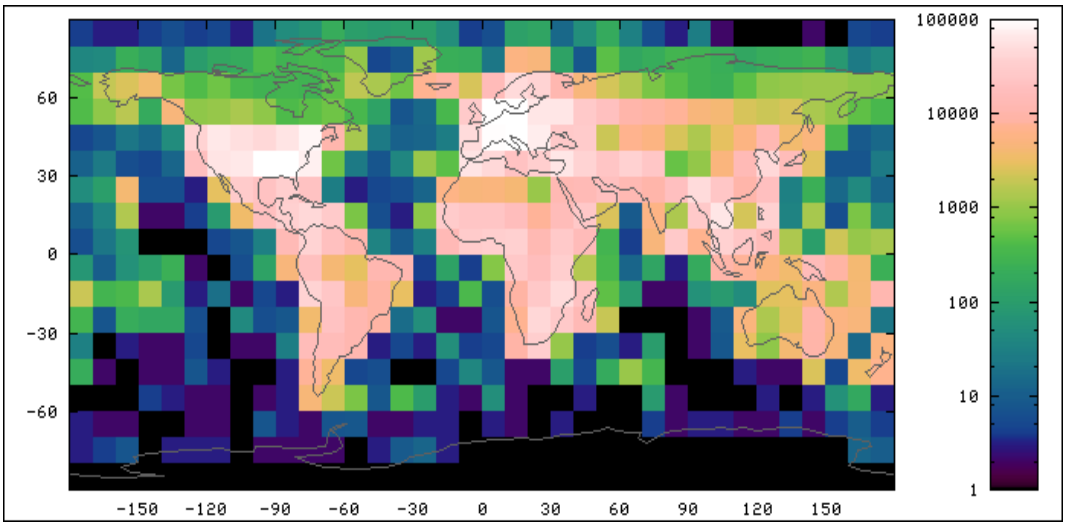


Abbildung 9.9: Abdeckung des ADL-Gazetteer

Im Gegensatz zu anderen verfügbaren geographischen Datensammlungen bietet

der Alexandria Gazetteer eine umfassende Abdeckung aller Regionen der Erde. Abbildung 9.9 visualisiert seine geographische Abdeckung. Die Farbe jedes Rechtecks der Karte reflektiert die Anzahl an Annotationen, die für diesen Bereich enthalten sind. Die Legende hierzu, die die Farben auf die entsprechende Anzahl abbildet, ist auf der rechten Seite zu sehen.

Der Gazetteer umfasst mehr als vier Millionen Einträge, die sowohl Städte, Orte und Regionen als auch Flüsse, Berge und Vulkane umfassen. Tabelle 9.3 zeigt hierzu einige statistische Angaben.

Beispielhaft wird im Folgenden aufgezeigt wie eine geographische Teilbedingung einer Anfrage mit Hilfe des Geo-Moduls ausgewertet wird. Eine Abfrage mit der geographischen Bereichsbedingung `Location = Rio de Janeiro - Salvador` (vgl. Abschnitt 6.2) läuft ab wie folgt:

1. Zunächst werden die Orte der Anfrage im Geo-Index zusammen mit ihren Eigenschaften nachgeschlagen. Falls die Ortsbezeichnungen nicht eindeutig sind, ist ein Feedback-Schritt zwischengeschaltet, in dem der Benutzer die korrekten Ortsangaben auswählen kann (Abb 9.10)

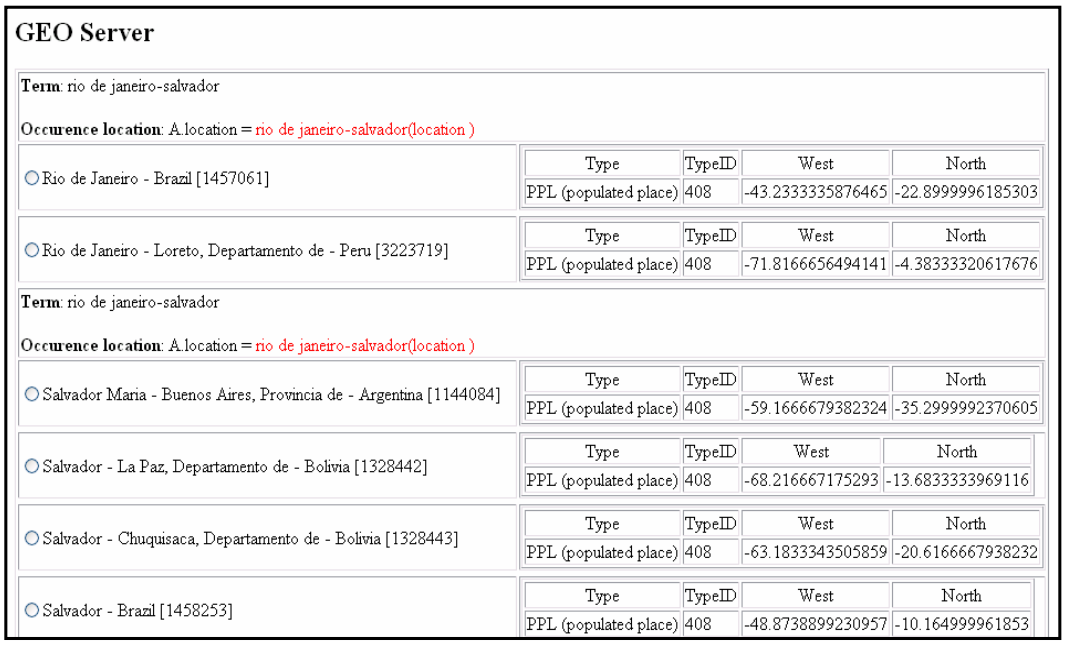


Abbildung 9.10: Feedback-Anfrage des Geo-Moduls

Dem Benutzer werden hierbei nicht nur Koordinaten präsentiert, um den gewünschten Ort auszuwählen, sondern auch aussagekräftige hierarchische Informationen. Für Orte habe diese beispielsweise die Form *Stadt/Region/Land*.

2. Wurden die Orte und ihre Koordinaten eindeutig bestimmt, werden anschließend in der Geo-Datenbank weitere Orte bestimmt, die zwischen den zugehörigen Koordinaten liegen. Im gegebenen Beispiel müssen die gesuchten Orte die Bedingungen $38.3 < longitude < 43.1$ und $13.0 < latitude < 22.5$ erfüllen [7]

3. Schließlich wird die ursprüngliche Anfrage um die so gefundenen Orte expandiert. Das Gewicht für die hinzugefügten Terme wird auf 1 gesetzt, da alle gefundenen Orte die Bedingung gleich gut erfüllen.

Wird die geographische Teilbedingung nicht als Bereichsbedingung, sondern als einfache Ähnlichkeitsbedingung der Form $\sim$ *Salvador* spezifiziert, werden Orte in einem Umkreis um den spezifizierten Ort bis zu einer maximalen Entfernung ermittelt. Hierbei entspricht (anders als bei der Bereichsbedingung) das Gewicht der durch die Expansion der Anfrage hinzugefügten Orte bzw. Terms dem Abstand zum spezifizierten Ort: je weiter ein Ort entfernt ist, desto geringer das Gewicht.

9.7.2 Das Ontologie-Modul

Das Ontologie-Modul dient zur Bestimmung der allgemeinen semantischen Ähnlichkeit zwischen Begriffen, die durch den $\sim$-Operator des Datentyps *String* spezifiziert wird. Es basiert auf der quantifizierten WordNet-Ontologie, die in Kapitel 2.4 beschrieben ist.

Eine typische Beispielanfrage mit Benutzung des $\sim$-Operators, bei der das Ontologie-Modul aufgerufen wird, ist die Anfrage: $A(\sim car, accident)$. Theoretisch müsste bei der Auswertung dieser Anfrage die Ähnlichkeit von *car* mit jedem Begriff jedes potentiellen Trefferdokuments bestimmt werden. Da dies aufgrund des Aufwands nicht praktikabel ist, wird der pragmatischere Weg gegangen und die Anfrage um ähnliche Begriffe expandiert. Viele Begriffe sind allerdings mehrdeutig, womit die Expansion mit ähnlichen Begriffen erst nach Bestimmung der tatsächlichen Bedeutung des Begriffs möglich ist. Da die Auflösung vom Mehrdeutigkeiten, die Disambiguierung, ein nicht-triviales Problem ist (für Details siehe [Theo06]), welches auch unter Umständen nicht auf Basis der Anfrage selbst gelöst werden kann, ist ein Feedback-Schritt zwischengeschaltet. In diesem Schritt kann der Benutzer das passende *SynSet*(vergleiche Kapitel 2.4) zu dem mit $\sim$-Operator kombinierten Suchterm, das die vom Benutzer intendierte Bedeutung repräsentiert, auswählen. Abbildung 9.11 zeigt eine Feedback-Rückfrage des Systems, nachdem der Benutzer die obige Anfrage gestellt hat.

[7] Diese Implementierung im Prototypen, bei der Orte innerhalb eines Rechtecks, dessen Eckpunkte die Ortskoordinaten sind, gesucht werden, ist nicht optimal, wurde aber wegen des geringen Berechnungsaufwandes gewählt.

Abbildung 9.11: Feedback-Anfrage des Ontologie-Moduls

Das System bietet für den Begriff *car* die Synsets zu den Konzepten Automobil, Aufzugkabine und Gondel an, für die in der englischen Sprache der Begriff *car* verwendet werden kann.

Hat der Benutzer ein Synset gewählt, werden die ähnlichsten Hypernyme, Hyponyme, Meronyme und Holonyme, deren Ähnlichkeitswert über einem spezifizierten Schwellwert liegt, angezeigt. Der Benutzer kann nun manuell Gewichte verändern, Begriffe von der Expansion ausschließen, oder den Systemvorschlag übernehmen. Die in diesem Fall nach dem Feedback-Schritt an das System übermittelte Anfrage sieht aus wie folgt, wobei hinter dem expandierten Begriff die entsprechenden der Anfrage hinzugefügten Begriffe mit Gewichten angegeben sind.

$$car(auto/1.0, automobile/1.0, car/1.0, machine/1.0, motorcar/1.0,\\ electric/0.16665901, electricautomobile/0.16665901, electriccar/0.16665901,\\ sportcar/0.1721242, sportscar/0.1721242, stockcar/0.14975625), accident$$

Nach Abschluss des Feedback-Vorgangs werden nun auch Dokumente berücksichtigt, die nicht nur den spezifizierten Suchbegriff, sondern auch einen durch die Ontologie hinzugefügten Begriff beinhalten. Die Ähnlichkeitsgewichte finden direkten Eingang in die Score-Berechnung.

9.8 Datenbankschema

9.8.1 Wichtige Relationen

Das zugrunde liegende Datenbankschema besteht aus insgesamt 12 Tabellen und insgesamt 18 Datenbankindizes. Die zugehörigen Relationen lassen sich drei logischen Teilbereichen zuordnen. Der Objektindex enthält die vollständige Information der ursprünglichen Dokumente, der Annotationsindex enthält die durch den Annotationsprozess ergänzten Annotationen und der Verweisindex enthält die Linkstruktur der Dokumente untereinander.

Die „IR-typischen“ invertierten Listen für Suchterme finden sich nicht unter den Relationen. Sie sind in Form von Datenbankindexstrukturen (B*-Bäume) für verschiedene Typen von Anfragen vorhanden.

Die wichtigsten durch ihr SQL-*Create Table*-Statement repräsentierten Relationen werden in den folgenden Abschnitten erläutert.

9.8.1.1 Der Objektindex

Die zentrale Relation ist die Relation *OBJECTS*. In ihr werden alle indexierten Dokumente vollständig gespeichert, d.h. die Originaldokumente können verlustfrei aus dem Index rekonstruiert werden.

```
CREATE TABLE INDEX_OBJECTS (
   URI_ID         NUMBER(20)     NOT NULL,
   OBJECT_ID      Varchar2(1000)   NOT NULL,
   ATTRIBUTE_ID    NUMBER(20)     NOT NULL,
   TOKEN_ID        NUMBER(20)     NOT NULL,
   TOKEN_ID_ABS    NUMBER(20)       NOT NULL,
   OBJECT_TYPE    NUMBER(20)     NOT NULL,
   TOKEN        Varchar2(200),
   TOKEN_ORIGINAL  Varchar2(200),
   CONTENT      CLOB,
   ROOT          Varchar2(1),
   IU_ID        NUMBER(20)     NOT NULL,
   WEBSERVICE_ID  Varchar2(20),
   FORM          NUMBER(20),
   PARAM        NUMBER(20),
   TYPE          NUMBER(20)
```

```
---PRIMARY KEY (URI_ID, OBJECT_ID, ATTRIBUTE_ID, TOKEN_ID)
)
```

- **URI_ID**: Eindeutige ID des zugehörigen Dokuments.
- **Object_ID**: Kodierung der Position des Knotens im Dewey-Encoding.
- **Attribute_ID**: Falls das Tupel ein Attribut repräsentiert, ist dies die fortlaufenden Nummer des Attributs, 0 sonst.
- **TOKEN_ID, TOKEN_ID_ABS**: Für Knoten, die aus mehreren Termen bestehen (Textblöcke, Attributwerte), ist dies die Position des Terms im Block vor (TOKEN_ID_ABS) und nach Stopworteliminierung (TOKEN_ID).
- **OBJECT_TYPE**: Gibt den Typ des Knotens als numerische Typ-ID an (Element:1, Text:2, Attribut:3).
- **TOKEN,TOKEN_ORIGINAL,TOKEN_ID**: Die einzelnen Terme des Elements als Originalform(TOKEN) und als Stammform(TOKEN_ORIGINAL). Das Attribut *TOKEN_ID* gibt die zugehörige Position des Terms an.
- **CONTENT**: Originalinhalt des Elements inklusive alle Sonderzeichen.

