JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

# Adaptive Suche mit thematischen Ontologien

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Masterstudium

INFORMATIK

Eingereicht von:
*Florian König, 0255220*

Angefertigt am:
*Institut für Informationsverarbeitung und Mikroprozessortechnik*

Betreuung:
*o. Univ. Prof. Dr. Jörg R. Mühlbacher*

Mitbetreuung:
*Alexandros Paramythis MSc*

*Linz, Oktober 2008*

# Kurzfassung

Diese Arbeit beschäftigt sich mit der wissenschaftlichen Beschreibung, Weiterentwicklung und Evaluierung der adaptiven personalisierten Meta-Suchmaschine Prospector. Dieses webbasierte System nützt die thematische Ontologie des Open Directory Project, um Suchergebnisse mit semantischen Metadaten anzureichern. Auf Basis dieser Informationen können durch Rückmeldungen von Benutzern deren Interessen modelliert werden. Bei einer Suche werden diese Modelle zum Umreihen der Ergebnisse verwendet, um die für den Benutzer relevantesten näher an den Anfang der Liste zu bringen.

Nach einer anfänglichen Motivierung des Themas werden die Grundlagen adaptiver Systeme und insbesondere adaptiver Suche erläutert. Näher eingegangen wird hierbei auf die Möglichkeiten zur Modellierung der Benutzer, Personalisierung von Systemverhalten und speziellen Herausforderungen in diesem Forschungsbereich. Konkretisiert werden die Ausführungen durch die Nennung von Anwendungen und die Beschreibung existierender Systeme.

Der Begriff der Ontologie wird definiert und als Ausprägung davon die Datenbasis des Open Directory Project beschrieben. Behandelt werden deren Struktur, Inhalt und Datenformat. Anschließend wird das System Prospector im Detail vorgestellt. Die personalisierten Funktionalitäten werden in das zuvor beschriebene wissenschaftliche Rahmenwerk adaptiver Suche eingeordnet. Anhand der Algorithmen werden die Vorgänge bei der Modellierung und Personalisierung erläutert.

Der Teil zur Weiterentwicklung des Systems behandelt zuerst das neue Carrot$^2$-Framework, und wie Prospector in dieses integriert wurde. Anschließend werden wichtige Erweiterungen des Algorithmus zur Verbesserung des Systemverhaltens beschrieben. Erläutert wird auch die Implementierung der umfassenden Systemüberwachung, mit der alle Abläufe des Systems zum Zwecke der Evaluierung oder als Quelle künftiger Modellierungsansätze protokolliert werden. Um das System praxistauglich zu machen, wurde seine Leistung analysiert und erfolgreich optimiert; die Ergebnisse und Erkenntnisse dieser Maßnahmen werden vorgestellt.

Zur Überprüfung der Wirksamkeit der Weiterentwicklungen wurde Prospector evaluiert. Die Gestaltung dies Studie, ihr Ablauf, die verwendeten Instrumente und die Teilnehmer werden näher beschrieben. Bei den Ergebnissen wird zuerst auf die Eigenschaften der Testbenutzer, deren Erwartungen und allgemein aufgetretene Probleme eingegangen. Besprochen werden die gemischten Einschätzungen zu Nützlichkeit und Suchleistung des Systems sowie der Zufriedenheit der Teilnehmer. Positive Resultate gibt es besonders aus den Bereichen Modellierung und Usability zu berichten. Die Anregungen der Teilnehmer und eigene Ideen geben im abschließenden Ausblick eine Perspektive für künftige Entwicklungen.

# Abstract

This thesis describes the scientific classification, further development and evaluation of the adaptive personalised meta-search engine Prospector. This web-based system uses the thematic ontology of the Open Directory Project for enriching search results with semantic meta-data. On the basis of this information feedback by the users on the results is used for modelling their interest. During search these models will be used for re-ranking the results so as to bring those items that are most relevant to the user further up.

Following an initial motivation for this topic the fundamentals of adaptive systems in general and specifically adaptive search are described. Techniques for modelling users and personalising system behaviour as well as particular challenges in this field of research are covered. Practical usage is described in the form of concrete fields of application and existing systems.

After defining the term ontology the Open Directory Project and its data are introduced. The description covers structure, content and data format of this ontology. Subsequently the Prospector system, its basic idea, history and inner workings are described. On the basis of the algorithms the methods used for modelling and personalisation are explained.

The part on the further development of the system starts off by describing the new Carrot$^2$ framework and how Prospector has been integrated into it. After that important enhancements to the algorithm that should have a positive influence on system behaviour are explained. The implementation of the comprehensive auditing facilities, that can be used for evaluation purposes or as data source for future modelling techniques, are described as well. In order to make the system useable in real life its up until then low performance has been analysed and successfully optimized. The results and findings of this process are presented.

To test the effect of the development Prospector has been evaluated. The design of this study, its process and the participants are described in the beginning. Then the properties of the users, general problems that occurred and expectations of the participants are covered as first results. Mixed conclusions are drawn with respect to the utility and performance of the system as well as the satisfaction of the users. Positive results can be reported in the area of modelling and usability. The feedback and suggestions of users as well as personal ideas provide the basis for the outlook on future developments at the end.

# Danksagung

Diese Arbeit wäre nicht ohne die Wegweisung und Unterstützung von Kollegen und mir nahestehenden Menschen möglich gewesen. Sie forderten und förderten mich gleichermaßen, gaben wertvolle Hinweise und halfen mir über schwierige Zeiten hinweg. An dieser Stelle möchte ich mich recht herzlich bei ihnen allen bedanken.

Besonderer Dank gilt meinem betreuenden Assistenten Herrn Alexandros Paramythis MSc. Mit seiner jahrelangen Erfahrung im Bereich adaptive Systeme stand er mir immer hilfreich zur Seite. Durch fachlich fundierte Anregungen und klare Zielsetzungen konnte er mich stets in die richtige Richtung leiten und meine Leistungen steigern.

Sehr herzlich bedanken möchte ich mich auch beim Vorstand des Instituts für Informationsverarbeitung und Mikroprozessortechnik Herrn o. Univ. Prof. Dr. Jörg Mühlbacher. Er ermöglichte mir die Arbeit an diesem spannenden Projekt, stellte die nötige Infrastruktur zur Verfügung und übernahm die verantwortungsvolle Aufgabe der leitenden Betreuung.

Finanziert wurde diese Arbeit dankenswerterweise über das Projekt P20260-N15 „Adaptive Support for Collaborative e-Learning“ (ASCOLLA) des Fonds zur Förderung der wissenschaftlichen Forschung (FWF).

Dank gebührt auch allen Mitarbeiterinnen und Mitarbeiter des Instituts für Informationsverarbeitung und Mikroprozessortechnik. Sie unterstützten mich während des Projekts in administrativen und technischen Belangen und gaben mir immer das Gefühl, ein geschätzter Kollege in ihrem Team zu sein.

Meinen Dank aussprechen möchte ich auch an Herrn Lex van Velsen MSc von der Universität Twente in Enschede in den Niederlanden. Durch sein Wissen auf dem Gebiet der Durchführung von Benutzerstudien trug er wesentlich zum Erfolg der Evaluierung bei. Mein Dank gilt an dieser Stelle auch den zahlreichen Teilnehmern der Studie. Durch ihre Rückmeldungen haben sie den Grundstein für die erfolgreiche Weiterentwicklung dieses Projekts gelegt.

Ganz besonders bedanken möchte ich mich auch bei meinen Eltern Eva und Bernhard König. Ihre beständige Unterstützung in allen Belangen des Lebens und Lernens machten mein Studium erst möglich. Es ist schön, Eltern zu haben, die den Wert einer akademischen Ausbildung zu schätzen wissen und ihrem Kind diese Chance bieten.

Mein allergrößter Dank gilt jedoch Lena, die mich schon viele Jahre meines Studiums begleitete und mir auch in den schwierigen Phasen dieser Arbeit stets zur Seite gestanden ist. In all der Zeit gab sie mir Kraft und Ausgleich und heiterte mich auf, wenn einmal etwas nicht so wie gewünscht funktionierte.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Quellcodeverzeichnis

# Abkürzungsverzeichnis

**AJAX** Asynchronous JavaScript and XML

**API** Application Programming Interface

**CGI** Common Gateway Interface

**CSS** Cascading Style Sheets

**CSV** Comma Separated Values

**DOM** Document Object Model

**GPS** Global Positioning System

**HSQLDB** Hyperthreaded Structured Query Language Database

**HTML** Hyper-Text Markup Language

**HTTP** Hyper-Text Transfer Protocol

**IR** Information Retrieval

**JDK** Java Development Kit

**JSON** JavaScript Object Notation

**JSP** Java Server Pages

**MD5** Message-Digest algorithm 5

**MVC** Model-View-Controller

**ODP** Open Directory Project

**REST** Representational State Transfer

**RDF** Resource Description Framework

**SQL** Structured Query Language

**SOAP** Simple Object Access Protocol

**UML** Unified Modeling Language

**URL** Uniform Resource Locator

**WAR** Web Application Archive

**XML** Extensible Markup Language

**XSL** Extensible Stylesheet Language

**XSLT** XSL Transformations

**YUI** Yahoo! User Interface Library

# Kapitel 1

# Einleitung

Ziel dieses Kapitels ist es, zuerst eine Motivation für das Thema der adaptiven personalisierten Suche mit thematischen Ontologien zu geben, und die Problemstellung zu umreißen. Anschließend werden grundlegende Lösungsideen entwickelt. Diese sind nicht detailliert ausgeführt, sondern geben nur die Richtung für die Entwicklung eines Systems vor, mit dem die zuvor geschilderten Probleme in Angriff genommen werden können. Im nachfolgenden Systemüberblick wird eine auf hoher Ebene gehaltene Übersicht zu den bisherigen Funktionalitäten des in dieser Arbeit behandelten Suchsystems Prospector und dessen Datenbasis gegeben. Den Abschluss bildet die konkrete Aufgabenstellung, die sich aus den Lösungsideen sowie den bisherigen Entwicklungs- und Evaluierungsaktivitäten in Bezug auf Prospector ergeben.

## 1.1 Motivation und Problemstellung

Im Juli 2008 überschritt die führende Web-Suchmaschine Google bei ihrer Indizierung des World Wide Web eine bedeutsame Grenze: das System zum Verarbeiten von Links, um neue Inhalte zu finden, zählte eine Billion eindeutige URLs in seinem Datenbestand. In Wirklichkeit fand das System mehr als eine Billion URLs, doch viele verwiesen auf den selben Inhalt. Zu dieser Billion Seiten kommen laut Google täglich mehrere Milliarden hinzu [Goo08d]. Die Zahl der tatsächlich auf Google durchsuchbaren Seiten beträgt laut eigenen Angaben 2008 mehr als 8 Milliarden [KOS08], 2005 waren es noch rund 4,3 Milliarden [KL05]. Diese gewaltige Menge an Informationen im World Wide Web führt bei den Benutzern zu einem „information overload“ [MGSG07].