Die Verwaltung der URLs selbst, ihre Zuordnung zu IDs und ihr Bearbeitungsstatus wird über die Tabelle *URI* verwaltet.

```
CREATE TABLE INDEX_URI (
         URI_ID NUMBER(20) NOT NULL,
         URI  VARCHAR2(3000 byte) NOT NULL
         TITLE VARCHAR2(1000 byte),
         COMPLETE VARCHAR2(1 byte)
         GATE VARCHAR2(1 byte),
         )
```

- **URI_ID**: Eindeutige ID des zugehörigen Dokuments.
- **URI**: URI/URL des Dokuments.
- **TITLE**: Titel des Dokuments.
- **COMPLETE**: Indexierungsstatus des Dokuments.
- **GATE**: Bearbeitungsstatus der Annotationskomponente.

9.8.1.2 Der Verweisindex

Die Relation OJBECT_LINK_STRUCTURE beinhaltet die Linkstruktur der Dokumente untereinander auf Elementebene.

```
CREATE TABLE INDEX_OBJECT_LINK_STRUCTURE (
       SOURCE_URI_ID NUMBER(20) NOT NULL,
       SOURCE_OBJECT_ID  Varchar2(1000)  NOT NULL,
       TARGET_URI_ID NUMBER(20) NOT NULL,
       TARGET_OBJECT_ID Varchar2(1000)  NOT NULL,
       XPATH Varchar2(1)
)
```

Sie beinhaltet die Dokument- sowie Knoten-ID in Form der Dewey-Kodierung für Quell und Zielknoten. Falls das Markierungs-Attribut XPATH ungleich null ist, enthält TARGET_OBJECT_ID die XPath-Repräsentation des Link-Ziels, die, sobald das Zieldokument indexiert wurde, durch die Dewey-ID des Zielknotens ersetzt wird.

Die Relation OJBECT_TRANSITIVE_STRUCTURE enthält die optional bis zu einer spezifizierten Tiefe berechnete transitive Hülle.

```
CREATE TABLE INDEX_OBJECT_TRANSITIVE_STRUCTURE (
       SOURCE_URI_ID NUMBER(20) NOT NULL,
       SOURCE_OBJECT_ID  Varchar2(1000)  NOT NULL,
       TARGET_URI_ID NUMBER(20) NOT NULL,
       TARGET_OBJECT_ID Varchar2(1000)  NOT NULL,
       DIST NUMBER(5)
)
```

Hierbei enthält DIST die Entfernung des Knotens mit der Knoten-ID SOURCE_OBJECT_ID des Dokument mit ID SOURCE_URI_ID zum Knoten TARGET_OBJECT_ID des Dokument mit ID TARGET_URI_ID. Da diese Tabelle aus der Linktabelle OJBECT_LINK_STRUCTURE berechnet wird, enthält sie Ausgangs- und Zielknoten von Links, die in OJBECT_LINK_STRUCTURE enthalten sind.

9.8.1.3 Der Annotationsindex

Der Annotationsindex besteht aus den drei Relationen *ANNOMAP*, *ANNOPAGE* und *ANNOTATIONS*.

- **ANNOMAP** beinhaltet die Abbildung der GATE Annotationsbezeichnungen und IDs (bestehend aus *Major*, *Major_ID*, *Minor*, *Minor_ID*) auf die SphereSearch-Bezeichnung (Type) und die zugehörige ID.

```
CREATE TABLE INDEX_ANNOMAP(
        ID NUMBER(10),
        TYPE VARCHAR2(256) NOT NULL,
        TYPE_ID NUMBER(10),
        MAJOR VARCHAR2(256),
        MAJOR_ID NUMBER(10),
        MINOR VARCHAR2(256),
        MINOR_ID NUMBER(10)
      )
```

- **ANNOPAGE** enthält Statistiken über die Anzahl an Annotation jedes Typs (Anzahl OCC der Annotation ANNO_ID) pro Seite (URI_ID)

```
CREATE TABLE INDEX_ANNOPAGE (
    URI_ID  NUMBER(20) NOT NULL,
    ANNO_ID NUMBER(10) NOT NULL,
    OCC     NUMBER(10) NOT NULL
)
```

- **ANNOTATION** enthält die eigentlichen Annotationen. Hierbei spezifiziert URI_ID das Dokument und OBJECT_ID die ID des mit der Annotation vom Typ ANNO_ID annotierten Knotens. CONTENT enthält immer die Zeichenkettenrepräsentation des Objekts (z.B. „17. November 1981 morgens“), CONTENT_NUM optional zusätzlich seine numerische Repräsentation (z.B. die Kodierung des Datums als Anzahl Tage seit 1900). Die numerische Repräsentation ist insbesondere für die Auswertung von Ähnlichkeits- und Bereichsabfragen (Kap 9.6) wichtig.

```
CREATE TABLE INDEX_ANNOTATIONS (
      URI_ID VARCHAR2(16) NOT NULL,
      OBJECT_ID VARCHAR2(128) NOT NULL,
      ANNO_ID NUMBER(10) NOT NULL,
      CONTENT VARCHAR2(512) NOT NULL
 )
```

9.8.1.4 Der Geo-Server-Index

Der Index der Geo-Servers besteht aus den vier Relationen FEATURE_LOCATION, FEATURE_DISPLAYNAME, CLASSIFICATION und TABLE SCHEME_TERM.

- **FEATURE_LOCATION** weist geographischen Koordinaten eine eindeutige ID zu. Die vier Koordinaten *North_Coordinate* (nördliche Länge), *South_Coordinate* (südliche Länge), *West_Coordinate* (westliche Breite) und *East_Coordinate* (östliche Breite) spannen hierbei eine räumliches Rechteck auf.

```
CREATE TABLE FEATURE_LOCATION (
    FEATURE_ID  NUMBER(8) NOT NULL,
    NORTH_COORDINATE  NUMBER(17, 14) NOT NULL,
    EAST_COORDINATE NUMBER(17, 14) NOT NULL,
    WEST_COORDINATE  NUMBER(17, 14) NOT NULL,
    SOUTH_COORDINATE NUMBER(17, 14) NOT NULL,
   )
```

- **FEATURE_DISPLAYNAME** enthält englischsprachige Bezeichnungen (DISPLAYNAME) zu den in Tabelle **FEATURE_LOCATION** enthaltenen und über *FEATURE_ID* referenzierten Koordinaten.

```
CREATE TABLE FEATURE_DISPLAYNAME (
    FEATURE_ID  NUMBER(8) NOT NULL,
    DISPLAYNAME VARCHAR2(255 byte) NOT NULL,
    SOURCE_FEATURE_IDS  VARCHAR2(127 byte),
  )
```

- **CLASSIFICATION** weist den über *FEATURE_ID* referenzierten Objekten Klassifikations-IDs zu. Eine Klassifikations-ID repräsentiert eine bestimmte Klasse geographischer Objekte, wie beispielsweise Flüsse, Berge oder Städte.

```
CREATE TABLE CLASSIFICATION (
  CLASSIFICATION_ID  NUMBER(10) NOT NULL,
  FEATURE_ID NUMBER(8) NOT NULL,
  CLASSIFICATION_TERM_ID NUMBER(10) NOT NULL,
 )
```

- **TABLE SCHEME_TERM** enthält die englischsprachige Erläuterung (*Term*) der in Tabelle *CLASSIFICATION* verwendeten Klassifikations-IDs.

```
CREATE TABLE SCHEME_TERM (
    SCHEME_TERM_ID NUMBER(8) NOT NULL,
    TERM  VARCHAR2(255 byte) NOT NULL,
    SCHEME_ID NUMBER(2) NOT NULL,
    )
```

10 Experimentelle Evaluation

10.1 Auswahl der Benchmarks

Zur Evaluation der SphereSearch-Suchmaschine wurden gängige Benchmarks in Betracht gezogen. Allerdings war es nicht möglich, einen geeigneten Benchmark für diese Art von Suchmaschine zu finden. Wir untersuchten sowohl XML Benchmarks wie XMach [BR01], XMark [SWK+02] oder INEX [INEX04] als auch klassische Information-Retrieval-Benchmarks wie TREC [TREC] auf ihre Eignung. Die ersten beiden genannten XML-Benchmarks sind zugeschnitten auf XQuery-artige, exakte Anfragen ohne Relevanz-Ranking auf schemabehafteten XML-Daten, wohingegen die aktuellen TREC-Benchmarks keine Struktur berücksichtigen und nur auf Dokumentebene arbeiten. Der INEX-Benchmark ist von den genannten Benchmarks derjenige, der unseren Anforderungen am nächsten kommt. Allerdings basiert der aktuelle INEX-Benchmark auf einer Sammlung von Publikationen aus IEEE-CS-Journalen, die zwar Volltexte mit XML-Tags sind, allerdings nur sehr einfache, generische und semantisch schwache Tags wie $<$ *section* $>$, $<$ *subsection* $>$, $<$ *caption* $>$ beinhalten. Außerdem werden, wie in allen übrigen Benchmarks auch, Links zwischen Dokumenten nicht für die Anfrageauswertung berücksichtigt [1]. Trotzdem führten wir Experimente mit dem INEX-Benchmark durch, um aufzuzeigen, dass das System auch gute Ergebnisse auf solchen (semantisch) schwach strukturierten XML-Dokumenten liefert. Was für eine aufschlussreiche Evaluation des Systems benötigt würde, wäre eine große, strukturell heterogene Datensammlung mit semantisch reichen und facettenreichen Tags. Da keine der untersuchten Optionen in Hinblick auf diese Anforderungen passend erschien, wurden zusätzlich zur etablierten, aber nicht gut passenden INEX-Datensammlung drei neue Datensammlungen zusammen mit passenden Anfragen erstellt.

[1] Web-Algorithmen wie beispielsweise Googles Page-Rank, die die Link-Struktur zur Berechnung des Ergebnis-Rankings ausnutzen, sind nicht gemeint, sondern Anfragen, deren Antworten über mehrere verlinkte Dokumente verteilt sind.

10.2 Die Benchmarks

10.2.1 Die Korpora

10.2.1.1 Wikipedia

Der Wikipedia-Korpus besteht aus Daten des Wikipedia-Projekts [WIKI], einer freien Enzyklopädie, die gemeinschaftlich von Nutzern des Internet gepflegt wird.

	Wikipedia	Wikipedia++
Korpusstruktur		
Dokumente	439,123	494,730
Links	13,786,112,	15,190,224
Links pro Dokument	31.4	30.7
Datengröße (File-System)	3.1 GB	3.4 GB
Dokumentstruktur		
XML-Elementknoten	24,272,144	28,697,928
Elementknoten pro Dokument	55	58
XML-Textknoten	24,327,084	27,196,960
XML-Attributknoten	24,192,095	29,185,399
Annotationen		
Gesamt	12,068,348	
Person	3,660,345	
Ort	2,832,172	
Datum	3,378,362	
Geld	230,666	
Organisation	1,853,075	

Tabelle 10.1: Statistische Daten des Wikipedia(++)-Korpus

Die englischsprachige Wikipedia-Version bestand zum Zeitpunkt der Experimente (Anfang 2005) aus über 400.000 Artikeln. Obwohl die Artikel oft gut durch Überschriften, Aufzählungen und Tabellen strukturiert sind, folgen sie keinem festen Schema. Des Weiteren weist der Wikipedia-Korpus keine feste Dokumentenhierarchie auf. Alle Dokumente sind allerdings stark miteinander verlinkt. Für die Experimente installierten wir eine lokale Kopie des Wikipedia-Web-Servers und importierten die Seiten mit dem SphereSearch-Crawler, der diese nach XML transformierte und im internen Index speicherte. Statistische Daten zum Wikipedia-Korpus sind in Tabelle 10.1 angegeben.

Anschließend wurden dem Korpusindex mit der Annotationskomponte verschiedene Annotationen hinzugefügt. Abbildung 10.1 zeigt ein Fragment einer Wikipedia-HTML-Seite und der annotierten XML-Version, die im internen Index gespeichert wird.

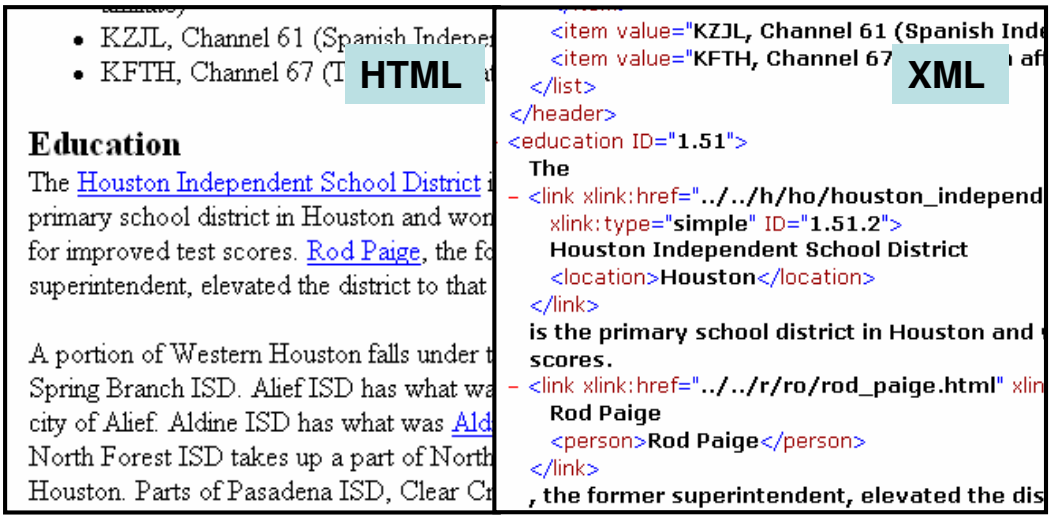


Abbildung 10.1: Wikipedia: HTML und annotiertes XML

10.2.1.2 Wikipedia++

Der *Wikipedia++ Korpus* ist eine Erweiterung des Wikipedia Korpus. Er ist eine Kombination aus Wikipedia und Daten der *Internet Movie Database (IMDB)* [IMDB]. Die IMDB enthält Informationen über praktisch alle Kinofilme und Schauspieler. Wir generierten aus den ursprünglichen IMDB-Daten XML-Dateien für jeden Film und jeden Schauspieler.

Abbildung 10.2: Beispiel-XML-Datei der *Internet Movie Database*

Abbildung 10.2 zeigt zwei Beispieldokumente der IMDB. Die Dokumente enthalten vielfältige semantisch reiche Tags zu den einzelnen Filmen und Schauspielern wie z.B. *Plot, Production Country, Vitae* sowie eine Vielzahl an Links zwischen den einzelnen Dokumenten. Um einen zusammenhängenden Korpus zu erhalten, generierten wir automatisiert Links in den Wikipedia-Korpus, falls die jeweiligen Filme oder Schauspieler dort vorhanden waren. Der nun erweiterte Korpus *Wikipedia++* enthält nun Links zwischen IMDB und Wikipedia für Filme und Schauspieler, die in beiden Korpora vorkommen. Statistische Daten zum Wikipedia++-Korpus sind ebenfalls in Tabelle 10.1 angegeben.

10.2.1.3 DBLP++

Der *DBLP++ Korpus* basiert auf dem DBLP-Projekt [DBLP], das bibliographische Informationen der wichtigsten Informatikjournale und Tagungsbände sammelt, aufbereitet und zur Verfügung stellt. Zum Zeitpunkt der Experimente umfasste es mehr als 480.000 Publikationen und enthielt mehrere Tausend Links zu Homepages von Wissenschaftlern, deren Veröffentlichungen enthalten sind. Die DBLP-Daten sind als eine einzige große XML-Datei (zusätzlich zur online verfügbaren HTML-Version) verfügbar. Aus dieser Datei generierten wir einzelne XML-Dokumente für jeden Autor, jeden Tagungsband, jeden Artikel usw. und verbanden diese mit den entsprechenden XLinks untereinander.

DBLP++	
Dokumente	970537
Links	3198095
Links pro Dokument	24
Dokumentstruktur	
XML-Elementknoten	9,319,820
Elementknoten pro Dokument	9,6
XML-Textknoten	7,849,264
XML-Attributknoten	8,174,065

Tabelle 10.2: Statistik für DBLP++-Korpus

Desweiteren indexierten wir zusätzlich zu diesen XML-Dateien die (HTML-)Homepages von über 30.000 Wissenschaftlern, zu denen Links in der Web-Version von DBLP enthalten sind. Hierzu extrahierten wir zunächst die Links auf die Wissenschaftler-Homepages von den DBLP-Web-Seiten und fügten sie als XLinks in die entsprechenden DBLP-XML-Dokumente ein. Diese Kombination führt, vergleichbar zum *Wikipedia++*-Korpus, zu einer semantisch reichen und heterogenen Datensammlung, sowohl in Bezug auf den Inhalt als auch die Struktur (vgl. Tabelle 10.2). Abbildung 10.3 zeigt zwei verlinkte DBLP-Dokumente.

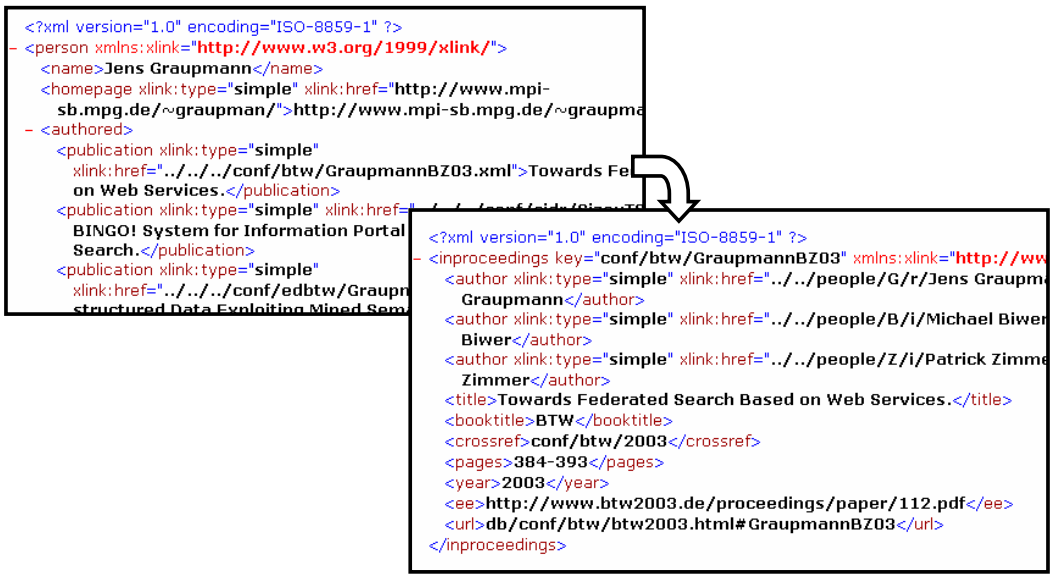


Abbildung 10.3: DBLP-Dokumente

10.2.1.4 INEX

Der *INEX*-Korpus [INEX04] umfasst 12.103 wissenschaftliche Artikel aus Veröffentlichungen der *IEEE Computer Society*. Tabelle 10.3 enthält statistische Daten zum INEX-Korpus. Zusätzlich gehören zum INEX-Benchmark eine Menge von Anfragen (die *Topics*), sowohl mit strukturellen Einschränkungen, als auch reine Schlüsselwortanfragen, zusammen mit manuell ausgewerteten Ergebnislisten für jede der Anfragen. Abbildung 10.4 zeigt ein INEX-Dokument sowie ein Anfrage (*Topic*). Das *title*-Element enthält hierbei die Anfrage im INEX-spezifischen NEXI-Format, die von einer Suchmaschine ausgewertet werden soll. Die Elemente *desciption* und *narrative* enthalten hierbei zusätzlich ein kurze bzw. eine ausführliche natürlichsprachliche Umschreibung der Suchanfrage, die allerdings nur für die spätere manuelle Relevanzbewertung der Ergebnis genutzt werden darf. Details zu NEXI-Anfragen und ihrer Verarbeitung in SphereSearch sind in Kapitel 10.5.2 zu finden. Desweiteren werden Auswertungs-Tools zur Verfügung gestellt, die eine automatische Auswertung der Effektivität einer XML-Suchmaschine für die gegebenen Daten ermöglichen. Wir benutzten die 46 Anfragen der Auswertungsrunde des Jahres 2004, die durch einen entsprechenden Adapter automatisch in SphereSearch-Anfragen übersetzt wurden.

10.2.2 Die Anfragen

Da abgesehen von den Anfragen des INEX-Benchmark keine Anfragen zur Verfügung standen, wurden Kollegen gebeten, Anfragen für die übrigen Korpora (Wikipedia,

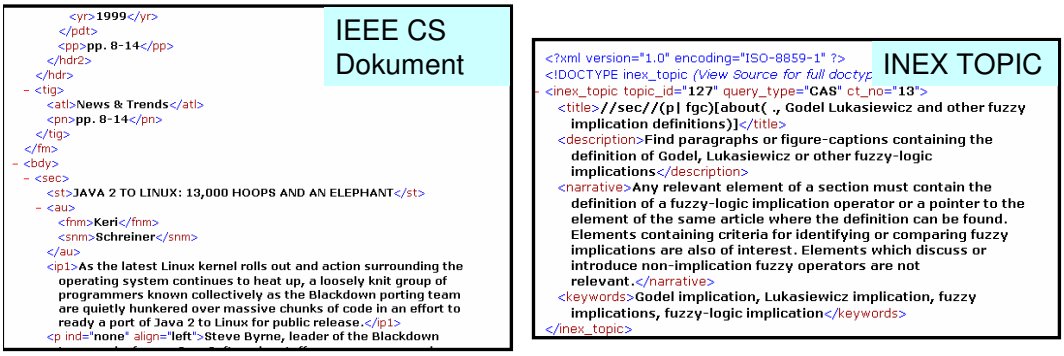


Abbildung 10.4: INEX-Dokument und -Anfrage

INEX	
Dokumente	12272
Links	9902
Links pro Dokument	0.8
Dokumentstruktur	
XML-Elementknoten	8,283,874
Elementknoten pro Dokument	675
XML-Textknoten	8,267,155
XML-Attributknoten	3,858,653

Tabelle 10.3: Statistische Daten zum INEX-Korpus

Wikipdia++, DBLP++) zu stellen. Als vage Anforderung an die Anfragen stellten wir, dass Resultate auf der einen Seite nicht so einfach zu finden sein sollten, dass einfache Schlüsselwortsuche offensichtlich direkt ein gutes Resultat finden würde, und auf der anderen Seite nicht zu schwierig sein sollten, dass nicht einmal mit Hilfe einer hochentwickelten XML-Suchmaschine und Wissen über die Struktur der Antworten eine korrekte Antwort gefunden werden könnte. Wir wählten 50 der eingereichten Anfragen aus, die verschiedene Sprachfeatures der Anfragesprache nutzten. Um sicherzustellen, dass die Anfragen zumindest einer gewissen Systematik folgten, wurden die Anfragesteller angewiesen, die Daten so zu sehen, als folgten sie einem „latenten Schema“. Für die verschiedenen Korpora wurden entsprechend angepasste Schemata angegeben. Sie sind zusammen mit allen für die Experimente genutzten Anfragen in Anhang A zu finden. Für den Wikipedia-Korpus sieht das latente Schema wie folgt aus:

`Wikipedia (Person, Date/Time, Location, Event, Keywords, Description)`

Für jede Anfragegruppe sollte hierbei ein Tupel mit beliebig vielen belegten Einträgen spezifiziert werden. Zusätzlich konnten Join-Bedingungen zwischen mehreren Gruppen (bzw. Tupeln) markiert werden, wobei jede Gruppe durch eine Zeile repräsentiert wird.

Die Orientierung an diesem Schema war keine Einschränkung, sondern nur eine Hilfe, da praktisch alle Anfragen mit Hilfe dieses Schemas formuliert werden konnten. Tabelle 10.4 zeigt das Beispiel einer Anfrage, wie sie unter Verwendung des Schemas gestellt wurde.

Person	Location	Date	Organisation	Event	Keywords	Description
	Germany	1800-1850			composer	Deutsche Komponisten der 1.Hälfte 18. Jhd.

Tabelle 10.4: Beispielanfrage mit latentem Schema

Anschließend wurden die Anfragen in verschiedene Sprachlevels, basierend auf den genutzten Features, kategorisiert. Die insgesamt vier Sprachlevels sind in Abbildung 10.5 dargestellt und wie folgt spezifiziert:

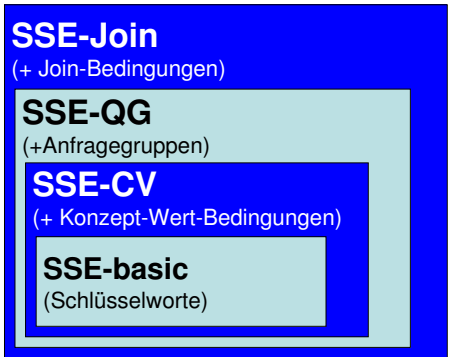


Abbildung 10.5: Sprachlevel

- *SSE-basic*: Die „Basisversion“, eingeschränkt auf Schlüsselwortbedingen, ohne Anfragegruppen (bzw. eingeschränkt auf eine einzige Gruppe) und Konzept-Wert-Bedingungen (Konzeptbewusstsein).
- *SSE-CV*: Die Basisversion mit zusätzlicher Unterstützung für Konzept-Wert Bedingungen.
- *SSE-QG*: Die CV-Version mit zusätzlicher Unterstützung für Anfragegruppen (volles Kontextbewusstsein).
- *SSE-Join*: Die vollständige Version unter Ausnutzung aller Feature inkl. Join-Bedingungen.

Auf allen Sprachlevels konnte der Ähnlichkeitsoperator $\sim$ genutzt werden.