Suchmaschinen setzen sich zum Ziel, diese riesige Menge an Daten durchsuchbar zu machen. Das Problem der Dienste ist dabei aber weniger, etwas nicht zu finden, sondern die Auswahl der angezeigten Ergebnisse. Eine Vorstellung von der Größe der Ergebnislisten gängiger Suchmaschinen sollen die Zahlen in Tabelle 1.1 auf der nächsten Seite geben. Von diesen Ergebnissen sind viele für den Suchenden von geringer Relevanz und nur die zehn oder zwanzig zuoberst Gereihten werden betrachtet. Dabei mögen die gefundenen Seiten wohl für die Begriffe der Suchanfrage relevant sind, doch diese drückt nur einen Teil des wahren Informationsbedürfnisses des Suchenden aus [KL05].

| **Suchbegriff** | **Google** | **Yahoo!** | **MS Live Search** |
|---|---|---|---|
| Linz | 30.500.000 | 47.500.000 | 11.900.000 |
| Pöstlingberg | 70.000 | 226.000 | 43.200 |
| Raabheim | 3.410 | 517 | 224 |

**Tabelle 1.1:** Ergebniszahlen laut Angabe der jeweiligen Suchmaschine (7.8.2008)

Für diese *unzulängliche Ausdrucksstärke* der Suchanfragen gibt es einige Gründe. Der wohl schwerwiegendste ist deren Kürze: eine typische Suchanfrage besteht nur aus zwei bis drei Begriffen und kann dementsprechend sehr vage sein [MGSG07]. Interessant ist hierbei, dass in einer Studie mit über 13.000 Suchvorgängen die Anfragen bei (für den Benutzer) ergebnislosen Suchen mit durchschnittlich 2.94 Begriffen länger waren als jene bei Suchen mit relevanten Ergebnissen (2.78 Begriffe) [CS07].

Ein weiterer Grund für mangelnde Ausdruckskraft von Suchanfragen ist die menschliche Sprache, in der diese formuliert sind. Sie ist inhärent ungenau und zweideutig und verursacht das *Wortschatz-Problem* (vocabulary problem) [FLGD87]. Zwei Arten von Wörtern können hierbei für Ungenauigkeiten sorgen:

- *Synonyme*: mehrere Wörter besitzen die selbe Bedeutung. Ihre Verwendung kann dazu führen, dass relevante Informationen nicht gefunden werden, wenn die Suchanfrage nicht exakt das gleiche Wort wie im gesuchten Dokument enthält.

- *Homonyme*: ein Wort besitzt mehrere Bedeutungen. Durch sie können nicht relevante Dokumente in der Ergebnisliste aufscheinen, wenn sie das Wort enthalten.

Bisher mussten die Benutzer selbst die Ausdruckskraft ihrer Suchanfragen erhöhen, indem sie beispielsweise nach einer ersten Suche mit unbefriedigendem Ergebnis ihre Anfrage verfeinerten. Doch Benutzer widmen dem Formulieren von Suchanfragen, dem Warten auf Ergebnisse und dem Durchsehen der einzelnen Seiten immer *weniger Zeit und Geduld* [MGSG07]. Wenn dies der Fall ist, oder die Benutzer schlichtweg nicht in der Lage sind, vollständig anzugeben, was sie suchen, so können intelligente Suchsysteme anhand der Benutzerinteraktionen trotzdem interessante Ergebnisse liefern [AM05].

Gängige Suchmaschinen bezogen jedoch bis jetzt bei der Berechnung der Relevanz eines Dokuments nur dessen Inhalt und die Struktur der Verlinkung mit ein. Sie gaben daher in einem gewissen Zeitrahmen, unabhängig davon wer eine Anfrage stellt, bei gleichen Suchbegriffen immer die selbe Ergebnisliste zurück [AM05] [KL05]. Lediglich grobe Einschränkungen der Suche hinsichtlich der bevorzugten Sprache und des Aufenthaltsorts der Benutzer wurden gemacht. Dieses Fehlen von Differenzierungen beim Reihen der Ergebnisse, besonders was ungenaue Suchanfragen betrifft, ist einer der Hauptkritikpunkte heutiger Web-Suchmaschinen. Die *Vielfalt der Informationsbedürfnisse* der Benutzer wird nicht berücksichtigt und schränkt so die Fähigkeiten der Suchmaschinen ein, zweideutige Anfragen zufriedenstellend zu beantworten [CS07].

Ein weiteres Problem heute gängiger Suchmaschinen ist, dass sie die Dokumente nur als Mengen von Begriffen betrachten. Die *Semantik* des Inhalts bleibt ihnen weitgehend verschlossen und ist somit auch für die Anwender nicht bei der Suche nutzbar. Diese Informationen könnten aber helfen, Zweideutigkeiten in den Suchanfragen zu klären und so die Suche in die vom Benutzer intendierte Richtung zu lenken. Sucht man beispielsweise nach „Kiwi Nachspeise“, so kann abgeleitet werden, dass man die Frucht zu einem Dessert verarbeiten möchte und nicht das vom Aussterben bedrohte Wappentier Neuseelands. Wenn also aus dem Zusammenhang rund um ein Vorkommen des Wortes „Kiwi“ in einem Dokument hervorgeht, dass es sich um den Vogel handelt, kann dieses Dokument als nicht relevant eingestuft werden.

Die Entwicklung des World Wide Web in den letzten Jahren, hin zu mehr Einbindung der Benutzer in Inhaltsgenerierung, Filterung und Verbreitung von Informationen, wurde von den großen Suchmaschinen ebenso nicht mitgetragen. Doch die *Benutzergemeinde* kann wertvolle Dienste leisten (zum Beispiel durch die Bewertung von Inhalten) und ist durch ihre große Zahl von Mitgliedern auch ein mächtiger Mitspieler in der heutigen Medienlandschaft. Es wurde außerdem die Beobachtung gemacht, dass „Mundpropaganda“ eine der häufigsten Methoden zur Informationsfilterung im täglichen Leben ist. Menschen neigen dazu, jene Bücher zu lesen, Filme anzusehen und Urlaubsdestinationen zu wählen, die ihnen von jemand Vertrautem empfohlen wurden [KL05].

Zusammenfassend kann man sagen, dass es eine große Herausforderung darstellt, in der stetig wachsenden, riesigen Fülle von Seiten des World Wide Web relevante Informationen zu finden. Viele der Schwächen heutiger Suchmaschinen sind dabei auf die Kürze und Ungenauigkeit der Suchanfragen zurückzuführen, im Speziellen auf die Mehrdeutigkeit natürlicher Sprache. Die Informationsbedürfnisse und der Kontext des Suchenden können so nicht genau erfasst werden. Ebenso wenig genützt werden Quellen semantischer Information und die Fähigkeiten der Netzgemeinde, die Informationsmengen direkt zu filtern und zu bewerten. Um nicht die Kontrolle über die Informationsflut zu verlieren, müssen diese Faktoren und insbesondere der Suchende und seine Interessen stärker berücksichtigt werden.

## 1.2 Lösungsideen

Der vorangehende Abschnitt zeigt, dass für Suchmaschinen viele Herausforderungen bestehen, um den Benutzern wirklich relevante Ergebnisse zu liefern. Zur Erreichung dieses Ziels gibt es einige grundlegende Ideen, die in späteren Kapiteln weiter verfeinert und in ihrer Umsetzung beschrieben werden.

Um für die Benutzer die immense Menge an Seiten besser durchsuchbar zu machen, bieten Suchmaschine diverse Möglichkeiten, die Anfragen möglichst unzweideutig und vollständig zu definieren. Einzelne Wörter lassen sich mit booleschen Operatoren wie UND und ODER verknüpfen, von der Suche ausschließen oder zu Phrasen, die als Ganzes im Suchergebnis vorkommen müssen, zusammenfassen. Die Suche kann auf eine bestimmte Sprache, Region oder

Domain eingeschränkt werden. Spezielle Suchmaschinen ermöglichen Suchen in einem gewissen Bereich wie beispielsweise nach Personen, Bildern, Nachrichtenartikeln oder Dateien. Auch auf bestimmte Dateitypen und das ‚Alter' einer Seite (basierend auf dem Änderungsdatum) kann gefiltert werden. Diesen Möglichkeiten gemein ist, dass sie vom Benutzer eine bewusste Beschäftigung mit den Dimensionen der Klassifizierung gewünschter Ergebnisse verlangen und mit einem *erhöhten Aufwand beim Formulieren* der Suchanfrage verbunden sind.

Bedenkt man, dass die meisten Suchanfragen nur zwei bis drei Wörter umfassen, so muss ein System, das die Benutzer beim Suchen im Internet sinnvoll unterstützen soll, *möglichst einfach* sein, um angenommen und verstanden zu werden. Diese Einschränkung betrifft nicht nur den Suchvorgang als solches, endet also nicht mit dem Anzeigen der Resultate. Auch vorbereitende Tätigkeiten wie Registrieren, Angabe von Interessen oder allgemeinen Suchpräferenzen müssen für Benutzer verständlich und nachvollziehbar sein. Ebenso sollte die Suchmaschine in Bezug auf die Ergebnisse die Benutzer nicht über die Maßen mit komplexen Aktionen zum Verfeinern der Suche oder Bewerten des Resultats überfordern.

Betrachtet man die Zahl der Ergebnisse, die gängige Suchmaschinen liefern, so scheint es weniger ein Problem zu sein, dass zu wenige zurückgegeben werden. Zudem werden, wie bereits weiter oben erwähnt, nur die obersten zehn oder zwanzig Ergebnisse betrachtet [KL05]. Um die Suchbedürfnisse der Benutzer befriedigen zu können, müssen die für sie relevanten Seiten daher ganz zu Beginn aufscheinen. Eine Möglichkeit, die keine weitere Interaktion vonseiten der Benutzer erfordert, ist das *Umreihen der Suchergebnisse*. Diese Methode hat auch den Vorteil, dass keine zusätzlichen Ergebnisse angefragt werden müssen und keine (eventuell doch relevanten) verworfen werden.

Traditionelle Suchmaschinen, so wie Google, Yahoo! und Microsoft Live Search es sind, liefern deterministische Ergebnisse in Bezug auf den anfragenden Benutzer. Zwei unterschiedliche Benutzer erhalten also für die selbe Suchanfrage die selbe Liste an Treffern. Die Suchmaschinen versuchen dabei, ein für sehr viele verschiedene Benutzer akzeptables Suchergebnis zu liefern. Um dem Informationsbedürfnis eines jeden einzelnen Benutzers gerecht zu werden, muss das *Suchergebnis personalisiert*, also an aktuelle oder länger bestehende Interessen und Präferenzen des Benutzers adaptiert werden. Verfügt eine derartige adaptive Suchmaschine über solche Informationen, so bilden diese einen Kontext, mit dem die Suchanfrage eindeutiger und vollständiger wird. Die personalisierte Suche kann zweideutige Begriffe wie Homonyme anhand der Informationen über den Benutzer klären und die Anfrage beispielsweise um Synonyme erweitern [MGSG07].

Sollten die Informationen zu den Interessen eines Benutzers nicht schon gespeichert sein, so müssen sie in Erfahrung gebracht werden. Die primäre Quelle ist hierbei der Benutzer selbst. Dabei gilt wiederum, dass dieser Prozess der *Informationsakquirierung nicht zu aufwändig und komplex* sein darf. Ansonsten ergäbe sich ja kein Vorteil gegenüber den oben beschriebenen, schon in Suchmaschinen vorhandenen Mitteln zum Verfassen exakter Suchanfragen.

Neben dem einzelnen Benutzer selbst kann aber auch eine größere Gruppe von Benutzern nützliche Informationen zu ihren Mitgliedern liefern. Sowohl wenn solch eine Zuordnung zu

einer Gruppe manuell vom Benutzer vorgenommen, als auch wenn sie automatisch vom System berechnet wird, können in beiden Fällen die *Gruppenmitglieder von Informationen ihrer Kollegen profitieren* und so ihr Interessensprofil erweitern und verfeinern. Besonders wenn ein Benutzer neu im System ist, kann er von der Mitgliedschaft in einer bestehenden Gruppe stark profitieren. Wo sonst noch wenige Informationen zum Benutzer vorlägen, kann das System nun schon auf Annahmen über ihn anhand seiner Gruppenmitgliedschaft zurückgreifen.

Die Informationen, welche eine derartige adaptive Suchmaschine zu den Interessen und Präferenzen seiner Benutzer hat, sind notwendig für die Personalisierung. Gleichzeitig stellen sie einen möglichen Eingriff in die Privatsphäre der Benutzer dar. Die Möglichkeit einer *anonymen Nutzung*, die zumindest teilweise die Vorteile des vollständig personalisierten Systems bietet, kann hier für Benutzer einen sanften Einstieg bieten und die Angst vor einem „Ausspionieren“ nehmen. Das Vertrauen der Benutzer kann durch eine zweite Maßnahme ebenfalls verbessert werden: sie erhalten *Einsicht in das Modell*, welches das System von ihren Interessen erstellt hat. Über die Möglichkeit, dieses zu bearbeiten, können Benutzer inkorrekte Daten entfernen oder korrigieren.

Die in der Folge beschriebene adaptive Suchmaschine stützt sich auf die soeben ausgeführten Lösungsideen. Wichtig für die praktische Umsetzung ist hierbei zudem eine weitere Idee, nämlich das weitestgehende Nutzen schon bestehender Systeme. Insbesondere auf das überaus komplizierte und aufwändige Entwickeln einer eigenständigen Suchmaschine mit Crawler, Indizierung, Ergebnisabruf und Reihung kann so verzichtet werden. Die vorgeschlagene Methode des Umreihens schon vorhandener Suchergebnisse erlaubt ein Zurückgreifen auf bestehende Suchmaschinen wie Google, Yahoo! oder Microsoft Live Search.

## 1.3 Systemüberblick

Als Grundlage für das Verständnis in den weiteren Kapiteln wird nun ein Überblick über die Version des Prospector-Systems gegeben, die als Basis für die in dieser Arbeit beschriebene Weiterentwicklung diente. Es handelt sich dabei um eine übersichtsartige Beschreibung der Systemfunktionen auf hoher Ebene, meist aus der Sicht des Anwenders. Die technischen Details der Implementierung und das Zusammenspiel werden Kapitel 5 noch erläutert. Ebenfalls kurz beschrieben wird die Datenbasis dieses Systems, die vom Open Directory Project (ODP) stammt und in Kapitel 4 näher beschrieben wird.

### 1.3.1 Prospector

Prospector ist eine web-basierte, adaptive *Meta-Suchmaschine*; das heißt sie generiert nicht selbst die Suchergebnisse, sondern nutzt jene anderer Suchmaschinen wie beispielsweise Google, Yahoo! oder Microsoft Live Search. Das System *personalisiert* seine Ausgaben und soll helfen, für den jeweiligen Benutzer relevante Ergebnisse schneller zu finden. Es lernt von seinen Benutzern, welche Interessen sie bei der Suche haben, und gibt die Ergebnisse in einer

personalisierten Reihenfolge zurück, angepasst an deren Präferenzen. Relevante Ergebnisse verschiebt Prospector dabei in der Ergebnisliste nach vorne, für den Benutzer als irrelevant gewertete Ergebnisse weiter nach hinten. In Kapitel 5 finden sich Details zu Idee, Suchablauf, adaptiven Funktionalitäten und den Algorithmen von Prospector.

#### Modellierung und Personalisierung

Interesse beziehungsweise Desinteresse kann der Benutzer durch *positives* respektive *negatives Bewerten von Suchergebnissen* bekunden. Die Bewertungen verändern im Benutzermodell die Gewichtungen von Themenbereichen in einer Ontologie. Die Gewichtungen der einzelnen Themenbereiche sollen dabei das Interesse des Benutzers an ihnen widerspiegeln. Die Struktur der thematischen Ontologie, in der die Webseiten des Suchergebnisses klassifiziert werden, stammt aus dem Open Directory Project (ODP). Details zu diesem Projekt werden in Unterabschnitt 1.3.2 und Kapitel 4 erläutert. Mit der Zeit baut Prospector durch diese Bewertungen ein Modell der Interessen des Benutzers auf, basierend auf der Struktur der Ontologie. Details zur Modellierung finden sich in Unterabschnitt 5.4.1.

Prospector verwendet diese Informationen zum *Umreihen der Ergebnisse*, sodass diese besser auf den Benutzer passen. Jedes Ergebnis wird entsprechend der Ontologie klassifiziert und diese Klassifikation mit dem Benutzermodell abgeglichen. Die dort verzeichneten Gewichtungen dienen zur Berechnung der Relevanz des Ergebnisses. Anschließend wird die Ergebnisliste nach den errechneten Relevanzen sortiert. Ein wirtschaftlich interessierter Mensch würde in dieser nach persönlichen Interessen umgereihten Liste dann beispielsweise bei der Suche „Bank“ die Finanzinstitute an vorderster Stelle finden, ein Hobbygärtner eher Seiten zu Gartenbänken. Eine Beschreibung der Personalisierung durch Prospector findet sich in Unterabschnitt 5.4.2.

#### Gruppen und Gruppenmodelle

Prospector verfügt auch über Modelle für eine fixe Zahl von *thematischen Gruppen* (arts, computers, business, recreation, ...). Ein Benutzer kann das Ausmaß seines Interesses zu jedem dieser allgemeinen Themengebiete angeben und ist dadurch Mitglied der jeweiligen Gruppe. Über sie werden automatisch Bewertungen zwischen den einzelnen Gruppenmitgliedern ausgetauscht. Die Idee ist, dass Benutzer mit ähnlichen Interessen von den Bewertungen ihrer Gruppenkollegen profitieren und somit ihre eigenen Suchergebnisse verbessern können.

Das Ausmaß ihrer *Affinität* zu einer Gruppe bestimmt dabei, wie stark ihre Bewertungen das Gruppenmodell beeinflussen. Umgekehrt werden die Gewichtungen im Modell einer Gruppe, deren Mitglied ein Benutzer ist, bei der Berechnung der endgültigen Relevanz von Ergebnissen entsprechend der Affinität des Benutzers zu dieser Gruppe gewertet. In die Berechnung fließt auch mit ein, wie viel Interesse insgesamt (bezogen auf alle Benutzer) an der Gruppe besteht. In einer kleinen Gruppe (also mit wenigen Benutzern, die daran Interesse haben) beeinflussen die einzelnen Mitglieder das Modell dieser Gruppe stark. Bei großen Gruppen haben die

einzelnen Mitglieder weniger Einfluss. Nähere Informationen zu Gruppen und deren Modelle werden in 5.4.1 präsentiert.

#### Suche fokussieren

Mithilfe der Gruppenmodelle können Benutzer ihre Suche auch fokussieren. Prospector erlaubt das *Umreihen von Suchergebnissen* nach einem durch den Benutzer gewählten *Gruppenmodell*. Die Berechnung der Relevanz erfolgt dann ausschließlich mit den Gewichtungen dieses einen Gruppenmodells und greift nicht mehr auf das persönliche Benutzermodell zurück. Die Reihung entspricht somit jener, die Mitglieder dieser Gruppe und dementsprechend Personen mit Interesse an deren thematischen Gebiet für sinnvoll erachten würden. Diese Möglichkeit der Umreihung nach einem Gruppenmodell existiert auch *für anonyme, nicht registrierte Benutzer*.

#### Modell einsehen und bearbeiten

Jeder Benutzer von Prospector kann sein persönliches Modell einsehen und bearbeiten. Dieses Konzept eines *scrutable user model* wird durch Kay [Kay00] beschrieben. Angezeigt werden jeweils die Themengebiete des Modells gemäß der Ontologie und ihre Gewichtungen. Die Gewichtung jedes Themas kann verändert werden. Sollte sich ein Gebiet im Benutzermodell finden, zu dem ein Benutzer keine Gewichtung gespeichert haben möchte, so kann er dieses aus dem Modell entfernen.

#### Alleinstellungsmerkmale

Prospector bietet Lösungsansätze für etliche der in Abschnitt 1.1 besprochenen Schwachstellen von Suchsystemen, die nur auf das Vergleichen von Suchbegriffen abstellen. Insbesondere bei homonymen und anderweitig nicht eindeutigen Begriffen kann Prospector helfen, da mit der ontologischen Klassifikation *mehr Semantik* zu den Ergebnissen zur Verfügung steht.

Zusätzlichen Kontext bieten auch die Informationen zu den Interessen eines Benutzers in seinen Benutzer- und Gruppenmodellen. Die *Gruppenmodelle* nutzen hierbei die *kollektiven Bewertungen* einer Gruppe und lassen die Gruppenmitglieder in neuen Suchkontexten, wo noch keine persönlichen Bewertungen vorliegen, also insbesondere nach dem Registrieren als neuer Benutzer davon profitieren.

Als vertrauensbildende Maßnahmen bietet Prospector die anonyme Suche und für registrierte Benutzer die *Einsicht in ihr Benutzermodell*. Die Benutzer können dieses Modell auch bearbeiten und inkorrekte Daten entweder korrigieren oder entfernen.

#### Entwicklung

Bisher wurden zwei Versionen von Prospector entwickelt; beide werden in Abschnitt 5.2 näher beschrieben. Die erste legte die grundlegenden Funktionen des Systems fest und stellte einen ersten lauffähigen Prototypen dar. Sie verfügte bereits über eine Web-Oberfläche und verarbeitete Suchergebnisse von Google. Eine Evaluierung bestätigte das Potenzial des Systems, zeigte aber auch nötige Verbesserungen beim Algorithmus und der Benutzerschnittstelle auf.

Auf Basis dieser Erkenntnisse wurde die zweite Version von Prospector entwickelt. In ihr wurde der Algorithmus zum Modellieren der Benutzer und zum Personalisieren der Ergebnisse grundlegend verändert und verbessert. Auch bei der Web-Oberfläche wurden Änderungen vorgenommen, die eine bessere Verständlichkeit und Bedienbarkeit liefern sollten. Diese Version wurde in einer umfangreicheren Studie von einem unabhängigen Team professionell evaluiert. Die Ergebnisse gaben Aufschluss über die nötigen Weiterentwicklungen, die in Version drei umgesetzt werden sollten und in der vorliegenden Arbeit beschrieben werden.

### 1.3.2 Open Directory Project (ODP)

Das Open Directory Project [ODP08] ist ein nach Themengebieten geordnetes *Verzeichnis von Webinhalten*. Es klassifiziert in seinen mehr als 750.000 vorwiegend hierarchisch strukturierten Kategorien über 4,6 Millionen Links zu Webseiten und anderen Inhalten des World Wide Web. Über die Seite `http://www.dmoz.org` kann man auf die Daten dieses offenen Gemeinschaftsprojekts zugreifen und selbst als einer der über 80.000 Redakteure Webinhalte klassifizieren (alle Zahlen Stand Juli 2008).

Das ODP dient als Datenquelle für das *Google Directory*, das Web-Verzeichnis von Google [Goo08a]. In Prospector werden die ODP-Daten als *ontologische Metadaten* zu den gefundenen Suchergebnissen verwendet. Zu diesem Zweck wird der Datenbestand in einem vom ODP frei zur Verfügung gestellten, RDF ähnlichen Format importiert. Hintergründe zu diesem Projekt, der genaue Aufbau seiner Ontologie sowie eine Beschreibung der Daten finden sich in Kapitel 4.

## 1.4 Aufgabenstellung

Die Entwicklung der dritten Version von Prospector ist Gegenstand dieser Arbeit. Die wichtigsten Weiterentwicklungen werden in der Folge beschrieben. Sie ergaben sich aus bisher noch nicht oder nur wenig implementierten Teilen der Lösungsideen (siehe Abschnitt 1.2) und als Folge der Evaluierungen der zwei vorangegangenen Versionen des Systems.

### Ziele

Auf der guten konzeptionellen und algorithmischen Grundlage der bisherigen Versionen sollte das System insgesamt *gebrauchstauglicher* gemacht werden. Diese Aufgabe umfasste mehrere Dimensionen. Die Benutzerschnittstelle sollte mit aktuellen Web-Technologien auf den derzeitigen Stand der Technik gehoben werden. Die Bedienung von Prospector war zu vereinfachen und verständlicher zu gestalten. Eine Steigerung von Effektivität und Effizienz bei Suche und unterstützenden Aktivitäten wie Registrierung, Bewerten und Modellbearbeitung war ebenfalls ein Ziel.