Um dieselben Anfragen für alle Komplexitätslevels nutzen zu können, emulierten wir Anfragen höherer Levels auf niedrigeren Levels, indem wir ausdruckstärkere Bedingungen (die auf dem niedrigerem Level nicht zur Verfügung standen) durch entsprechende Schlüsselwortanfragen ersetzten. Anfragen niedriger Levels wurde ohne Modifikation in höhere Levels übernommen. Somit basiert ein Gewinn in Hinblick auf Präzision von einem Level zu einem höheren Level ausschließlich auf dem Teil aller Anfragen, die ursprünglich mit den Mitteln dieses (höheren) Levels formuliert wurden.

Wir verglichen SphereSearch unter Nutzung der obigen Sprachlevels mit verschiedenen Gegnern:

- *SSE-KW*: Eine simplifizierte Version von SphereSearch ohne Sphären-Scoring. SphereSearch entspricht in diesem Modus einer einfachen schlüsselwortbasierten Suchmaschine, die Okapi-BM25 ([RW94]) auf Dokumentebene zur Score-Berechnung nutzt.
- *GoogleWiki*: Google, eingeschränkt auf die Domäne *wikipedia.org*. Experimente haben verifiziert, dass Google so gut wie alle Wikipedia-Seiten indexiert hat.
- *Google∼Wiki*: Google, eingeschränkt auf wikipedia.org, mit Benutzung von Google's ∼-Operator zur Anfrageexpansion
- *GoogleWeb*: Google, nicht eingeschränkt, d.h. auf dem gesamten Web (incl. Wikipedia).
- *Google∼Web*: Google, nicht eingeschränkt, auf dem gesamten Web mit Anfrageexpansion (Google-∼-Operator)

10.3 Experimentalaufbau

Die für die Experimente gewählten Parameter und der Experimentalaufbau sind in Tabelle 10.5 angegeben.

Als Messwerte interessierten uns insbesondere der Makrodurchschnitt der Präzision unter den Top-10-Ergebnissen (macro-averaged precision@10, kurz MP@10), d.h. die Präzision der Top-10-Treffer, gemittelt über alle k Anfragen $Q = \{q_1, .., q_k\}$, wobei die Relevanz von Ergebnissen manuell überprüft wurde (Formel 10.1). Die Präzision unter den Top-10 für eine Anfrage q (Precision@10, kurz $P@10(q)$, Formel 10.2) ist definiert als der Anteil der relevanten Dokumente unter den ersten 10 erhaltenen Treffern ($r_{10}(q)$).

$$MP@10(Q) = \frac{\sum_{i=1}^{k} P@10(q_i)}{k} \tag{10.1}$$

Hardware & Software	
System	Sun V40z
Betriebssystem	MS Windows 2003 Server
CPU	4 * AMD Opteron 2.4 GHz
Festplatten	6x146GB (SCSI,RAID-5)
Hauptspeicher	16 GB
Application-Server	Tomcat 4.0.6
Datenbanksystem	Oracle 10g
SphereSearch-Parameter	
Link-Gewicht λ	1
Sphärengröße D	6
Dämpfungsfaktor α	0.5
Kompaktheitsgewicht β	0.5

Tabelle 10.5: Experimentalaufbau

$$P@10(q_i) = \frac{r_{10}(q_i)}{10} \tag{10.2}$$

10.4 Ergebnisse für Wikipedia(++) und DBLP++

Zunächst werden in diesem Abschnitt die aggregierten Ergebnisse präsentiert, bevor beispielhaft auf einzelne Anfragen der verschiedenen Anfrage-Levels eingegangen wird um zu veranschaulichen, worauf die Gewinne gegenüber einfacheren Anfrage und Anfrageauswertungen zurückzuführen sind.

Abbildung 10.6 zeigt den Makrodurchschnitt der Präzision unter den Top-10-Treffern für den Wikipedia-Korpus. Hierfür wurde SphereSearch auf allen Sprachlevels mit allen oben genanten Gegnern verglichen. Als erstes ist festzustellen, dass SSE-KW, die simplifizierte dokumentbasierte Version von SphereSearch, auf diesem Korpus vergleichbar mit Google ist. Als wichtigstes Ergebnis zeigt die Auswertung, dass durch Benutzung weiterer Sprach-Features die durchschnittliche Präzision fast auf den doppelten Wert der Präzision von Google bzw SSE-KW erhöht werden kann. Dies zeigt deutlich, dass selbst für eher schwach strukturierte Dokumente, wie die des Wikipedia-Korpus, die strukturorientierte Anfragesprache von SphereSearch hilfreich ist.

Für stärker strukturierte Dokumente wie die der *Wikipedia++*- und *DBLP++*-Korpora fällt der Zuwachs hinsichtlich Präzision, der durch Einsatz von Sprachkonstrukten höherer Levels erreicht werden kann, noch deutlicher aus. Abbildung 10.7 zeigt die Auswertung für diese Korpora. Da die Anfragen auf diesen Korpora nicht mehr sinnvoll mit den Ergebnissen von Google verglichen werden können, wurde hierauf verzichtet. Da allerdings SSE-KW zu Google vergleichbare Ergebnisse liefert, kann diese Variante hier als Vergleichsgröße (*Baseline*) dienen. Abbildung 10.8

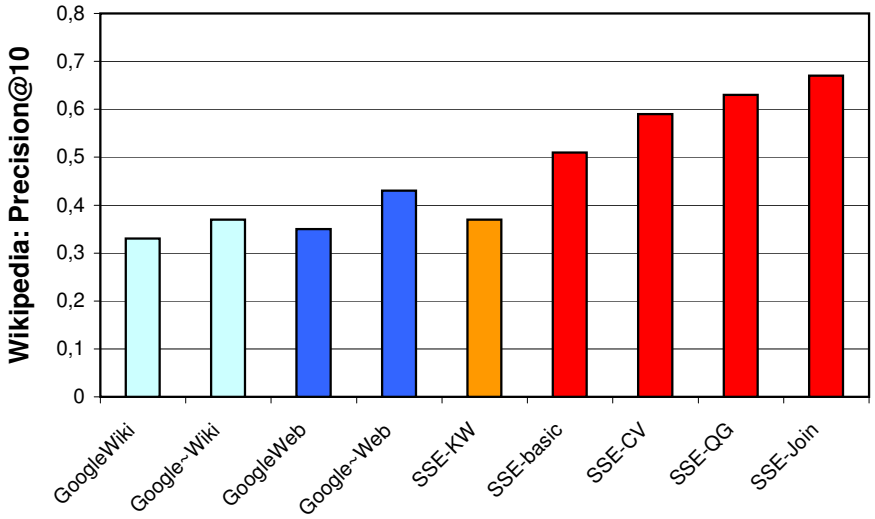


Abbildung 10.6: Aggregierte Ergebnisse für Wikipedia

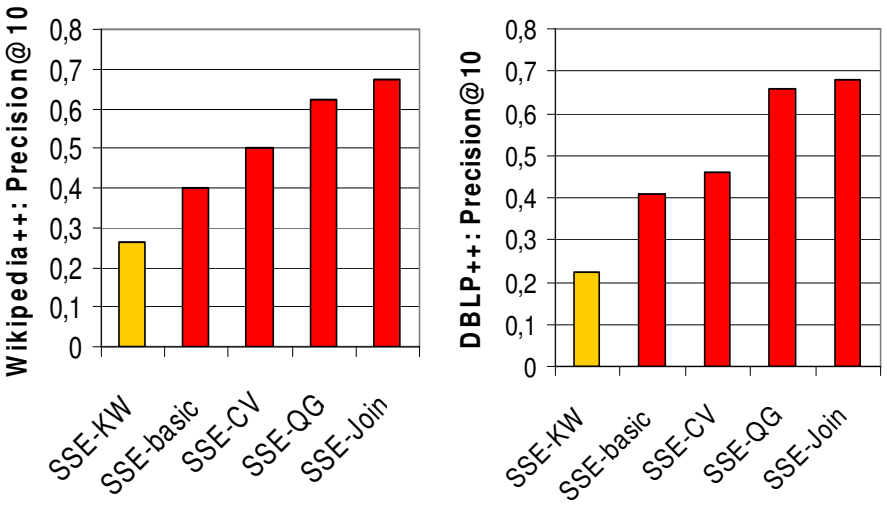


Abbildung 10.7: Aggregierte Ergebnisse für Wikipedia++ und DBLP++

zeigt die durchschnittlichen Laufzeiten für die Anfragen der verschiedenen Levels. Hierbei ist farblich hervorgehoben, welchen Anteil jeweils die Datenbank (DB) und welchen Anteil die SphereSearch-Engine (SSE) zur Gesamtlaufzeit beiträgt. Mit steigender Komplexität steigt auch die Laufzeit der Anfragen. Außerdem nimmt prozentual auch deutlich der Rechenaufwand der SphereSearch-Engine zu. Beträgt er bei Schlüsselwortanfragen nur 30%, so wird bei komplexen Anfragen mit Join-Operationen fast 80% der Laufzeit durch die SphereSearch-Engine verursacht. Im Hinblick auf die großen Datenkorpora und den Prototyp-Status der Suchmaschine befinden sich die absoluten Laufzeiten in einem akzeptablen Rahmen.

In den folgenden Abschnitten wird beispielhaft auf einzelne Anfragen der ver-

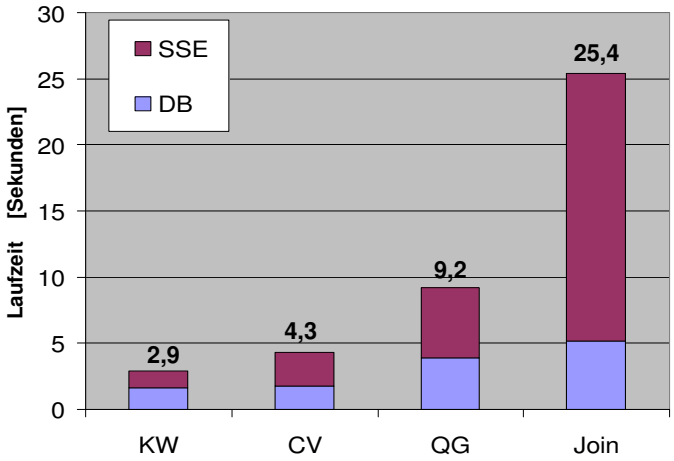


Abbildung 10.8: Laufzeiten

schiedenen Sprach-Levels eingegangen.

10.4.1 Schlüsselwortanfragen

Sogar für einfache Schlüsselwortanfragen, die keine spezifischen SphereSearch-Sprachfeatures benutzen, liefert SphereSearch bessere Resultate als Google und SSE-KW. Ein Grund hierfür liegt in Googles PageRank-Berechnung, die Hub-Seiten (in Wikipedia insbesondere Listen, beispielsweise mit den Name aller Personen oder Ereignisse nach Jahren sortiert), die typischerweise kaum spezifische Informationen enthalten, hohe Scores zuweist.

Eine einfache Anfrage des Testkorpus ist die Anfrage `(Inventor,World Wide Web)`, die nach den Erfindern des Word Wide Web sucht. Sowohl Google als auch SSE-KW geben fast dieselben, auf den ersten Blick guten Ergebnisse zurück. Allerdings enthalten die Top-10 Ergebnisse nur die prominenteste Person, die als „Erfinder“ des WWW angesehen wird: Tim Berners-Lee.

SphereSearch hingegen gibt für diese Anfrage auch Ergebnisse zurück, die Robert Cailliau, den Miterfinder und früheren Arbeitskollegen von Berners-Lee, nennen. Diese Seiten erhalten bei SphereSearch insbesondere durch das häufige Vorkommen der Begriffe `(Inventor,World Wide Web)` auf durch ein- und ausgehende Links verbundenen Seiten hohe Sphären-Scores für die entsprechenden Dokumentknoten.

10.4.2 Anfragen mit Konzept-Wert-Bedingungen

Bestimmte Typen von Anfragen profitieren besonders von Konzept-Wert-Bedingungen. Insbesondere sind dies Anfragen, die mehrdeutige Teilbedingungen enthalten. Wenn zum Beispiel der Benutzer versucht den Vornamen der amerikanischen Politikerin (Condoleezza) Rice mit der Anfrage `(American, politician, Rice)`

in Erfahrung zu bringen, ist nicht ein einziges korrektes Ergebnis unter den Top-10-Ergebnissen. Wird die Anfrage hingegen als `(person=rice,politician)` gestellt, erhöht dies die Präzision@10 von 0 auf 0,6 (in den übrigen 4 Ergebnissen ist die Person 'Rice' keine Politikerin).

Andere Anfragetypen, die signifikant von Konzept-Wert-Bedingungen profitieren, sind Anfragen die sich auf Annotationen beziehen und eine numerische Auswertung ermöglichen bzw. erfordern. Ein Beispiel hierfür ist eine Anfrage nach französischen Mathematikern des neunzehnten Jahrhunderts, die kaum mit einfachen Schlüsselworten ausgedrückt werden kann. Als Konzept-Wert-Anfrage `(french mathematician, date=1800-1899)` kann diese allerdings problemlos gestellt und mit hoher Präzision durch SphereSearch ausgewertet werden.