Prospector sollte zudem mehr Informationen, insbesondere aus der zur Klassifizierung verwendeten Ontologie, verarbeiten und für die Benutzer auch in verständlicher Weise darstellen. Ein großes Ziel war zudem das Erreichen einer mit gängigen Suchmaschinen vergleichbaren Suchgeschwindigkeit. Um Prospector auf eine *solide architektonische Basis* zu stellen und um schon vorhandene Komponenten nützen zu können, sollte es in das Open Source Clustering Framework für Meta-Suchmaschinen Carrot$^2$ eingebunden werden, das in Abschnitt 6.1 näher beschrieben wird.

### Datenquelle

Die neue Version von Prospector sollte *unabhängig von Google* im Bezug auf die Suchergebnisse werden. Schon immer war eine Registrierung nötig, um Zugriff zum Webservice von Google zu erhalten. Als registrierter Benutzer erhielt man daraufhin einen digitalen Schlüssel. Die Anzahl der Suchen bzw. der abgefragten Ergebnisse war pro Schlüssel beschränkt und zu gering, um das System sinnvoll bei einer Evaluierung, geschweige denn im Echtbetrieb einzusetzen. Zudem befindet sich der Dienst im Auslaufen und wird von Google nicht mehr aktiv unterstützt. Viel schwerer wiegt jedoch, dass Google nicht mehr erlaubt, seine Suchergebnisse umzureihen [Goo08b]. Mit den schon vorhandenen Komponenten des Carrot$^2$-Frameworks sollte ein Zugriff auf die Ergebnisse alternativer Suchmaschinen geschaffen werden.

Bereits in der zweiten Version waren die Daten des ODP lokal in einer Datenbank gespeichert und nicht mehr über die Suchergebnisse von Google verfügbar. Für den Zugriff auf diese Datenquelle sollte die *Abfragegeschwindigkeit optimiert* werden, da sie eine der Hauptursachen für die unzureichende Performanz von Prospector ist. Zugleich sollte die Abfrage intelligenter vorgehen und nicht nur anhand des Domainnamens in dem Uniform Resource Locator (URL) eines Ergebnisses klassifizieren, sondern auch einzelne Seiten unterscheiden können.

### Algorithmus

Wie sich in der Evaluierung der zweiten Version gezeigt hatte, war die dort verwendete Heuristik, nur jenes Themengebiet pro Klassifizierung eines Ergebnisses zu verarbeiten, das die meisten Seiten besitzt, nicht erfolgreich. Es sollten daher nun *alle Themen*, denen ein Ergebnis

zugeordnet ist, durch den Algorithmus berücksichtigt werden, sowohl beim Berechnen der Relevanz als auch beim Speichern einer Bewertung. Dies macht es auch nötig, die Gewichtungen unterschiedlicher Themengebiete in einer sinnvollen Weise zu einer Relevanzwahrscheinlichkeit zu kombinieren.

Bisher wurde die *ursprüngliche Reihenfolge* der Suchergebnisse, so wie sie von der dahinter liegenden Suchmaschine zurückgegeben wurden, nur insofern berücksichtigt, als bei Ergebnissen mit gleicher Relevanz auf sie zurückgegriffen wurde. In der neuen Version von Prospector sollte diese Reihenfolge mehr Gewicht bekommen und ihr Einfluss variabel gestaltet werden.

#### Benutzerschnittstelle

Mit der Einbettung von Prospector in das Carrot$^2$-Framework sollte auch dessen bestehende Web-Schnittstelle genützt werden. Mit ihr soll es möglich sein, die *Themenbereiche*, in denen die einzelnen Ergebnisse klassifiziert werden, *darzustellen*. Eine neue Möglichkeit zum Bewerten sollte geschaffen werden, die einerseits positive Bewertungen auch in der Ergebnisliste ermöglicht und andererseits die Nachteile der bisherigen Bewertungsfunktionalität umgeht.

Insgesamt sollte der Bedienkomfort von Prospector stark verbessert werden. Die Anwendung sollte dynamischer, flüssiger und schneller im Gebrauch werden. Bisherige Schwachstellen wie beispielsweise unklare Gruppennamen sollten ausgebessert werden. Die Benutzer sollten durch entsprechende Elemente der Benutzerschnittstelle und farb-basierte Hinweise bei der Wahl von Werten (z. B. Gewichtungen) unterstützt werden.

#### Evaluierung

Zur Überprüfung der Sinnhaftigkeit der Weiterentwicklungen war geplant, die erste größere Evaluierung aller Funktionalitäten von Prospector und mit echten Benutzern durchzuführen. Diese sollten über eine gewisse Zeit Prospector benutzen und für eine qualitative Bewertung Feedback geben. Gleichzeitig sollten die internen Vorgänge im System protokolliert werden, um Daten für eine quantitative Untersuchung zu liefern. Zu diesem Zweck musste ein System zum umfangreichen Protokollieren der Benutzeraktionen und Vorgänge im System entwickelt werden. Dieses sollte für die Auswertung einen schnellen und strukturierten Zugriff auf die protokollierten Daten erlauben.

# Kapitel 2

# Adaptive Systeme

In diesem Kapitel wird ein Überblick über die Grundlagen, Methoden, Usability-Herausforderungen und Anwendungen adaptiver Systeme gegeben. Die wichtigsten Begriffe werden definiert und gängige Praktiken beschrieben. Dies bildet die Grundlage für die Beschreibung adaptiver Suche im darauf folgenden Kapitel 3. Für mehr Details finden sich reichlich Verweise auf die Fachliteratur.

## 2.1 Grundlagen

Adaptive Systeme werden bereits seit mehr als zwei Jahrzehnten erforscht und entwickelt. Trevellyan und Browne [TB87] beispielsweise beschreiben die Arbeit an einem Mitte der 80er Jahre entwickelten, adaptiven hierarchischen Verzeichnisses für Telefonnummern. Die einzelnen Ebenen der Hierarchie ergaben sich dabei anhand der Häufigkeit, mit der eine Nummer in der Vergangenheit gewählt wurde. Häufig gewählte Nummern wurden in einer der oberen Ebenen angezeigt, selten gewählte Nummern in weiter unten liegenden Gruppen.

Durch dieses adaptive Verhalten konnte eine 32-prozentige Verringerung der durchschnittlichen Suchtiefe im Vergleich zu einem statischen System festgestellt werden. Die Zeit pro Nummernwahl konnte um 35% reduziert werden, die Zahl der Fehler pro Menü um 40%. In Kombination mit der verringerten Suchtiefe ergab dies eine Verringerung der Fehlerzahl von 60% pro Nummernwahl. Die meisten Testbenutzer bevorzugten das adaptive System, da es kürzere Suchpfade aufwies und man sich weniger merken musste.

Diese Ergebnisse zeigen, dass bereits früh der Wert adaptiver Systeme insbesondere im Bereich *Mensch-Maschine-Kommunikation* erkannt und erforscht wurde. Über die Zeit erhielten Systeme, die sich an ihre Benutzer anpassen, immer wieder unterschiedliche Bezeichnungen von adaptiven Schnittstellen über Benutzermodellierungssysteme bis hin zu Software-Agenten oder intelligente Agenten. Zu Beginn der 1990er-Jahre setzte sich für die Funktionen der breitere Begriff *Personalisierung* durch [Jam03].

Aufgrund der geschichtlichen Entwicklung des Bereichs der adaptiven Systeme, deren wechselnde Einsatzgebiete und Funktion sowie die unterschiedlichen Benennungen der einzelnen

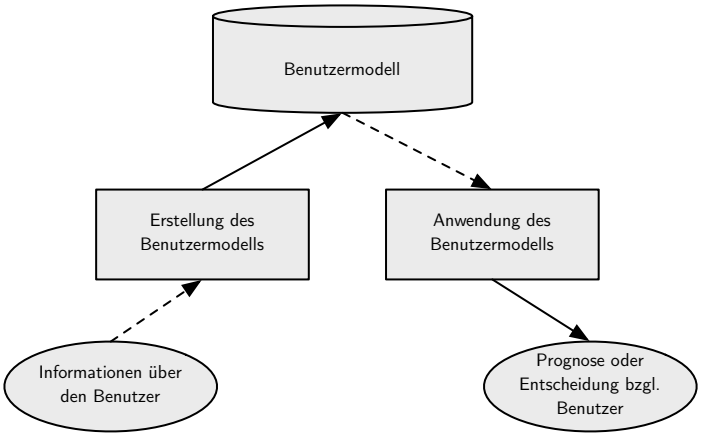


**Abbildung 2.1:** Verarbeitungsschema für ein Benutzer-adaptives System [Jam03]

Ausprägungen ist eine einheitliche Begriffsbestimmung nicht leicht. Eine gängige Definition Benutzer-adaptiver Systeme ist nach Jameson [Jam03]:

> *„A user-adaptive system can be defined as an interactive system that adapts its behavior to individual users on the basis of processes of user model acquisition and application that involve some form of learning, inference, or decision making.“*

Bei der Erstellung des Benutzermodells („user model acquisition“) lernt das System und zieht Rückschlüsse aus den Informationen über den Benutzer. Das Benutzermodell beschreibt dabei den Benutzer nur in bestimmten Aspekten. Das System wendet dieses Modell dann auf relevante Eigenschaften der aktuellen Situation an („user model application“) und bestimmt damit, wie es sein Verhalten an den Benutzer anpassen soll [Jam03].

Jameson [Jam03] entwickelte zu dieser Definition ein allgemeines *Schema des Verarbeitungsablaufs* in einem Benutzer-adaptiven System (siehe Abbildung 2.1). Ovale stehen hierbei für Ein- bzw. Ausgaben, Rechtecke für Verarbeitungsmethoden und der Zylinder für gespeicherte Information. Entlang der gestrichelten Pfeile wird Information verwendet, während Pfeile mit durchgezogenem Strich anzeigen, dass Ergebnisse erzeugt werden. Mit diesem Schema lässt sich in vielen adaptiven Systemen der Datenfluss darstellen.

Eine ähnliche Definition für *adaptive Benutzerschnitttellen* geben Dieterich et al. [DMKSH93]:

> *„Adaptive User Interfaces are designed to tailor a system's interactive behavior with consideration of both individual needs of human users and altering conditions within an application environment. . . . An Adaptive User Interface either supports users in the adaptation of the interface to their own needs and preferences or performs the adaptation automatically. The focus of adaptation extends to a broader range than in current flexible interfaces by including functionality and the demands of the application.“*

Diese Definition betrifft vor allem die Benutzerschnittstelle und ihre Fähigkeiten zur Adaption. Dabei passt sie ihr Verhalten nicht nur an die Bedürfnisse des Benutzers sondern auch

| **Methode** | **Initiative** | **Vorschläge** | **Entscheidung** | **Ausführung** |
|---|---|---|---|---|
| Adaption | Benutzer | Benutzer | Benutzer | Benutzer / System |
| System-initiierte Adaption | System | Benutzer | Benutzer | Benutzer / System |
| Computer-gestützte Adaption | Benutzer | System | Benutzer | System |
| Benutzer-gesteuerte Selbst-Adaption | System | System | Benutzer | System |
| Benutzer-initiierte Selbst-Adaption | Benutzer | System | System | System |
| Selbst-Adaption | System | System | System | System |

**Tabelle 2.1:** Agenten- und Aktivitätskombinationen des Adaptionsprozesses [DMKSH93]

an geänderte Bedingungen in der Umgebung an. Es wird auch die Möglichkeit eingeräumt, dass sich das System nicht automatisch anpasst, sondern dass der Benutzer auch bei einer manuellen Anpassung unterstützt wird.