10.4.3 Anfragegruppen

Da Anfragegruppen genutzt werden können um Schlüsselworte, die sich auf dasselbe Element beziehen, zu gruppieren, führt dieses zusätzliche strukturelle Element meist zu höherer Präzision. Falls Ergebnisse auch verteilt auf mehreren miteinander verbundenen Seiten gefunden werden können, werden hierdurch zusätzliche Treffer gefunden und somit der *Recall*, das Verhältnis von gefundenen zu gesamt vorhandenen Treffern für die Anfrage, erhöht. Eine typische Anfrage, deren Resultatsgüte signifikant von der Nutzung von Anfragegruppen profitiert, ist die Schlüsselwortanfrage `(California, governor, movie)`, die nach Filmen, bei denen der kalifornische Gouverneur mitgespielt hat, sucht. Diese Anfrage liefert eine Präzision@10 von 0, da ausschließlich Filme, die entweder in Kalifornien spielen oder in deren Handlung ein Gouverneur eine Rolle spieltm unter den Ergebnissen auftauchen. Die mit Hilfe von Anfragegruppen formulierte Anfrage `G(California, governor) M(movie)` hingegen erhöht die Präzision@10 auf 0.4.

10.4.4 Anfragen mit Joins

Eine typische Anfrage, die die Aussagekraft von Join-Bedingungen ausnutzen kann, ist die Anfrage nach Filmen, bei denen der Ehemann der Sängerin Madonna Regie geführt hat. Ohne Kenntnis des Namens von Madonnas Mann (Guy Ritchie) ist es fast unmöglich eine Schlüsselwortanfrage zu stellen, die korrekte Ergebnisse zurückgibt. Dieselbe Anfrage als SphereSearch-Anfrage
`A(Madonna,husband) B(director) A.person=B.director`
mit einer Join-Bedingung gibt 4 relevante Treffer unter den Top-10 zurück. Die Ergebnisse haben meist die Struktur, dass sie jeweils aus einem Paar von Elementen bestehen, wobei ein Element die Information enthält, dass Guy Ritchie der Ehemann von Madonna ist, und das andere Element einen Kinofilm repräsentiert, in dem Guy Ritchie Regie geführt hat.

10.5 Ergebnisse für INEX

10.5.1 INEX-Evaluation

Die 46 Anfragen des INEX 2004 Benchmarks [2] setzten sich aus 25 Schlüsselwortanfragen (*Content-only Topics*(CO)) und 21 Anfragen mit zusätzlichen strukturellen Einschränkungen (*Content-and-Structure Topics*(CAS)) zusammen. Zur Spezifizierung der Anfrage und insbesondere der strukturellen Bedingungen wird die XPath-ähnliche Anfragesprache *NEXI (Narrowed Extdended XPath I)* benutzt [TS04].

Die strukturellen Bedingungen bei CAS-Anfragen werden bei INEX im *VCAS*-Modus (*Vague content and structure*) ausgewertet. Dies bedeutet, dass die in der Anfrage spezifizierten Strukturbedingungen nicht notwendigerweise von einem Dokument erfüllt werden müssen, damit dieses als Treffer angesehen werden kann. Vielmehr werden die strukturellen Bedingungen als zusätzliche Hilfe betrachtet. Eine *SCAS*-Auswertung (*strict content and structure*), die eine strikte Einhaltung der Strukturbedingungen fordert, ist, im Gegensatz zu den Vorjahren, im INEX-Benchmark des Jahres 2004 nicht mehr vorgesehen.

Die Relevanzbewertung beruht auf der manuellen Auswertung der Ergebnisse aller teilnehmenden Suchmaschinen. Nachdem jeder Teilnehmer Ergebnislisten mit maximal 1500 Treffern pro Anfrage in einem genau spezifizierten Format eingereicht hat, werden die Top-100-Ergebnisse aller Suchmaschinen pro Topic zusammengefasst und müssen von den Teilnehmern auf Relevanz hin untersucht werden. Die Ergebnisse werden auf Elementebene mit einem XPath-Ausdruck spezifiziert, der den an diesem Element beginnenden Dokumentteilbaum als potentiellen Treffer markiert. Ein solcher Teilbaum wird im Folgenden als Dokumentfragment bezeichnet.

Die Bewertung der Relevanz eines Treffers setzt sich hierbei aus zwei Teilbewertungen zusammen:

- **Specificity** (Spezifität) beurteilt, inwieweit das betrachtete Dokumentfragment auf die Anfrage fokussiert ist. Ein Dokumentfragment, das beispielsweise viele Themen diskutiert, unter denen sich auch das Thema der Anfrage befindet, kann somit nicht maximal spezifisch sein.

- **Exhaustivness** (Vollständigkeit) hingegen beurteilt, inwieweit ein Element das Thema inhaltlich abdeckt. Falls beispielsweise das auf das betrachtende Dokumentfragment folgende Fragment innerhalb des selben Dokument einen anderen für die Anfrage relevanten Aspekt diskutiert, kann das betrachtete Element nicht maximal vollständig sein.

[2] Der Benchmark findet jährlich statt. Hierbei findet eine stetige Weiterentwicklung des Benchmarks im Hinblick auf die Kollektion, die Anfragen und die Evaluation statt. Somit sind die Ergebnisse verschiedener Jahre nicht miteinander vergleichbar.

Für jedes Maß können die Werte 0 bis 3 angegeben werden. Ein Wert von 0 gibt an, dass das betrachtete Dokumentfragment nicht spezifisch bzw. vollständig ist, d.h. dass es selbst im weiteren Sinne keinen Treffer für die Anfrage darstellt. Ein Wert von 3 hingegen bewertet ein Element als maximal spezifisch bzw. vollständig.

Es gibt mehrere Bewertungsmetriken, umgesetzt durch Quantisierungsfunktionen, nach denen beim INEX-Benchmark die einzelnen Ergebnisse der Teilnehmer, basierend auf den Relevanzbewertungen, beurteilt werden können. Die Quantisierungsfunktion bildet das zweigeteilte Relevanzmaß von *Specificity* und *Exhaustivness* auf einen Wert zwischen 0 und 1 ab. Im Folgenden wird ein Wertepaar, bestehend aus *Specificity* und *Exhaustivness*, als (Specificity/Exhaustivness), z.B. (3/2), angegeben.

Die beiden wichtigsten Metriken *Strict Quantisation* und *Generalized Quantisation* sind durch die beiden folgenden Quantisierungsfunktionen umgesetzt:

- **Strict Quantisation:** Die strikte Quantisierung berücksichtigt nur Treffer, die als maximal spezifisch und maximal vollständig (also mit einem Wert von 3 für Specificity und Exhaustivness) bewertet wurden.

$$q_{strict}(spec, exhaust) = \begin{cases} 1 & \text{Falls } (3/3) \\ 0 & \text{sonst} \end{cases}$$

- **Generalized Quantisation:** Die verallgemeinerte Quantisierung weist auch Treffern, die nicht maximal spezifisch und vollständig sind, einen Wert größer 0 zu. Maximal spezifische und vollständige Treffer werden mit 1 bewertet.

$$q_{strict}(spec, exhaust) = \begin{cases} 1 & \text{Falls } 3/3 \\ 0.75 & \text{Falls } 2/3, 3/2, 3/1 \\ 0.5 & \text{Falls } 1/3, 2/2, 2/1 \\ 0.25 & \text{Falls } 1/1, 1/2 \\ 0 & \text{Falls } 0/0 \end{cases}$$

Auf Basis der durch die Quantisierungsfunktion berechneten Werte werden Recall/-Precision-Kurven berechnet. Abbildung 10.9 zeigt eine solche Kurve, die die Präzision abhängig vom Recall darstellt. Die Präzision wird auf Basis der *Precall*-Methode an Standard-Recall-Punkten, wie beschrieben in [RBJ89], berechnet. Das Ergebnis eines Benchmark-Durchlaufs mit allen Topics wird berechnet als durchschnittliche mittlere Präzision (*Mean Average Precision (MAP)*), gemittelt über alle Anfragen. Weitere Details sind in [GKFL03] zu finden.

10.5.2 NEXI-nach-SSL-Konvertierung

Um die Anfragen des INEX-Benchmarks auszuwerten, müssen zunächst die Anfragen im NEXI-Format nach SSL übersetzt werden.

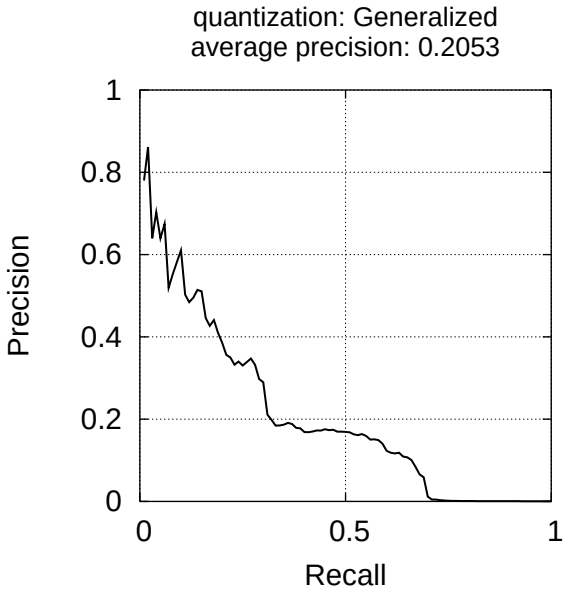


Abbildung 10.9: Precision/Recall-Graph eines INEX-Topics.

Die Sprache *NEXI (narrowed extended XPath I)* ist, wie der Name schon impliziert, eine auf wenige syntaktische Mittel eingeschränkte XPath-Variante, die um eine IR-typische *about*-Funktion ergänzt wurde. Eine NEXI-Anfrage sieht vereinfacht aus wie folgt (für eine vollständige Darstellung der Sprache siehe [TS04]):

$$//path_1[abouts_1]//...//path_n[abouts_n]$$

Jedes *abouts* ist hierbei eine Boole'sche Kombination von *about*-Funktionen der Form *about(path, cont)*. Hierbei gibt *path* einen relativen Pfad zum aktuellen Kontextpfad an, auf den sich die Inhaltsbedingung *cont* bezieht. Der Kontextpfad ergibt sich aus den Pfadbedingungen $path_1...path_i$ (Dies ist analog zu XPath [XPATH02]).

Die folgende Anfrage soll mit *sec*-Tags beginnende Dokumentfragmente über *virtual reality* zurückliefern, die 1998 erschienen sind:

$$//article[about(.//fm//yr, 1998)]//sec[about(.//p, "virtual\ reality")]$$

Da alle Angaben entsprechend der *VCAS*-Auswertungsmethodik nicht exakt ausgewertet werden müssen, sondern im weiteren Sinne Hinweise darstellen, müssen weder die angegebenen Tags noch ihre Abfolge in einem Ergebnisdokument zutreffen.

Auf das SphereSearch-Suchparadigma übertragen heißt dies, dass die entsprechenden Elemente in einer gemeinsamen Sphäre vorkommen sollen. Somit wird die obige Anfrage in die folgende SphereSearch-Anfrage übersetzt.

$$A(\sim article, \sim fm, \sim yr, \sim 1998, \sim sec, \sim p, \sim \ virtual\ reality)$$

Die Semantik des $\sim$-Operators ist allerdings bei der Ausführung von INEX-Anfragen leicht geändert. Bezogen auf Tags expandiert der $\sim$-Operator die angegebenen Tags um äquivalente Tags. Welche Tags äquivalent sind, ist in der INEX-Spezifikation angegeben. Da die Standardauswertung für SSL-Anfragen immer konjunktiv ist, Tags aber als Hinweis, also disjunktiv im Kontext der Anfrage zu interpretieren sind, werden Tags immer auch um ein *leeres Tag* expandiert. Da dieses *leere Tag* immer erfüllbar ist, symbolisiert dies die Optionalität der Bedingung.

Im Falle der Inhaltsbedingungen, die oft Phrasen, wie in dem folgenden Beispiel $about(.,'intelligent\ transportation\ system')$, sind, bewirkt der $\sim$-Operator eine Expansion um Teilphrasen. So werden auch Treffer mit „*transportation system*“ oder „*intelligent transportation*“ zurückgeliefert.

Die Expansion der obigen Anfrage sieht dann aus wie folgt. Hierbei werden die Terme, die der Anfrage durch den $\sim$-Operator hinzugefügt wurden, in Klammern dargestellt.

$$A(article(' '), fm(' '), yr(' '), 1998, sec(ss1, ss2, ss3,' '), p(ilrj, ip1, ip2, ...,' '),\\ virtual\ reality(virtual, reality))$$

Im Gegensatz zu den *CAS*-Anfragen können *CO*-Anfragen, da diese nur aus Schlüsselworten bestehen, einfach übernommen werden. Bei Phrasen wird allerdings auch hier eine Expansion wie bei *CAS*-Anfragen durchgeführt.