Dieterich et al. [DMKSH93] identifizieren zu diesem Zweck bei der Adaption des Verhaltens eines Systems zwei mögliche *Akteure*: das System selbst und der Benutzer. Der Adaptionsprozess ist in vier *Aktivitäten* geteilt, wobei jede jeweils durch einen der beiden Akteure ausgeführt bzw. gesteuert wird. Diese sind:

- *Initiative* (initiative): einer der Akteure beschließt, eine Adaption vorzuschlagen.
- *Vorschläge* (proposals): Möglichkeiten der Adaption werden vorgeschlagen.
- *Entscheidung* (decision): eine der Möglichkeiten wird gewählt.
- *Ausführung* (execution): die gewählte Adaption wird ausgeführt.

Bei zwei Agenten und vier Aktivitäten ergeben sich so 16 unterschiedliche Kombinationen; die interessantesten davon haben Dieterich et al. explizit benannt. Sie sind in Tabelle 2.1 aufgelistet.

Die *Ziele von Adaption* sehen Dieterich et al. ebenfalls aus der Sicht von adaptiven Benutzerschnittstellen. Nichtsdestotrotz sind viele davon auch allgemein für adaptive Systeme gültig. Konkret soll Adaption die Bedienung vereinfachen und insbesondere komplexe Systeme benutzbar machen. Sie soll sicherstellen, dass ein System effektiv und effizient zu benutzen ist und jene Inhalte darstellt, die der Benutzer sehen will. Durch Adaptivität soll die Benutzerschnittstelle für eine heterogene Benutzergruppe besser geeignet sein und zunehmende Erfahrung in der Bedienung berücksichtigen [DMKSH93].

Wichtig ist auch die *Unterscheidung* zwischen adaptiven und *adaptierbaren Systemen*. Letztere bieten für den einzelnen Benutzer nämlich lediglich die Möglichkeit, sie an seine eigenen Bedürfnisse anzupassen [Jam03]. Adaptierbarkeit betrifft also die Anpassung basierend auf Wissen, das dem System vor der Aufnahme der Interaktion vorliegt oder von ihm erworben wird, und auch vor Beginn der Interaktion dazu angewendet wird. Adaptive Systeme erwerben dieses Wissen während der interaktiven Arbeitssitzung und wenden es auch an, während der Benutzer mit dem System interagiert [SPAS98]. Wie die Zuteilung der einzelnen Aktivitäten zu System oder Benutzer in Tabelle 2.1 zeigt, gibt es auch Mischformen.

Den Unterschied zwischen Adaptivität und Adaptierbarkeit illustrieren die Menüs in Microsoft Office ab der Version 2000. Diese sind adaptierbar; welche Menüeinträge gezeigt werden, kann also vom Benutzer konfiguriert werden. Zusätzlich gibt es die Funktion der *Smart Menus*, eine adaptive Alternative zum manuellen Konfigurieren der Menüs. Sie registrieren die Häufigkeit, mit der die einzelnen Menüpunkte vom Benutzer angewählt werden, und speichern diese im Benutzermodell. Darauf basierend zeigen sie nach einer gewissen Zeit nur mehr die häufigst verwendeten Einträge und erst bei Betätigen einer speziellen Schaltfläche das gesamte Menü.

### 2.1.1 Adaptive Hypermedia-Systeme

Adaptive Hypermedia-Systeme und adaptive Web-Systeme zählen zu den Benutzer-adaptiven Systemen [BM07]. Dazu gehören alle Hypertext- und Hypermedia-Systeme, die bestimmte Eigenschaften des Benutzers in seinem Benutzermodell abbilden und dieses dazu verwenden, gewisse sichtbare Aspekte des Systems an ihn anzupassen [Bru96b]. Beispiele für Inhalte im Benutzermodell sind Ziele, Präferenzen und das Wissen des Benutzers [Bru01].

Die Forschung an adaptiven Hypermedia-Systemen geht bis in die frühen 1990er-Jahre zurück [Bru01]. In dieser Zeit sah man sie als Erweiterung von klassischen Hypermedia-Systemen um intelligente Agenten, die den Benutzer bei der Arbeit unterstützen. Der Agent kann beispielsweise den Inhalt der Hypermedia-Seite an den Wissensstand und die Ziele des Benutzers anpassen und die relevantesten Links zur Weiterverfolgung anbieten. Die Möglichkeiten zur Adaption sind dabei relativ beschränkt auf den Inhalt (*content-level adaptation*) und die Links (*link-level adaptation*) [Bru96a].

Der Aufstieg des World Wide Web hatte einen entscheidenden Einfluss auf die Zahl und Typen von adaptiven Hypermedia-Systemen. Das WWW hat sich zur bevorzugten Umgebung für die Entwicklung derartiger Systeme entwickelt. Dies liegt unter anderem an dem Bedarf nach Adaptivität aufgrund der sehr unterschiedlichen Zielgruppen im Web. Bis 1996 beschäftigten sich noch die meisten Arbeiten mit klassischen Hypertext- und Hypermedia-Systemen. Nach 1996 lag der Fokus der Forschung auf web-basierten adaptiven Hypermedia-Systemen.

## 2.2 Benutzermodellierung

Die Grundlage für die Anpassungen eines adaptiven Systems an seine Benutzer bilden Modelle über diese, so genannte Benutzermodelle. In diesen Modellen sind *relevante Aspekte der Benutzer abgebildet*. Die konkreten Daten dazu erhält das System entweder implizit, beispielsweise durch das Überwachen der Benutzerinteraktionen, oder explizit durch direktes Befragen der Benutzer [BM07].

### 2.2.1 Benutzeridentifizierung

Eine Grundvoraussetzung für die Benutzermodellierung ist, dass Benutzer eindeutig identifiziert werden können. Gauch et al. [GSCM07] nennen hierzu für (web-basierte) adaptive Systeme fünf Möglichkeiten:

- *Software Agenten*: ein kleines Programm auf dem Rechner des Benutzers sammelt Informationen und sendet diese an einen Server. Dies ist sehr zuverlässig, bedeutet aber Aufwand für den Benutzer, weil er das Programm installieren muss.

- *Anmeldung*: der Benutzer identifiziert sich durch einen Anmeldevorgang am System. Diese Methode ist verlässlich und erlaubt den Zugriff von mehreren Rechnern aus. Aufwändig für den Benutzer können lediglich die Registrierung und das An- bzw. Abmelden bei der Systembenutzung sein.

- *Spezielle Proxy-Server*: der Benutzer registriert den Rechner, von dem aus er das System nutzt, bei einem Proxy-Server, welcher so den Benutzer identifiziert. Das Verfahren ist ziemlich genau, problematisch könnten nur der Registrierungsschritt sein, sowie dass alle verwendeten Rechner beim selben Proxy-Server registriert werden müssen.

- *Cookies*: die Benutzeridentifikation wird in einem Cookie am Rechner des Benutzers gespeichert und bei Anfragen an das System automatisch mitgesendet. Diese Methode ist für den Benutzer transparent und ermöglicht die Identifikation über mehrere Arbeitssitzungen. Beim Löschen der Cookies geht die Information jedoch verloren.

- *Session IDs*: der Benutzer wird innerhalb einer Arbeitssitzung identifiziert. Die Identifizierung über mehrere Sitzungen ist aber nicht möglich.

Auch über die Anwendung von Data Mining-Methoden auf Log-Dateien des adaptiven Systems können Benutzer identifiziert und nachverfolgt werden. Als guten Kompromiss sehen Gauch et al. die Verwendung von Cookies und optional eine Anmeldemöglichkeit für Benutzer [GSCM07].

### 2.2.2 Benutzermodelle

Brusilovsky und Millan [BM07] differenzieren bei den Benutzermodellen auf drei Ebenen:

- *Art*: was wird modelliert.

- *Struktur*: wie wird die Information abgebildet.

- *Methode*: wie werden die unterschiedlichen Arten von Modellen erstellt und aktualisiert.

### Modellart

Als die fünf gängigsten und nützlichsten Merkmale, die für einen einzelnen Benutzer modelliert werden können, sehen Brusilovsky und Millan [BM07]:

- *Wissen*: das Wissen des Benutzers über das in der Anwendung zum Tragen kommende Gebiet. Es wird vor allem bei adaptiven Lernsystemen und adaptiven Hypermedia-Systemen modelliert. Es liegt in der Natur dieses Merkmals, dass es variabel ist und im Modell beständig aktualisiert werden muss.
- *Interesse*: die Interessensgebiete des Benutzers. Dieses Merkmal ist das wichtigste (und oft einzige) in Benutzermodellen adaptiver Such- und Filtersysteme für große Informationsmengen. Auch Systeme, die automatisch generierte Empfehlungen abgeben (recommender systems), modellieren das Interesse des Benutzers.
- *Ziele und Tätigkeiten*: sie modellieren den unmittelbaren Zweck der Arbeit eines Benutzers in dem adaptiven System. Dies kann, abhängig vom genauen Systemtyp, ein Arbeitsziel, ein Informationsbedürfnis oder ein Lernziel sein. Es ist dies das variabelste aller Merkmale und kann sich von Sitzung zu Sitzung ändern, teilweise auch innerhalb einer Arbeitssitzung. Die Erkennung ist schwierig und im Allgemeinen nicht sehr präzise.
- *Hintergrund*: dies ist ein Sammelbegriff für Merkmale, welche die bisherigen Erfahrungen des Benutzers außerhalb des Kernbereichs der Anwendung betreffen. Beispiele dafür sind sein Beruf, Arbeitserfahrung oder Sprachkenntnisse. Diese Merkmale sind über die Zeit hinweg relativ stabil.
- *Individuelle Wesenszüge*: die Eigenschaften, deren Summe den Benutzer als Individuum definieren. Beispiele dafür sind Charaktereigenschaften (z. B. introvertiert/extrovertiert), kognitive Stile der Informationsorganisation und -repräsentation (z. B. holistisch/-seriell), kognitive Faktoren (z. B. Größe des Kurzzeitgedächtnisses) oder Lernstile. Diese Merkmale sind sehr stabil und ändern sich gar nicht oder nur sehr langsam. Sie können durch spezielle psychologische Tests in Erfahrung gebracht werden.

### Modellstruktur

Die Ausprägungen der genannten Merkmale müssen im Modell entsprechend strukturiert abgespeichert werden. Brusilovsky und Millan [BM07] nennen drei gängige Strukturen zur Organisation eines Benutzermodells (siehe auch die Beispiele in Abbildung 2.2 auf der nächsten Seite):

- *Mengen- bzw. Vektor-Modell*: es besteht aus einer Menge von unabhängigen Merkmalsausprägungen und besitzt keine interne Struktur im engeren Sinn. Diese einfachste Modellform verfügt über keine Verbindungen zwischen den Konzepten. Aus diesem Grund lassen sich auch schlecht Aussagen über neue Merkmale auf Basis schon modellierter treffen.

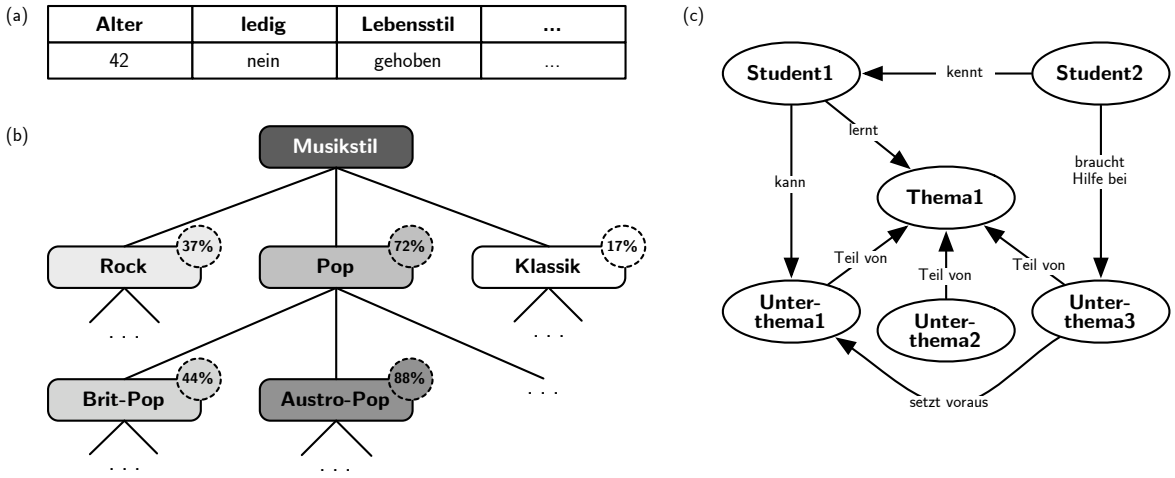


**Abbildung 2.2:** Modellstrukturen: a) Vektor-Modell mit demographischen Benutzerdaten, b) Taxonomie-Modell von Interessen, c) Ontologie-Modell im e-Learning

- *Taxonomie-Modell*: die Konzepte (hier oft Themen, Klassen oder Kategorien genannt) sind nach einem hierarchischen Klassifikationsschema strukturiert. Bevorzugt eingesetzt wird dieser Modelltyp in Systemen mit erweiterbaren Inhaltssammlungen wie beispielsweise Web-Verzeichnissen oder adaptiven Nachrichtensystemen. Durch die Verwendung von mehr als einer Klassifikationshierarchie können unterschiedliche Aspekte des Benutzers modelliert werden.

- *Netzwerk- bzw. Ontologie-Modell*: die Merkmalsausprägungen sind in einem Netzwerk strukturiert, verbunden durch unterschiedliche Arten von Kanten. Die Kanten modellieren die Semantik der Beziehungen zwischen den Konzepten, beispielsweise „ist ein“, „Teil von“, „ähnlich zu“. Verwendung finden aussagestarke Repräsentation wie Ontologien vor allem in Systemen mit abgeschlossenen Inhaltssammlungen, pädagogischen Anwendungen und Informationssystemen.

Sowohl Taxonomie-Modelle als auch Ontologie-Modelle basieren ihre Struktur oft auf schon bestehenden Modellierungen ihres Anwendungsgebiets. In einer Bibliothek könnte dies beispielsweise ein traditionelles Katalogisierungsschema sein, in einem Web-Shop eine Produktgruppenhierarchie oder in einem Informationssystem eine Ontologie über die Fachbegriffe und ihre Beziehungen untereinander. Die Merkmalsausprägungen eines Benutzers werden in seinem Benutzermodell in einem sogenannten *Overlay-Modell* gemäß der Struktur des darunterliegenden Anwendungsgebiet-Modells organisiert.

Overlay-Modelle sind eine weit verbreitete Form strukturierter Modelle. Sie modellieren für den jeweiligen Benutzer eine Untermenge der Merkmalsausprägungen des Anwendungsgebiet-Modells. Für die Modellierung des Wissens eines Benutzers können beispielsweise die Teilgebiete, die er beherrscht, nach der Struktur einer vollständigen Modellierung des Wissens durch Experten gespeichert werden. Für jeden Teil dieses Gesamtmodells enthält das Benutzermodell eine Angabe über das Ausmaß des Wissens in diesem Teilgebiet. In einem *Overlay-Modell*

*mit mehreren Schichten* kann für den jeweiligen Teil des Modells je nach Schicht ein anderer Wert gespeichert sein und es können somit mehrere Aspekte des Benutzers in der selben Struktur abgebildet werden [BM07].

#### Modellierungsmethode

Die konkreten Methoden zur Modellierung können auf mehreren Ebenen unterschieden werden. Auf der untersten Ebene stellt sich die Frage, wie die einzelnen Merkmalsausprägungen gespeichert werden. Brusilovsky und Millan [BM07] nennen hier einige Möglichkeiten:

- *Qualitative Modelle*: sie repräsentieren die konkreten Ausprägungen eines Merkmals mit Werten einer qualitativen Abstufung wie beispielsweise „hoch – mittel – niedrig“. Sie werden bevorzugt in regelbasierten adaptiven Systemen eingesetzt.
- *Numerische Modelle*: die Merkmalsausprägungen werden durch numerische Werte repräsentiert. Die Nutzungsdauer des Systems könnte zum Beispiel in Stunden gespeichert sein und so die Erfahrung des Benutzers mit dem System anzeigen.
- *Modelle mit Ungewissheitskomponente*: die Ungewissheit oder Ungenauigkeit bezüglich der exakten Ausprägung eines Merkmals wird mit Wahrscheinlichkeiten, Bayesschen Netzwerken oder Fuzzy-Logik ausgedrückt. So kann beispielsweise modelliert werden, dass ein Benutzer mit einer Wahrscheinlichkeit von 75% Interesse an einem bestimmten Sachgebiet hat.

Auf einer höheren Ebene unterscheiden sich die Modellierungsansätze laut Brusilovsky und Millan [BM07] in der Granularität der Unterscheidung zwischen einzelnen Benutzern:

- *Merkmalsbasierte Benutzermodellierung*: bestimmte Merkmale werden für einzelne Benutzer modelliert. Diese können sich ändern und müssen durch das System aktualisiert werden. Dies ist gegenwärtig die vorherrschende Form der Benutzermodellierung in adaptiven Web-Systemen.
- *Modellierung mit Stereotypen*: in der ältesten Methode der Benutzermodellierung werden Benutzer zu Gruppen mit stereotypischen Eigenschaften zusammengefasst. Ändern sich Merkmale eines Benutzers, so kann er einfach einem anderen Stereotypen zugeordnet werden. Eine Möglichkeit ist auch, dass Benutzer (mit unterschiedlichen Wahrscheinlichkeiten) mehr als einem Stereotyp zugeordnet sind.
- *Kombinationen*: die zwei Ansätze lassen sich kombinieren. Eine der beliebtesten Kombinationen ist die Verwendung von Stereotypen zur Initialisierung eines individuellen, merkmalsbasierten Benutzermodells. So lässt sich das Problem eines leeren Modells bei neuen Benutzern lösen.

Auf oberster Ebene gibt es ebenfalls noch Unterschiede bei den Methoden der Benutzermodellierung. Bei der *Wahl des Speicherorts für die Modelle* gibt es beispielsweise zwei grundlegende

Möglichkeiten: zentral auf einem Server oder bei jedem Benutzer individuell. Nicht jede dieser Möglichkeiten eignet sich für alle Typen von adaptiven Systemen.

Ebenso unterscheiden kann man die *Aktualisierungshäufigkeit* des Modells. Dieses kann „online“ beständig aktualisiert werden, oder aber auch „offline“ in einem separaten Schritt. Ein Nachteil von Offline-Modellen ist, dass sich das System während einer Arbeitssitzung nicht an Änderungen des Benutzers anpassen und ganz allgemein nur mittel- und langfristige Merkmale sinnvoll erfassen kann [KL05]. Ein Vorteil wäre, dass aufwändigere Methoden zur Informationsgewinnung und Modellierung (z. B. Data Mining in Logdateien) angewendet werden können, da dies im Gegensatz zu Online-Modellen nicht neben dem normalen Systembetrieb geschehen muss.

### 2.2.3 Informationen zum Benutzer sammeln

Bisher wurde nur besprochen, wie die Benutzermodellierung strukturiert sein soll, und welche Daten die Modelle enthalten können. Der wichtigste Schritt, das eigentliche Sammeln der Daten, fehlte noch. Jameson [Jam03] beschreibt Möglichkeiten, wie einerseits der Benutzer diese Informationen dem System zur Verfügung stellen und andererseits das System diese von sich aus akquirieren kann. Er trifft dabei die allgemein anerkannte Unterscheidung zwischen *expliziter* und *impliziter* Informationsgewinnung.

#### Explizite Selbstbeurteilung und Bewertungen

Der Benutzer wird durch das System aufgefordert, Informationen zu liefern. Problematisch kann dies sein, wenn es für den Benutzer einen großen geistigen oder körperlichen Aufwand darstellt, nicht direkt für die Anwendung relevant scheint oder zu sehr die Privatsphäre verletzt. Jameson unterscheidet:

- *Selbstbeurteilung zu objektiven, persönlichen Eigenschaften*: Alter, Beruf oder Wohnort können einen Einfluss auf die Adaption haben. Ein Vorteil ist, dass sich diese Informationen relativ selten ändern. Nachteilig ist der oft große Aufwand für den Benutzer (Optionen wählen, Text eingeben), und dass dieser seine Privatsphäre verletzt sieht. Beschränkung auf die notwendigsten Daten, Erklärungen zur Verwendung der Daten und spielerisch aufbereitete Abfragen können hier helfen.

- *Selbstbeurteilung zu allgemeinen Bereichen*: dies betrifft beispielsweise das Interesse an einem bestimmten Thema, das Wissen in einem Fachbereich oder die Wichtigkeit eines gewissen Kriteriums. Ber Benutzer kann auf Bewertungsskalen oder durch An- bzw. Abwählen eines Begriffs antworten. Problematisch sind unklare Bedeutungen der einzelnen Bewertungsstufen und der eher kognitive denn körperliche Aufwand beim Beantworten. Verzerrungen sind möglich, wenn Benutzer so antworten, wie sie es für gesellschaftlich akzeptiert halten.

- *Selbstbeurteilung durch spezifische Bewertungen*: der Benutzer hat die Möglichkeit, bestimmte Objekte, die er beispielsweise gerade sieht, gerade durchgeführte Aktionen oder Dinge auf Basis ihres Namens oder von Beschreibungen zu bewerten. Möglich ist dies durch Betätigen von Symbolen (z. B. Daumen nach oben/unten), An- bzw. Abwählen oder mit Bewertungsskalen. Der Aufwand hängt davon ab, wie geistig präsent das zu bewertende Objekt ist. Da für die Benutzer die Bewertungen oft nicht direkt Teil der aktuellen Aufgabe ist, kann es besser sein, sie durch implizite Aktionen (Auswahl/Überspringen eines Objekts in einer Liste) anstatt explizit zu ermitteln.
- *Antworten auf Testfragen*: der Benutzer antwortet auf explizite Fragen, beispielsweise um sein Wissen einschätzen zu lassen. Außer bei Lernanwendungen sind Benutzer zurückhaltend bei diesen Fragen, es sei denn diese werden ansprechend und spielerisch präsentiert.

#### Nicht-explizite Eingaben

Der Benutzer wird nicht direkt durch das System um Eingaben ersucht, sondern dieses versucht Informationen aus seinen Aktionen, schon vorhandenen Daten, körperlichen Merkmalen und Signalen zur Umgebung zu gewinnen. Mögliche Quellen hierfür sind:

- *Natürlich auftretende Aktionen*: Ereignisse der Interaktion des Benutzers mit dem System wie beispielsweise der Kauf eines Gegenstandes oder das Scrollen einer Seite werden beobachtet. Der Benutzer wird dadurch nicht gestört, doch seine Privatsphäre kann beeinträchtigt sein. Die Interpretation der Aktionen ist schwierig, dennoch ist dies die wichtigste Quelle impliziter Information.
- *Bestehende Informationen*: es ist möglich, dass aufgrund früherer Kontakte beim Betreiber des Systems schon Daten zum Benutzer vorliegen. Auch in öffentlichen Quellen und online (z. B. auf der persönlichen Homepage) lassen sich Informationen finden. In speziellen Servern könnte darüber hinaus auch schon ein Benutzermodell vorliegen.
- *Psychologische Indikatoren*: Wut oder Frustration, Attraktivität von bestimmten Objekten, sowie Stress und kognitive Belastung können gemessen werden. Sensoren am Körper des Benutzers können physiologische Werte wie Blutdruck, Leitfähigkeit der Haut oder Atmung messen. Anhand von Audio- und Videoaufzeichnungen kann man Gesichtsausdrücke und sprachliche Artikulation analysieren. Die Auswertung erfordert Techniken der Mustererkennung, und oft müssen die Daten mit anderen Informationen kombiniert werden. Ein Vorteil ist der ständige Datenstrom, nachteilig kann körperliches und gesellschaftliches Unbehagen beim Benutzer aufgrund der Sensoren sein.
- *Signale zur Umgebung*: bei mobilen Anwendungen können genaue Informationen zur Position des Benutzers (z. B. über das Global Positioning System (GPS)) oder Bilder und Geräusche helfen, die Umgebung zu analysieren und entsprechende Adaptionen vorzunehmen. Bei einem hohen Geräuschpegel könnten beispielsweise automatisch Warntöne mit größerer Lautstärke ausgegeben werden.