10.5.3 Ergebnisse

Für den aus 46 Anfragen zusammen mit ihren Auswertungen bestehenden INEX-Benchmark des Jahres 2004 erreichte SphereSearch eine mittlere durchschnittliche Präzision (MAP) von 0.060 für Schlüsselwortanfragen (CO) und 0.055 für Anfragen mit zusätzlichen strukturellen Einschränkungen (CAS). Dies ist leicht besser als die XXL-Suchmaschine [TW02] und hätte für SphereSearch zu einem Platz unter den Top-30 aller 70 Systeme, die an dem Benchmark 2004 teilgenommen haben, geführt. Es bleibt festzustellen, dass SphereSearch im Gegensatz zu den am besten abschneidenden Systemen nicht nur nicht für den INEX-Benchmark optimiert wurde, sondern auch dass die strukturellen Bedingungen der meisten Anfragen nicht einfach in die SphereSearch-Anfragesprache transformiert werden konnten, da die entsprechenden Sprachkonstrukte für Pfade in SphereSearch nicht zur Verfügung stehen. In Anbetracht dieser eigentlich schlechten Eignung der SphereSearch-Suchmaschine für den INEX-Benchmark überzeugt das Ergebnis durchaus. Im Durchschnitt dauerte die Verarbeitung einer INEX-Anfrage 1-2 Sekunden, was deutlich schneller ist, als die schnellste veröffentlichte Laufzeit von INEX Teilnehmern, die im Schnitt 13 Sekunden pro Anfrage betrug.

11 Fazit

In dieser Arbeit wurde die SphereSearch-Suchmaschine, die einheitliches Retrieval mit Ranking auf heterogenen semi- und unstrukturierten Daten ermöglicht, vorgestellt. Die Arbeit untersucht relevante Aspekte der Verarbeitung von Dokumenten, ihrer internen Repräsentation sowie einer entsprechenden Anfragesprache und ihre Auswertung.

Die neuartige Anfragesprache von SphereSearch ist deutlich einfacher als existierende XML-Anfragesprachen, aber erheblich ausdrucksstärker als Anfragesprachen herkömmlicher Web-Suchmaschinen. Sie vereinigt die Eigenschaften Konzeptbewusstein, Kontextbewusstsein und Abstraktionsbewusstsein, die eine Suchmaschine der nächsten Generation aufweisen muss, um der stetig zunehmenden Informationsflut heterogener Daten Herr zu werden. Für XML-Daten sind solche Eigenschaften vereinzelt zwar bereits verfügbar, allerdings ist eine solche ganzheitliche Lösung für heterogene Datensammlungen, auf denen weder XML-Anfragesprachen wie XQuery noch XML-IR-Ansätze anwendbar sind, innovativ und neu.

Die interne Sichtweise auf Daten in SphereSearch unterscheidet sich ebenfalls grundlegend von früheren Ansätzen, weil nicht baumstrukturierte Dokumente einzeln betrachtet werden, sondern die Gesamtheit aller Dokumente als *ein* Graph. Somit müssen Ergebnisse nicht auf einzelne Dokumente oder Dokumentfragmente beschränkt sein, sondern sind beliebige Teilgraphen, die sich über Dokumentgrenzen einzelner Dokumente hinweg erstrecken. Diese Sichtweise erfordert auch neue Ansätze zur Gütebewertung einzelner Treffer, die spezifische Eigenschaften der Graphstruktur eines Ergebnisses berücksichtigen. Zu diesem Zweck wurden die neuen Konzepte Sphären und Kompaktheit, die das Umfeld von Teilergebnissen und die Gesamtgröße eines Ergebnisse in Hinblick auf seinen zugrunde liegenden Teilgraphen in Betracht ziehen, eingeführt und mit traditionellen IR-Bewertungsschemata kombiniert.

Aufgrund der formatunabhängigen internen Dokumentsicht ist es vollkommen transparent, in welchem Format die Ursprungsdokumente verfasst waren. Nicht nur der Korpus kann aus Dokumenten verschiedener Formate bestehen, sondern auch einzelne Treffer selbst. Zu diesem Zweck werden zunächst alle Daten in ein einheitliches XML-Zwischenformat konvertiert. Es wird versucht, die semantische Struktur, die den Dokumenten zugrunde liegt, in eine explizit für die Suchmaschine nutzbare Repräsentation zu überführen. Hierzu werden sowohl auf Heuristiken beruhende Transformationen als auch Annotationen durch Einsatz von Tools aus dem Bereich des Natural Language Processing eingesetzt. Insbesondere bei Anfra-

gen, die implizit einen Datentyp wie beispielsweise eine Orts- oder Datumsangabe enthalten, zeigt sich das erhebliche Potenzial gegenüber einer einfachen untypisierten Textrepräsentation.

Die Experimente haben belegt, dass der in dieser Arbeit vorgestellte Ansatz die Ergebnisgüte signifikant verbessern und dabei effizient durch einen Top-k-Ansatz umgesetzt werden kann. Dies wurde sowohl auf reinen HTML- und XML-Korpora als auch auf hybriden Korpora mit unterschiedlichen Ursprungsformaten demonstriert.

A Wikipedia-Anfragen

Die vier Schemata Wikipedia (W), DBLP (D), IMDB (I) und Homepages (H) wurden den Benutzern zur Formulierung ihrer Anfrage zur Verfügung gestellt. Da der Wikipedia++-Korpus die Wikipedia- und IMDB-Daten umfasst, konnte für diesen Korpus das Vereinigungsschema von Wikipedia- und IMDB-Korpus genutzt werden. Analog konnten für DBLP++ Anfragen im Vereinigungsschema von DBLP und Homepages gestellt werden.
Falls die Anfrage aus mehreren Anfragegruppen bestehen sollte, sollte für jede Gruppe eine eigene Zeile des Schemas genutzt werden. Für eine Join-Bedingung sollten die entsprechenden Spalten mit demselben Buchstaben markiert werden.
Außerdem konnte, falls nach einem speziellen Attribut gefragt wurde ('Wer war ....' → Person), diese Spalte mit einem Fragezeichen markiert werden.

Wikipedia-Schema

W	Person	Location	Date	Organisation	Event	Keywords
-						

DBLP-Schema

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-					

IMDB-Schema

I	Title	Prod.-year	Prod.-country	Prod.-location	actor	plot	Keywords
-							

Homepages-Schema

H	Person	Location	Date	Organisation	Event	Keywords
-						

Wikipedia-Anfragen

1. In welchem Land fanden sowohl eine Sommer- als auch eine Winter-Olympiade statt ?

W	Person	Location	Date	Organisation	Event	Keywords
-		X			summer olympics	
W	Person	Location	Date	Organisation	Event	Keywords
-		X			winter olympics	

2. Welche Wissenschaftler des Max-Planck-Instituts veröffentlichten eine Arbeit auf der VLDB 2001 ?

W	Person	Location	Date	Organisation	Event	Keywords
-	?			Max-Planck		

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-			2001	VLDB-Journal	

3. Mit welchen in Philadelphia geborenen Schauspielern drehte Julia Roberts Filme ?

W	Person	Location	Date	Organisation	Event	Keywords
-	Julia Roberts					
W	Person	Location	Date	Organisation	Event	Keywords
-	actor	Philadelphia			born	

4. In welchen Städten am Rhein wurden Weltmeisterschaften ausgetragen?

W	Person	Location	Date	Org.	Event	Keywords
-		X			world championships	
W	Person	Location	Date	Org.	Event	Keywords
-		X				Rhine

5. Wer hat das World Wide Web erfunden?

W	Person	Location	Date	Organisation	Event	Keywords
-	?				invention	world wide web

6. Wie heißt die amerikanische Politikerin „Rice“ mit Vornamen ?

W	Person	Location	Date	Organisation	Event	Keywords
-	Rice					∼politician

7. An welchem Ort findet man sowohl eine gothische als auch eine romanische Kirche?

W	Person	Location	Date	Organisation	Event	Keywords
-		X				gothic church
W	Person	Location	Date	Organisation	Event	Keywords
-		X				romanic church

8. Wer veröffentlichte auf dem EFIS-Workshop ein Papier über Corba?

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-	?			EFIS	corba

9. Welche deutschen Komponisten wirkten in der 1.Hälfte des 18. Jahundert ?

W	Person	Location	Date	Organisation	Event	Keywords
-	?	Germany	1800-1850			composer

10. Welche französischen Mathematiker des 18. Jahrhunderts gab es?

W	Person	Location	Date	Org.	Event	Keywords
-	?	∼France	1800-1899			∼mathematics

11. Welche amerikanischen Organisationen waren in den Watergate-Skandal verwickelt ?

W	Person	Loc.	Date	Org.	Event	Keywords
-	?, ∼Politician				Watergate Scandal	

12. Welche Ärzte haben sich für Dritte-Welt-Probleme eingesetzt (z.B. Albert Schweitzer)?

W	Person	Location	Date	Organisation	Event	Keywords
-		∼third world				physician

13. Welche deutschen Physiker des 20. Jahrhunderts sind in die USA emigriert und haben den Nobelpreis bekommen haben?

W	Person	Location	Date	Org.	Event	Keywords
-	∼scientist	∼USA,			nobel prize	∼physics,
-		∼Germany				immigration

14. Welche Länder gehörtem den früheren Warschauer Pakt an ?

W	Person	Location	Date	Organisation	Event	Keywords
-		?	1945-1990			Warsaw states

15. Welche Nobelpreisträger der Physik hatten Einfluss auf die Informatik (z.B. Planck wegen Quanten-Computing) ?

W	Person	Location	Date	Org.	Event	Keywords
-	?				nobel prize	∼physics
-						∼computer science

16. Zu welchem Land oder Fürstentum gehörte Prag im 16. Jahrhundert?

W	Person	Location	Date	Org.	Event	Keywords
-		prague	1500-1599			∼country ∼fiefdom

17. In an welchen Orten von Konzerten traten Rock- und Folk-Musiker gemeinsam oder zu verschiedenen Zeitpunkten auf?

W	Person	Location	Date	Organisation	Event	Keywords
-		X			∼concert	rock music

W	Person	Location	Date	Organisation	Event	Keywords
-		X			∼concert	folk music

18. Welche Europäer haben den Friedensnobelpreis erhalten ?

W	Person	Location	Date	Organisation	Event	Keywords
-		∼Europe				nobel prize, peace

19. Wo kämpfte Muhammad Ali zwischen 1970 und 1973?

W	Person	Location	Date	Org.	Event	Keywords
-	Muhammad Ali	?	1970-1973			boxing

20. Welche Firmen gingen nach 2000 an die Börse?

W	Person	Location	Date	Organisation	Event	Keywords
-			> 2000			IPO

21. Welche Personen verunglückten bei Autorennen tödlich ?

W	Person	Location	Date	Org.	Event	Keywords
-					fatal accident	∼car,∼racing

22. Wer gewann in diesem Jahrhundert Nobelpreise ?

W	Person	Location	Date	Organisation	Event	Keywords
-			> 2000			nobel price

23. Welche einflussreichen Personen der französischen Revolution waren Mitglieder der Jakobiner?

W	Person	Location	Date	Organisation	Event	Keyw.
-		∼France		Jacobin Club	french revolution	

24. Welche russische Opernkomponisten gibt es ?

W	Person	Location	Date	Organisation	Event	Keywords
-		∼Russia				opera, composer

25. In welcher Organisation waren sowohl Graf Mirabeau als auch Maximilien Robespierre?

W	Person	Location	Date	Organisation	Event	Keywords
-	Mirabeau			X		

W	Person	Loc.	Date	Org.	Event	Keywords
-	Maximilien Robespierre			X		

26. Welche australische Sängerin hat im Film Moulin Rouge mitgespielt ?

W	Person	Location	Date	Organisation	Event	Keywords
-	?	Australia				singer

I	Title	Prod.-year	...	...	actor	plot	Keywords
-	Moulin Rouge						

27. Welche Schauspieler spielten in Filmen mit, in denen es um eine Atombombe geht ?

I	Title	Prod.-year	...	...	actor	plot	Keywords
-					?	∼nuclear weapon	

28. In welchen Filmen hat der kalifornische Gouverneur mitgespielt ?

W	Person	Location	Date	Organisation	Event	Keywords
-	X	California				governor

I	Title	Prod.-year	...	Prod.-location	actor	plot	Keywords
-					X		Movie

29. Welche anderen Rollen spielten James Bond Darsteller ?

I	Title	Prod.-year	...	...	actor	plot	Keywords
-					X	James Bond	

I	Title	Prod.-year	...	...	actor	plot	Keywords
-					X	-James Bond	

30. Welche Filme spielen in New York nach dem 11 September ?

I	Title	Prod.-year	...	...	...	plot	Keyw.
-	?					> September 11th, 2001	

31. Welche Liebesfilme der letzten Jahre gibt es?

I	Title	Prod.-year	...	Prod.-location	actor	plot	Keywords
-	?	>2000				∼love	

32. In welchen Filmen hat der Mann von Madonna Regie geführt ?

W	Person	Location	Date	Organisation	Event	Keywords
-	X					husband, Madonna

I	Title	Prod.-year	...	Prod.-location	actor	plot	Keywords
-							director=X

33. Welcher Schauspieler, der in Science-Fiction-Filmen mitgespielt hat, ist im realen Leben Gouverneur ?

W	Person	Location	Date	Organisation	Event	Keywords
-	X					governor

I	Title	Prod.-year	...	...	actor	plot	Keywords
-					X		Science-Fiction

34. Welcher in Idar Oberstein geborene Schauspieler läuft in einem Film durch den Nakatomi Tower ?

W	Person	Location	Date	Organisation	Event	Keywords
-	X	Idar Oberstein			born	

I	Title	Prod.-year	...	..	actor	plot	Keywords
-					X	Nakatomi Tower	

35. Welche in Schottland geborenen Schauspieler spielten in James Bond Filmen mit ?

W	Person	Location	Date	Organisation	Event	Keywords
-	X	∼Scotland			born	

I	Title	Prod.-year	...	...	actor	plot	Keywords
-					X,?	James Bond	

36. Welche Filme handeln von politischen Skandalen (z.B. Watergate) ?

I	Title	Prod.-year	...	...	actor	plot	Keywords
-						scandal, ∼politics	

37. Welche Autoren aus der Schweiz haben ein Papier zum Thema P2P auf der VLDB veröffentlicht ?

H	Person	Location	Date	Organisation	Event	Keywords
-		Switzerland				

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-		P2P		VLDB	

38. Wer hat eine Vorlesung über XML gehalten und ein Papier über Indexing veröffentlicht ?

H	Person	Location	Date	Organisation	Event	Keywords
-					∼Course	XML

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-		∼Indexing		VLDB	

39. Welche neueren Publikationen bei der SIGMOD von Mitgliedern deutscher Universitäten gibt es?

H	Person	Location	Date	Organisation	Event	Keywords
-		Germany				University

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-			>1995	SIGMOD	

40. Welcher Autoren, die gerne radfahren, haben Paper auf der VLDB?

H	Person	Location	Date	Organisation	Event	Keywords
-						∼bike

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-				VLDB	

41. Welche SIGMOD-Autoren nehmen an der Blue-Ribbon-Campagne teil?

H	Person	Location	Date	Organisation	Event	Keywords
-						blue ribbon

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-				SIGMOD	

42. Welche SIGMOD Papiere wurden von Autoren deutscher und US-amerikanischer Authoren zusammen geschrieben ?

H	Person	Location	Date	Organisation	Event	Keywords
-		Germany				

H	Person	Location	Date	Organisation	Event	Keywords
-		USA				

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-				SIGMOD	

43. Welche Papiere wurden von Autoren geschrieben, die auf Ihrer Homepage Musik erwähnen?

H	Person	Location	Date	Organisation	Event	Keywords
-						music

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-		?			