Wichtig ist auch, auf die *Unsicherheiten* bei den gewonnen Informationen entsprechend zu reagieren. Sich widersprechende Informationen müssen durch spezielle Strategien zur Konfliktauflösung bewertet werden. Darüber hinaus ist selbst bei direkten Fragen an die Benutzer nicht sichergestellt, dass die Informationen konsistent und nicht verzerrt sind. Psychologische Studien haben gezeigt, dass Menschen keine zuverlässigen Quellen von Informationen über sich selbst sind [DMKSH93].

## 2.3 Usability Herausforderungen

Durch ihre enge Verknüpfung mit dem Feld der Mensch-Maschine-Kommunikation wird gerade der Gebrauchstauglichkeit (*usability*) große Aufmerksamkeit geschenkt. Jameson [Jam03] nennt einige Eigenschaften, die teils allgemein erstrebenswert für interaktive Systeme, teils speziell für Benutzer-adaptive Systeme relevant sind:

- *Berechenbarkeit* (predictability): wie berechenbar arbeitet das Systen, und wie gut kann der Benutzer die Auswirkungen seiner Interaktionen vorhersehen.
- *Verständlichkeit* (transparency): wie gut ist das System verständlich, und wie klar ist das Bild des Benutzers darüber, wie das System arbeitet.
- *Steuerbarkeit* (controllability): wie gut kann der Benutzer gewisse Aktionen oder Zustände im System hervorrufen. Obwohl nicht hundertprozentig korreliert, profitiert die Steuerbarkeit von guter Berechenbarkeit und Verständlichkeit.
- *Unaufdringlichkeit* (unobtrusiveness): wie stark beansprucht das System die Aufmerksamkeit des Benutzers und lenkt ihn dadurch von seinem primären Ziel ab.
- *Privatheit* (privacy): wie stark wird durch die Sammlung und Verwertung von (persönlichen) Daten die Privatsphäre des Benutzers verletzt. Dieses Merkmal ist besonders relevant bei adaptiven System und hier insbesondere im Bereich e-Commerce und Kollaborationsunterstützung.
- *Erlebnisreichtum* (breadth of experience): wie stark vermindert das System durch die Adaption das „Erlebnis“ des Benutzers. Werden beispielsweise bestimmte Dokumente, Produkte oder Menschen automatisch vom System ausgewählt, so lernt der Benutzer weniger über das jeweilige Gebiet. Auch ein unvollständiges Benutzermodell kann das Benutzererlebnis einschränken, wenn das System damit Adaptionen vornimmt, die nicht zu diesem Benutzer passen.

Die Überprüfung dieser Kriterien kann wertvolle Hinweise zur Gebrauchstauglichkeit eines adaptiven Systems geben. In der Evaluierung der hier beschriebenen Version von Prospector wurden die Benutzer nach ihren diesbezüglichen Einschätzungen befragt. Details dazu finden sich in Kapitel 7.

## 2.4 Anwendungen

Adaptive Systeme finden in vielen Gebieten Anwendung. Speziell für den Bereich adaptiver Hypermedia-Systeme nennt Brusilovsky [Bru96a] fünf Hauptanwendungsgebiete:

- *e-Learning*: die Lernenden werden beim Bearbeiten und Erlernen der Materialien unterstützt. Das System kann sich beispielsweise an unterschiedliche Wissensstände anpassen und einen Weg durch die Unterlagen weisen. Auch bei der Kollaboration zwischen Lernenden kann das System helfend eingreifen.
- *Online Informationssysteme*: dieser Bereich umfasst Systeme mit Online-Dokumentationen bis hin zu kompletten Enzyklopaedien. Das System kann beim Auffinden, Navigieren und Verarbeiten der gebotenen Informationen unterstützen und nimmt beispielsweise auf unterschiedliche Vorkenntnisse und Ziele Rücksicht.
- *Online Hilfesysteme*: derartige Systeme sind ähnlich zu Informationssystemen, aber im Gegensatz zu diesen nicht eigenständig sondern Teil einer Anwendung wie beispielsweise einer Tabellenkalkulation oder einer Entwicklungsumgebung. Informationen aus der Nutzung der Anwendung können hier beispielsweise helfen.
- *Information Retrieval-Systeme*: ein derartiges System kann beim Durchsuchen einer Dokumentensammlung mit geeigneten Verlinkungen helfen, relevante Dokumente zu finden, oder den Suchprozess selbst an den Benutzer anpassen. Information Retrieval muss dabei umfassend verstanden werden, und darf nicht auf die Suche, wie man sie von Web-Suchmaschinen kennt, reduziert werden. Dieses Teilgebiet nimmt jedoch einen großen Stellenwert ein und wird im Teil zu adaptiver Suche in Kapitel 3 näher behandelt.

Ein wichtiger Bereich, der in dieser Auflistung fehlt, weil er zu dieser Zeit (1996) noch wenig relevant war, ist *e-Business*. Das wohl bekannteste adaptive System ist hierbei der Web-Shop von Amazon[1]. Dieser identifiziert auf Basis der Suchanfragen und der betrachteten oder gekauften Produkte die Interessen des Benutzers und präsentiert selbständig Vorschläge für andere möglichlicherweise passende Produkte.

In der Einteilung von Jameson [Jam03] zu den Funktionen Benutzer-adaptiver Systeme findet sich diese Anwendung unter „Produkte empfehlen“. Die einzelnen Funktionen unterscheidet er in Unterstützung bei der Systembenutzung und Unterstützung bei der Informationsgewinnung. Beide Ausprägungen werden in der Folge näher beschrieben.

### 2.4.1 Unterstützung bei der Systembenutzung

Ein mögliches Einsatzgebite für Adaptivität ist, den Benutzer dabei zu unterstützen, ein System erfolgreich und effektiv zu nutzen. Jameson [Jam03] nennt vier Beispiele für eine derartige Unterstützung:

[1] `http://www.amazon.com`

- *Routineaufgaben teilweise übernehmen*: Tätigkeiten, die zwar zeitaufwändig sind, aber wenig Intelligenz oder Wissen vom Anwender verlangen, können vom System übernommen werden und so Zeit und Aufwand einsparen. Am Besten funktioniert dies, wenn das System komplett eigenständig handelt, doch meist wird der Benutzer noch in bestimmte Entscheidungen eingebunden.

- *Benutzerschnittstelle anpassen*: das System kann seine Benutzerschnittstelle so anpassen, dass die besser zur Arbeitsweise des Benutzers passt. Auch Menschen mit physischen Beeinträchtigungen und Wahrnehmungstörungen können so unterstützt werden. Die Oberfläche darf sich aber nicht zu oft ändern, da ansonsten die insgesamte Bedienbarkeit leidet.

- *Hinweise zur Systembenutzung geben*: anstelle einer Anpassung der Benutzerschnittstelle kann das System beispielsweise auf Basis kürzlich vom Benutzer getätigter Aktivitäten Informationen und Hinweise zur Bedienung anbieten. Schwierig ist hierbei, die Absichten des Benutzers zu erkennen und nicht die Benutzerfreundlichkeit durch zu aufdringliche Meldungen negativ zu beeinflussen.

- *Dialoge steuern*: neben dem Anbieten von Hilfe und Produktempfehlungen kann ein System auch seine Strategie für das Führen natürlichsprachlicher Dialoge anpassen. Wann und wie Informationen angeboten oder vom Benutzer erfragt werden, lässt sich so beispielsweise auf dessen sprachlichen und geistigen Fähigkeiten und die Qualität der Verständigung abstimmen.

### 2.4.2 Unterstützung bei der Informationsgewinnung

Als zweiten großen Bereich von Funktionen Benutzer-adaptiver Systeme sieht Jameson [Jam03], die Benutzer dabei zu unterstützen, das was sie suchen in einer Form zu finden, in der sie etwas damit anfangen können. Mögliche Anwendungen dabei sind:

- *Hilfe beim Finden von Informationen*: Ziel ist das Finden relevanter elektronischer Dokumente vom kurzen Nachrichtenartikel bis zum komplexen Multimediaobjekt. Dies ist auf drei Arten möglich:

  - *Durchsuchen unterstützen*: auf Basis der Informationsbedürfnisse des Benutzers vielversprechende Dokumente oder Suchrichtungen vorschlagen oder hervorheben.

  - *Suchabfragen und Filtern unterstützen*: die Auswahl der gefundenen Dokumente und deren Präsentation (z. B. Reihung) an den Benutzer anpassen.

  - *Spontan Informationen liefern*: für die aktuelle Situation relevante Informationen anbieten. Besonders wichtig ist hier, nicht zu aufdringlich zu sein.

- *Präsentation von Informationen anpassen*: das Interesse und Wissen des Benutzers in einem bestimmten Gebiet, seine Vorlieben für eine bestimmte Form der Präsentation

(z. B. bei Menschen mit Behinderungen) und die Darstellungsfähigkeiten seines Endgeräts können hier berücksichtigt werden.

- *Produkte empfehlen*: das System kann selbständig Empfehlungen aussprechen, selbst wenn der Benutzer nicht weiß, welche Aspekte generell oder ihm persönlich dabei wichtig sind. Es kann dabei eine viel größere Menge an Produkten nach Eignung durchsuchen als der Anwender. Techniken des *Collaborative Filtering* liefern so für einzelne Benutzer sehr genaue Empfehlungen.
- *Kollaboration unterstützen*: spontane Kollaboration ohne vorheriges Kennenlernen kann das System dadurch unterstützen, dass es Teilnehmer zusammenführt, die in Bezug auf Wissen, Interessen oder Ziele zu einander passen oder einander ergänzen.
- *Lernen unterstützen*: unter Berücksichtigung von Fachwissen, bevorzugtem Lernstil, Motivation und Fortschritt kann das System einen Benutzer durch Kurse und Lernmaterialien führen, Inhalte anpassen und zu den richtigen Zeiten passende Hinweise und Rückmeldungen geben.

# Kapitel 3

# Adaptive Suche

Adaptive Suche ist ein konkreter Anwendungsbereich der in Kapitel 2 behandelten adaptiven Systeme. In diesem Kapitel werden die Grundlagen, Methoden und Herausforderungen adaptiver Suche beschrieben. Im Detail wird dabei auf die Möglichkeiten der Benutzermodellierung und der Personalisierung eingegangen. Abschließend werden einige repräsentative adaptive Suchsysteme kurz vorgestellt.

## 3.1 Grundlagen

In der Einleitung in Kapitel 1 wurden einige Probleme bisheriger und Herausforderungen für neuartige Suchmaschinen erläutert. Um der gewaltigen Datenmenge im World Wide Web Herr zu werden, muss es unter anderem möglich sein, seine Informationsbedürfnisse besser zu spezifizieren. Mehrdeutigkeiten natürlicher Sprachen sollen in Suchanfragen berücksichtigt und aufgelöst werden. Die Eigenschaften des Suchenden, seine Interessen und Präferenzen sollen als zusätzliche Information in die Suche einfließen. Semantische Informationen zu den gesuchten Dokumenten müssten ebenso besser genützt werden wie auch Bewertungen aus der Anwendergemeinde. Die Ergebnisse sollen in ansprechender und leicht verständlicher Form dem Benutzer präsentiert werden.