44. Welche Publikationen der Universität von Upsala gab es bei der SIGIR 2001?

H	Person	Location	Date	Organisation	Event	Keywords
-		Upsala				University

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-			2001	SIGIR	

45. Welche Veröffentlichungen, die Filmportale behandeln und an der Universität des Saarlandes geschrieben wurden, gibt es ?

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-					∼movie, ∼portal
-					Saarland University

46. Welche Papiere stammen von VDLB-Autoren, die mit EU-Projekten zu tun haben?

H	Person	Location	Date	Organisation	Event	Keywords
-						EU,∼project

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-				VLDB	

47. Welche Phd-Studenten haben auf der VLDB veröffentlicht ?

H	Person	Location	Date	Organisation	Event	Keywords
-						Phd student

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-				VLDB	

48. Welchen Veröffentlichungen zu INEX gibt es von Europäern?

H	Person	Location	Date	Organisation	Event	Keywords
-		∼Europe				

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-					INEX

49. Welche Autoren des MPI haben sowohl Papiere auf der SIGIR als auch auf der SIGMOD publiziert?

H	Person	Location	Date	Organisation	Event	Keywords
-						MPI

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-				SIGIR	

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-				SIGMOD	

50. Welcher deutsche Autor hat sich mit Transaktions-Systemen beschäftigt und hat aktuelle Publikationenen auf der VLDB?

H	Person	Location	Date	Organisation	Event	Keywords
-	X	∼German				transaction systems

D	Author	Title	Pub.-year	Conf./Journal	Keywords
-	X		>2000		

B Inex-Anfragen

In den folgenden beiden Abschnitten sind einige repräsentative INEX-Anfrage der Kategorien *Content and Structure (CAS)* und *Content only (CO)* aufgeführt.

B.1 CAS-Anfragen

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="129" query_type="CAS" ct_no="25">
  <title>
   //article[about(.//atl, new book review bookshelf)]
     //sec[about(., database "data warehouse")]
  </title>
  <description>
     Find book reviews or descriptions that discuss databases
     or data warehousing. Book reviews are typically sections of documents
     titled something along the lines of "new books", "book review", or
     "bookshelf".
  </description>
  <narrative>
     Relevant text describes books that are about databases or
     data warehousing. Books that use databases as motivating examples or
     applications are also relevant, but less so. Books that have chapters
     on databases are relevant, but less relevant than books that are
     entirely about databases or data warehousing. In a sense, the degree
     of relevance of the text is related to the relevance of the book. The
     structural constraints of the query are intended as guidance for the
     system - the user has some knowledge of where the book reviews tend to
     occur.
  </narrative>
  <keywords>
     data warehouse, database, SQL, bookshelf, book review, new books
  </keywords>
</inex_topic>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="127" query_type="CAS" ct_no="13">
  <title>
     //sec//(p| fgc)[about( ., Godel Lukasiewicz and other fuzzy
                              implication definitions)]
```

```
  </title>
  <description>
     Find paragraphs or figure-captions containing the
     definition of Godel, Lukasiewicz or other fuzzy-logic
     implications
  </description>
  <narrative>
     Any relevant element of a section must contain the
     definition of a fuzzy-logic implication operator or a pointer to the
     element of the same article where the definition can be
     found. Elements containing criteria for identifying or comparing fuzzy
     implications are also of interest. Elements which discuss or introduce
     non-implication fuzzy operators are not relevant.
  </narrative>
  <keywords>
    Godel implication, Lukasiewicz implication, fuzzy
    implications, fuzzy-logic implication
  </keywords>
</inex_topic>
```

B.2 CO-Anfragen

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
<inex_topic topic_id="170" query_type="CO" ct_no="39">
  <title>
    "topic maps" +web
  </title>
  <description>
    We are looking for articles discussing the ISO standard
    on topic maps for structuring web sites.
  </description>
  <narrative>
    The goal is to find documents that discuss the usability
    of topic maps to structure web sites. We need this information for
    deciding how to implement meta-data in our web site, but also to learn
    whether the technology can be used to keep track of the structure of
    the site. In particular critical articles concerning the relationship
    between topic maps and other technologies would be of interest. Also
    how it relates to traditional principles of knowledge
    organisation. However, general articles introducing the technology
    would be relevant.</narrative>
  <keywords>
    topic maps, topic map, web structuring, knowledge organisation
  </keywords>
</inex_topic>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE inex_topic SYSTEM "topic.dtd">
```

```
<inex_topic topic_id="187" query_type="CO" ct_no="120">
  <title>
     LSI +"dimension reduction" "latent semantics" "random
     projection" -stoplist "curse of dimensionality"
  </title>
  <description>
     We are looking for articles containing description of
     dimension reduction methods. These methods allow us to lessen effects
     of the "curse of dimensionality" and make retrieval of documents
     faster. Examples of this methods are well-known latent semantic
     indexing (LSI) which improves recall of the retrieval system and
     random projection which does not modify distances too much. We are not
     interested in stoplist filtration which does reduce the dimension a
     bit as a byproduct of term removal.
  </description>
  <narrative>
     When indexing text or multimedia documents as vectors in
     high-dimensional space, all distances are almost the same. This
     results in the infamous "curse of dimensionality" - it is faster to
     scan whole collection than traverse ... .
  </narrative>
  <keywords>
     LSI, latent semantics, latent semantic indexing, curse of
     dimensionality, PCA, principal component analysis, rank-k SVD,
     k-reduced singular decomposition, random projection, projection
     matrix
  </keywords>
</inex_topic>
```

Literaturverzeichnis

[A+97] Serge Abiteboul et al. The Lorel query language for semistructured data. *Int. Journal on Digital Libraries*, 1(1):68–88, May 1997.

[AC96] Martin Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.

[ACD02] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, California, February 2002.*

[ACMM03] Luigi Arlotta, Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Automatic annotation of data extracted from large web sites. In *Proceedings of the Eleventh Italian Symposium on Advanced Database Systems (SEBD 2003), Cetraro (CS), Italy, June 2003.*

[ACR04] Gianni Amati, Claudio Carpineto, and Giovanni Romano. Merging XML Indices. In Fuhr et al. [INEX04], pages 253–260.

[ACS02] Sihem Amer-Yahia, SungRan Cho, and Divesh Srivastava. Tree Pattern Relaxation. In *Proceedings of the 8th International Conference on Extending Database Technology (EDBT 2002), Prague, Czech Republic, March 2002.*

[AG03] Arvind Arasu and Hector Garcia-Molina. Extracting Structured Data from Web Pages. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003), Bangalore, India, March 2003.*

[AKS03] Sihem Amer-Yahia, Nick Koudas, and Divesh Srivastava. Approximate Matching in XML. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003), Bangalore, India, March 2003.*

[Alex] Alexandria Digital Library Gazetteer. Santa Barbara CA: Map and Imagery Lab, Davidson Library, University of California, Santa Barbara. Copyright UC Regents. http://www.alexandria.ucsb.edu/gazetteer.

[ALP04] Sihem Amer-Yahia, Laks V.S. Lakshmannan, and Shashank Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2004), Paris, France, June 2004.*

[AM98] G. Arocena and A. Mendelzon. WebOQL: Restructuring Documents, Databases and Webs. In *Proceedings of the 14th Engineering (ICDE 1998), Orlando, Florida, USA, February 1998.*

[AMM97] P. Atzeni, G. Mecca, and P. Merialdo. To weave the Web. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 1997), Athens, Greece, August 1997.*

[App99] D.E. Appelt. The complete TextPro Reference Manual. http://www.ai.sri.com/ appelt/TextPro, 1999.

[BANY97] Klemens Böhm, Karl Aberer, Erich J. Neuhold, and Xiaoya Yang. Structured Document Storage and Refined Declarative and Navigational Access Mechanisms in HyperStorM. *VLDB Journal*, 6(4):296–311, 1997.

[BCJ+94] G. Elizabeth Blake, Mariano P. Consens, Pekka Kilpeläinen, Per-Åke Larson, T. Snider, and Frank Wm. Tompa. Text / Relational Database Management Systems: Harmonizing SQL and SGML. In *Applications of Databases, Proceedings of the First International Conference (ADB-94) , Vadstena, Sweden, June 1994.*

[BEG+05] Robert Baumgartner, Thomas Eiter, Georg Gottlob, Marcus Herzog, and Christoph Koch. Information Extraction for the Semantic Web. In Norbert Eisinger and Jan Maluszynski, editors, *Reasoning Web*, volume 3564 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2005.

[BHN+02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, California, February 2002.*

[BN02] R. A. Baeza Yates and G. Navarro. XQL and Proximal Nodes. *Journal of the American Society of Information Systems and Technology*, 53(6):504–514, 2002.

[BR01] T. Böhme and E. Rahm. XMach-1: A Benchmark for XML Data Management. In *Proceedings of Datenbanksysteme in Büro, Technik und Wissenschaft (BTW 2001), 9. GI-Fachtagung, Oldenburg, März 2001.*

[BR99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[BSAG98] A. Borthwick, J. Sterling, E. Agichtein, and Grishman R.:. NYU: Description of the MENE Named Entity System as Used in MUC-7. In *Proceedings of the 7th Message Understanding Conference (MUC-7), Manchester, UK, 1998.*

[C+02] Chin-Wan Chung et al. APEX: An Adaptive Path Index for XML data. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002), Madison, Wisconsin, June 2002.*

[CABF05] Emiran Curtmola, Sihem Amer-Yahia, Philip Brown, and Mary Fernandez. Galatex: a conformant implementation of the xquery full-text language. In *Proceedings of Special interest tracks and posters of the 14th international conference on World Wide Web (WWW 2005), Chiba, Japan, 2005.*

[CBD99] S. Chakrabarti, M. van den Berg, and B. Dom. Focused Crawling: A New Approach to Topic-specific Web Resource Discovery. In *Proceedings of the 8th International WWW Conference (WWW 1999), Toronto, Canada, May 1999.*

[CK01] Taurai Tapiwa Chinenyanga and Nicholas Kushmerick. Expressive and Efficient Ranked Querying of XML data. In *Proceedings of the Fourth International Workshop on the Web and Databases (WebDB 2001), Santa Barbara, California, USA, May 2001.*

[CMBT02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). Philadelphia, July 2002.*

[CMP02] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: automatic data extraction from data-intensive web sites. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002), Madison, Wisconsin, June 2002.*

[Coh03] Sara Cohen et al. XSEarch: A Semantic Search Engine for XML. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, September 2003.*

[Coh04] Wiliam W. Cohen. MinorThird. Project Web Site, http://minorthird.sourceforge.net/, 2004.

[Coh98] William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD 1998), Seattle, Washington, USA, June 1998.*

[Conn] Connexor - Natural Knowledge: Machinese product family. http://www.connexor.com/.

[CRF00] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *Proceedings of the Third International Workshop on the Web and Databases (WebDB 2000), Dallas, Texas, USA, May 2000.*

[CRF03] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A Comparison of String Metrics for Matching Names and Records. In *Proceedings of the VLDB Workshop on Information Integration on the Web (IIWeb 2004), Toronto, 2004.*

[Cun02] H. Cunningham. GATE, a General Architecture for Text Engineering . *Computers and the Humanities*, 36:223–254, 2002.

[DBLP] Michael Ley. DBLP XML Records. http://dblp.uni-trier.de/xml/. Downloaded Sep 1st, 2003.

[Dewey] Online Computer Library Center. Introduction to the Dewey Decimal Classification. http://www.oclc.org/oclc/fp/about/about_the_ddc.htm.

[DFF+98] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, and Dan Suciu. XML-QL. In *QL '98, The Query Languages Workshop, W3C Workshop*, December 1998.

[DNS91] D. DeWitt, J. Naughton, and D. Schneider. An Evaluation of Non-Equijoin Algorithms. In *Proceedings of the 17th Conference on Very Large Databases (VLDB 1991), Barcelona, Spain, 1991.*

[DOM] W3C DOM Working Group. Document Object Model (DOM) Level 2 Traversal and Range Specification, 2000. http://www.w3.org/TR/2000/PR-DOM-Level-2-Traversal-Range-20000927/DOM2-Traversal-Range.pdf.

[DS87] Paul F. Dietz and D.D. Sleaton. Two Algorithms for Maintaining Order in a List. In *19th Annual ACM Symposium on Theory of Computing (STOC 1987), New York, 1987.*

[EBNF] ISO/IEC – International Organization For Standardization. 14977-2:1996: Extended Backus-Naur form. 1996.

[Fell98] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[FG01] Norbert Fuhr and Kai Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2001), New Orleans, Louisiana, USA, September 2001.*

[FK00] Dayne Freitag and Nicholas Kushmerick. Boosted Wrapper Induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, Austin, Texas, USA, July 2000.*

[INEX04] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Zoltán Szlávik, editors. *Advances in XML Information Retrieval, Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Dagstuhl Castle, Germany, December 6-8, 2004, Revised Selected Papers*, volume 3493 of *Lecture Notes in Computer Science.* Springer, 2005.

[FLN03] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2001), Santa Barbara, California, USA, May 2001.*

[FR98] Norbert Fuhr and Thomas Rölleke. HySpirit - A Probabilistic Inference Engine for Hypermedia Retrieval in Large Databases. In

Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98), Valencia, Spain, March 1998.

[FSC+03] Mary F. Fernández, Jérôme Siméon, Byron Choi, Amélie Marian, and Gargi Sur. Implementing Xquery 1.0: The Galax Experience. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, September 2003.*

[Fuhr02] Norbert Fuhr. XML Information Retrieval and Information Extraction. In F. Franke, G. Nakhaeizadeh, and I. Renz, editors, *Text Mining. Theoretical Aspects and Applications*, pages 21–32, Heidelberg, 2003. Physica Verlag.

[G+03] Lin Guo et al. XRANK: ranked keyword search over XML documents. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003), San Diego, California, USA, June 2003.*

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000), Cairo, Egypt, September 2000.*

[GBT+04] Jens Graupmann, Michael Biwer, Christian Zimmer, Patrick Zimmer, Matthias Bender, Martin Theobald, and Gerhard Weikum. COMPASS: A Concept-based Web Search Engine for HTML, XML, and Deep Web Data. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB 2004), Toronto, Canada, August 2004.*

[GCS05] Jens Graupmann, Jun Cai, and Ralf Schenkel. Automatic Query Refinement Using Mined Semantic Relations. In *ICDE Workshop on Challenges in Web Information Retrieval and Integration (WIRI), Tokyo, Japan, 2005.*

[GKB+04] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. The Lixto Data Extraction Project - Back and Forth between Theory and Practice. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2004), Paris, France, June 2004.*

[GKFL03] Norbert Gövert, Gabriella Kazai, Norbert Fuhr, and Mounia Lalmas. Evaluating the effectiveness of content-oriented XML retrieval. Technical report, University of Dortmund, Computer Science 6, 2003.

[Grau04] Jens Graupmann. Concept-Based Search on Semi-structured Data Exploiting Mined Semantic Relations. In *Current Trends in Database Technology - EDBT 2004 Workshops, EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 2004, Revised Selected Papers.*

[Gris97] R. Grishman. Architecture Design Document Version 2.3. Technical report, DARPA, 1997.

[Gru02] T. Grust. Accelerating XPath Location Steps. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002), Madison, Wisconsin, June 2002.*

[GS03] Torsten Grabs and Hans-Jörg Schek. PowerDB-XML: Scalable XML Processing with a Database Cluster. In Henk Blanken, Torsten Grabs, Hans-Jörg Schek, Ralf Schenkel, and Gerhard Weikum, editors, *Intelligent Search on XML Data*, volume 2818 of *LNCS*. Springer, September 2003.

[GS05] Jens Graupmann and Ralf Schenkel. The Light-Weight Semantic Web: Integrating Information Extraction and Information Retrieval for Heterogeneous Environments. In *SIGIR Workshop on Heterogeneous and Distributed Information Retrieval (HDIR 2005), Salvador, Brazil, August 2005.*

[GSBG98] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity Search in Databases. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB 1998), New York City, New York, USA, August 1998.*

[GSW05] Jens Graupmann, Ralf Schenkel, and Gerhard Weikum. The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005), Trondheim, Norway, August 2005.*

[GW97] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 1997), Athens, Greece, August 1997.*

[HAB+96] Jerry R. Hobbs, Douglas E. Appelt, John Bear, David J. Israel, Megumi Kameyama, Mark E. Stickel, and Mabry Tyson. FAS-

TUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. *The Computing Research Repository (CoRR)*, cmp-lg/9705013, 1997.

[Hak71] S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

[HFAN98] Gerald Huck, Peter Fankhauser, Karl Aberer, and Erich J. Neuhold. Jedi: Extracting and Synthesizing Information from the Web. In *Proceedings of the 3rd International Conference on Cooperative Information Systems (IFCIS 1998), New York City, New York, USA, August 1998*.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *Proceedings of the 28th International Conference on Very Large Data Bases, (VLDB 2002), Hong Kong, China, August 2002*.

[HTK00] Y. Hayashi et al. Searching Text-rich XML Documents with Relevance Ranking. In *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval, Athens, Greece, July 2000*.

[IMDB] The Internet Movie Database (IMDB). http://www.imdb.org, Stand vom 25.10.2004.

[JHOT] Erich Gamma: The JHotDraw graphics framework (Open Source). http://www.jhotdraw.org/.

[JTIDY] Andy Tripp, Fabrizio Giustina, Gary L Peskin, Russell Gold, and Sami Lempinen: JTIDY - HTML Parser and pretty-printer in Java. http://jtidy.sourceforge.net/.

[Kep04] Stephan Kepser. A Simple Proof for the Turing-Completeness of XSLT and XQuery. In *Extreme Markup Languages, Montreal, Canada, August 2004*.

[Kil99] Pekka Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, Dept. of Computer Science, University of Helsinki, 1999.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A Query System for the World-Wide Web. In *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB 1995), Zurich, Switzerland, September 1995*.

[LN99] Y.-K. Ng S.J. Lim. An automated approach for retrieving hierarchical data from HTML tables. In *Proceedings of the 1999 ACM International Conference on Information and Knowledge Management (CIKM 1999), Kansas City, Missouri, USA, November 1999.*

[LPH00] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000), San Diego, California, USA, February 2000.*

[LT95] Hongjun Lu and Kian-Lee Tan. On Sort-Merge Algorithm for Band Joins. *IEEE Transactions on Knowledge and Data Engineering* , 7(3):508–510, 1995.

[Mar03] Philippe Martin. Correction and Extension of WordNet 1.7 for Knowledge-based Applications. In *Proceedings of the 11th International Conference on Conceptual Structures (ICCS 2003), Dresden, Germany, July 2003.*

[MBC03] D. Maynard, K. Bontcheva, and H. Cunningham. Towards a semantic extraction of named entities. In *Recent Advances in Natural Language Processing (RANLP 2003), Borovets, Bulgaria, September 2003*, 2003.

[MGM99] Andrei Mikheev, Claire Grover, and Marc Moens. XML Tools And Architecture for Named Entity Recognition. *Markup Languages*, 1(3):89–113, 1999.

[Mih96] G. Mihaila. WebSQL—A SQL-like Query Language for the World Wide Web, Master's thesis, Department of Computer Science, University of Toronto, 1996.

[MJKZ98] Sung-Hyon Myaeng, Don-Hyun Jang, Mun-Seok Kim, and Zong-Cheol Zhoo. A Flexible Model for Retrieval of SGML Documents. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1998), Melbourne, Australia, August 1998.*

[MMK99] Ion Muslea, Steven Minton, and Craig A. Knoblock. A Hierarchical Approach to Wrapper Induction. In *Proceedings of the 3rd international Conference on Autonomous Agents (Agents 1999), Seattle, Washington, May 1999.*

[MUC98] *Proceedings of the 7th Message Understanding Conference (MUC-7).* Morgan Kaufmann, 1998.

[NR99] Surya Nepal and M. V. Ramakrishna. Query Processing Issues in Image (Multimedia) Databases. In *Proceedings of the 15th International Conference on Data Engineering (ICDE 1999), Sydney, Austrialia, March 1999.*

[One04] Patrick E. O'Neil et al. ORDPATHs: Insert-Friendly XML Node Labels. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2004), Paris, France, June 2004.*

[Ple81] J. Plesn'ik. A bound for the Steiner tree problem in graphs. *Math. Slovaca*, 1981:155–163, 31.

[PLYM96] W. Paik, E.D. Liddy, E. Yu, and M. McKenna. *Corpus Processing for Lexical Acquisition.*, chapter Categorizing and Standardizing Proper Nouns for Efficient Information Retrieval, pages 61–73. MIT Press, 1996.

[RBJ89] Vijay V. Raghavan, Peter Bollmann, and Gwang S. Jung. Retrieval System Evaluation Using Recall and Precision: Problems and Answers. In *Proceedings of the 12th International Conference on Research and Development in Information Retrieval (SIGIR 1989), Cambridge, Massachusetts, USA, June 1989.*

[RC01] Soumya Ray and Mark Craven. Representing Sentence Structure in Hidden Markov Models for Information Extraction. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001), Seattle, Washington, USA, August 2001.*

[RW94] S.E. Robertson and S. Walker. Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval. In *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1994), Dublin, Ireland, July 1994.*

[SA99] Arnaud Sahuguet and Fabien Azavant. Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F. In *Proceedings of the 25th International Conference on Very Large Data Bases, (VLDB 1999), Edinburgh, Scotland, UK, September 1999*, 1999.

[Sch02] Torsten Schlieder. Schema-Driven Evaluation of Approximate Tree-Pattern Queries. In *Proceedings of the 8th International Conference on Extending Database Technology (EDBT 2002), Prague, Czech Republic, March 2002.*

[SG83] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, first edition, 1983.

[STSW02] Sergej Sizov, Martin Theobald, Stefan Siersdorfer, and Gerhard Weikum. BINGO!: Bookmark-Induced Gathering of Information. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering (WISE 2002), Singapore, December 2002*.

[SWK+02] A. Schmidt et al. XMark: A Benchmark for XML Data Management. In *Proceedings of the 28th International Conference on Very Large Data Bases, (VLDB 2002), Hong Kong, China, August 2002*.

[Theo03] Anja Theobald. *Die XXL-Suchmaschine zur ontologiebasierten Suche auf XML-Daten*. PhD thesis, Saarland University, 2003.

[Theo06] Martin Theobald. *TopX: Efficient Top-k Query Processing for text and semistructured data*. PhD thesis, Saarland University, Max-Planck-Institute for Computer Science, 2006.

[TREC] Text REtrieval Conference (TREC), National Institute of Standards and Technology (NIST). http://trec.nist.gov/.

[TS04] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In Fuhr et al. [INEX04], pages 16–40.

[TVK+02] Igor Tatarinov, Stratis Viglas, Kevin S. Beyer, Jayavel Shanmugasundaram, Eugene J. Shekita, and Chun Zhang. Storing and querying ordered XML using a relational database system. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002), Madison, Wisconsin, June 2002*.

[TW02] Anja Theobald and Gerhard Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In *Proceedings of the 8th International Conference on Extending Database Technology (EDBT 2002), Prague, Czech Republic, March 2002*.

[VPG04] J.-N. Vittaut, B. Piwowarski, and P. Gallinari. An Algebra for Structured Queries in Bayesian Networks. In Fuhr et al. [INEX04], pages 58–64.

[WH02] Y .Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the the Eleventh International World Wide Web Conference (WWW 2002), Honolulu, Hawaii, USA, May 2002*.

[WIKI] Wikipedia, the free encyclopedia. http://en.wikipedia.org/, Stand vom 11.02.2005.

[WSM05] Felix Weigel, Klaus U. Schulz, and Holger Meuss. The BIRD Numbering Scheme for XML and Tree Databases - Deciding and Reconstructing Tree Relations Using Efficient Arithmetic Operations. In *Proceedings of the Third International XML Database, Symposium (XSym 2005), Trondheim, Norway, August 2005.*

[XAMP] Felix Golubov. XAmple XML Editor (Open Source Software). http://www.felixgolubov.com/XMLEditor.

[XPATH02] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML Path Language (XPath) 2.0, November 2002. http://www.w3.org/TR/xpath20/.

[XQUERY] S. Boag et al. XQuery 1.0: An XML Query Language. W3C Recommendation, World Wide Web Consortium, 2002. `http://www.w3.org/TR/xquery`.

[YJR03] C. Yu, H. V. Jagadish, and D. Radev. Querying XML using structures and keywords in Timber. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2003), Toronto, Canada, July 2003.*

[YTT01] M. Yoshida, K. Torisawa, and J. Tsujii. A method to integrate tables of the World Wide Web. In *In Proceedings of the International Workshop on Web Document Analysis (WDA 2001), Seattle, Washington, September 2001.*