Adaptive Suchsysteme bieten *Lösungsansätze* für die genannten Probleme. Anders als traditionelle Suchmaschinen berücksichtigen sie nicht nur die Suchbegriffe der aktuellen Anfrage sondern auch die Informationen in einem Modell der langfristigen und/oder kurzfristigen Interessen und Präferenzen des Benutzers [Bru01]. Sie liefern dem Benutzer eine auf dieses Modell oder den aktuellen Kontext individuell abgestimmte Auswahl von Ergebnissen. Auch Hintergrund und Wissensstand des Benutzers können in die Abfrage einfließen [MGSG07].

Keenoy und Levene [KL05] nennen einige grundlegende *Unterscheidungsmerkmale* für personalisierte Suchsysteme, die auch teilweise allgemein für adaptive Systeme gültig sind. Die unterschiedlichen Ausprägungen der meisten dieser Merkmale werden in diesem Kapitel erläutert:

- *Sammelmethode für Daten über den Benutzer*: Werden die Daten explizit vom Benutzer erfragt oder implizit aus seiner natürlichen Interaktion mit dem System abgeleitet?

- *Modellspeicherung*: Ist das Modell auf der Client-Seite am Rechner des Benutzers oder am Server gespeichert?
- *Adaptivität*: Passt sich das System im Laufe der Zeit automatisch an den Benutzer an? Wenn nicht, kann der Benutzer das Verhalten des Systems manuell beeinflussen, beispielsweise durch das Aktualisieren eines statischen Modells?
- *Modellerstellung*: Wird das Benutzermodell online oder offline erstellt und aktualisiert?
- *Modellinhalt*: Was wird genau im Modell gespeichert?
- *Personalisierungsmethode*: Reiht das System die Ergebnisse einer anderen Suchmaschine um oder filtert sie, sendet es eine angepasste Anfrage an einen anderen Suchdienst oder macht es eine eigene personalisierte Suche?
- *Algorithmen*: Welche Algorithmen werden verwendet, um ein personalisiertes Ergebnis zu erzeugen?
- *Benutzeroberfläche*: Wie werden die personalisierten Ergebnisse dargestellt?

## 3.2 Benutzermodellierung

Benutzermodelle bilden die Grundlage der Adaptionen in personalisierten Suchsystemen. Insbesondere im Kontext von Information Retrieval wurden und werden diese auch Benutzerprofile genannt. Sie repräsentieren zumeist die Interessen des Benutzers in der Form von einzelnen Begriffen oder ganzen Konzepten [BM07].

Benutzermodelle, die verändert und erweitert werden können, nennt man dynamisch. *Statische Modelle* hingegen enthalten über die Zeit ihrer Nutzung hinweg die selben Informationen. *Dynamische Modelle* können auch eine Zeitkomponente berücksichtigen und somit zwischen kurzfristigen, aktuellen Informationen zum Benutzer (z. B. seine Interessen) und längerfristigen, sich selten ändernden unterscheiden [GSCM07].

Da es sich bei adaptiven Suchsystemen um eine Untergruppe adaptiver Systeme handelt, gelten auch hier die selben grundlegenden Merkmale zur Klassifizierung von Aufbau und Funktion. Aus diesem Grund ist dieser Abschnitt ähnlich strukturiert wie Abschnitt 2.2 „Benutzermodellierung“.

### 3.2.1 Modellarten

In 2.2.2 wurden als gängigste Merkmale, die zu einem Benutzer modelliert werden, Wissen, Interesse, Ziele und Tätigkeiten, Hintergrund und individuelle Wesenszüge genannt. Bei der adaptiven Suche ist zumeist das *Interesse des Benutzers* an bestimmten Sachgebieten und Themen und die Dauer dieses Interesses Inhalt des Benutzermodells [GSCM07]. Es wird teilweise auch als der wichtigste und oft einzige Teil des Benutzermodells in adaptiven Such- und Filtersystemen für große Informationsmengen angesehen [BM07].

Adaptive Suchsysteme nützen aber durchaus auch Informationen zu anderen Aspekten der Benutzer. Deren Wissen, Ziele und aktuellen Aktionen, Hintergründe und individuelle Wesenszüge können ebenfalls Teil des Benutzermodells werden. Micarelli et al. [MGSG07] unterscheiden bei dessen Inhalt zwischen zwei *Arten von Daten*:

- *Benutzerdaten*: Hintergrundinformationen über den Benutzer wie demographische Daten, Bildungsniveau, Erfahrung mit einem Interessensgebiet, aktueller Kontext
- *Nutzungsdaten*: Informationen über das Verhalten des Benutzers bei der Interaktion mit dem System, z. B. Häufigkeit der Verwendung, Navigation, Suchanfragen, Bewertungen

### 3.2.2 Modellstrukturen

Im Gebiet der adaptiven Suche werden alle in 2.2.2 bereits näher beschriebenen Strukturen für Benutzermodelle eingesetzt: Mengen- bzw. Vektor-Modelle, Taxonomie-Modelle und Netzwerk- bzw. Ontologie-Modell. Im Folgenden werden die einzelnen Ansätze und ihre spezielle Anwendung in adaptiven Suchsystemen vorgestellt.

#### Mengen- bzw. Vektor-Modell

Dieser Ansatz wurde von beinahe allen adaptiven Such- und Filtersystemen benutzt und ist auch heute noch der bevorzugte. Die einzelnen Element der Menge bzw. des Vektors entsprechen dabei einzelnen Schlüsselwörtern [BM07]. Diese *Schlüsselwörter* wurden entweder automatisch aus den gefundenen Dokumenten extrahiert oder vom Benutzer angegeben. Üblicherweise ist für jedes Wort auch ein Gewicht gespeichert. Dieses entspricht dem Interesse des Benutzers an diesem Wort, wird also durch dessen Feedback (z. B. positive oder negative Bewertungen) verändert [GSCM07].

Bei der Suche werden zu den einzelnen gefundenen Dokumenten die jeweiligen Schlüsselwörter bestimmt. Diese können beispielsweise direkt aus dem Dokument oder aus einem im Suchindex gespeicherten Abriss des Inhalts gewonnen werden. Die Schlüsselwörter des so entstehenden Vektors werden mit Standardverfahren des Information Retrieval wie beispielsweise Term Frequency-Inverse Document Frequency (TF/IDF) gewichtet. Jeder dieser Vektoren wird anschließend mit dem Benutzermodell verglichen, etwa mit dem Cosinus Ähnlichkeitsmaß [GSCM07].

Gauch et al. [GSCM07] nennen auch einen Ansatz, der nicht mit einzelnen Wörtern sondern mit ganzen *Wortsequenzen* arbeitet. Damit sollen Zweideutigkeiten von Wörtern (siehe Abschnitt 1.1) durch den Kontext ihrer Verwendung besser aufgelöst werden. Wortpaare sind laut Untersuchungen dieses Verfahrens genauer als einzelne Schlüsselwörter.

Der Vorteil von Mengen- bzw. Vektor-Modellen mit Schlüsselwörtern ist, dass sie am einfachsten zu implementieren und erstellen sind. Nachteilig ist jedoch die große Menge an Benutzerfeedback, die für ihre Erstellung nötig ist [GSCM07].

### Taxonomie-Modell

Taxonomie-Modelle bilden abstrakte Themen und deren Beziehungen untereinander ab. Durch eine hierarchische Strukturierung ist es auch möglich, Generalisierungen zu modellieren und abzuleiten. Als Quelle für die verwendeten Themen und ihre Beziehungen dienen oft Referenztaxonomien und Thesauri [GSCM07]. Der Inhalt des Benutzermodells ist ein gewichtetes Overlay über diese Referenzstrukturen, die ein umfassendes Modell des Anwendungsgebiets bilden. Im Allgemeinen sind die so entstehenden Benutzermodelle mächtiger als jene mit Schlüsselwörtern und erlauben eine präzisere Abildung der Interessen des Benutzers [BM07].

Mit einem umfangreichen Modell des Anwendungsgebiets können auch unterschiedliche Aspekte der Benutzerinteressen separat modeliert werden. Über semantische Information zu Beziehungen zwischen den Themen im Modell des Anwendungsgebiets ist es darüber hinaus möglich, Interessensinformationen auch beispielsweise auf verwandte Bereiche zu schon im Benutzermodell verzeichneten auszuweiten. Damit kann man der dünnen „Besiedelung“ (*sparsity*), einem Standardproblem großer Overlay-Modelle entgegenwirken [BM07].

Früher war es nur in Systemen mit *geschlossenen Dokumentsammlungen* (z. B. Bibliotheken, Akten) möglich, diese Art von Modellstruktur einzusetzen. Durch manuelle Indizierung wurden die einzelnen Dokumente im Modell des Anwendungsgebiets den einzelnen Themen zugeordnet. Systeme mit *offenen Dokumentsammlungen* (z. B. Internetsuchmaschinen) mussten auf Ansätze mit Schlüsselwörtern zurückgreifen. In neueren Systemen ist es jedoch möglich, mittels Textklassifizierung gefundene Dokumente automatisch einem Thema des Anwendungsbebiets zuzuordnen [BM07] [GSCM07].

Das Erstellen einer umfangreichen Hierarchie von Themen ist aufwändig und muss meist manuell durchgeführt werden. Aus diesem Grund werden üblicherweise Untermengen schon vorhandener Themenhierarchien verwendet. Zur Verwendung gekommen sind laut Gauch et al. [GSCM07] unter anderem schon die Sensus Ontologie[1], das Yahoo! Directory[2] und das Open Directory Project (ODP) [ODP08].

Um derartig große *Referenztaxonomien und -ontologien* nützen zu können, müssen sie zuerst *bereinigt* werden. Jene Knoten, die keinen Themen im engeren Sinn entsprechen sondern nur der Gliederung dienen, müssen entfernt werden. Werden schon klassifizierte Dokumente wie beispielsweise die verlinkten Webseiten im ODP als Trainingsdokumente für den Klassifizierungsalgorithmus verwendet, so werden davon pro Thema eine gewisse Zahl benötigt. Zu kleine Themen mit zu wenigen Dokumenten müssen daher in ihr übergeordnetes eingegliedert werden [GSCM07].

Bei der Verwendung umfangreicher Strukturen wie beispielswese der des ODP kann das Overlay im Benutzermodell sehr groß und feingliedrig werden und viele spezialsierte Themen enthalten. Beim Bewerten von Suchergebnissen auf der Basis der Informationen im Modell muss daher eventuell auf Informationen in höheren Ebenen der Themenhierarchie zurückgegriffen werden oder eine Möglichkeit gefunden werden, deren Gewichtung abzuleiten [GSCM07].

[1] `http://www.isi.edu/natural-language/projects/ONTOLOGIES.html`
[2] `http://dir.yahoo.com`

#### Netzwerk- bzw. Ontologie-Modell

Noch komplexere Modelle als mit Taxonomien kann man semantischen Netzwerken oder auf Basis von echten Ontologien erstellen. Selbst wenn einige Systeme von Ontologien sprechen, so arbeiten sie im engeren Sinn nur mit Themenhierarchien. Mit den zusätzlichen semantischen Informationen in Ontologien können die Suchergebnisse verbessert werden.

Der Vorteil gegenüber Modellen mit Schlüsselwörtern ist, dass Beziehungen zwischen bestimmten Begriffen und Konzepten auf höherer Ebene modelliert werden können. Ein Nachteil von semantischen Netzen ist, dass sie die Terminologie der einzelnen Konzepte erst durch ausreichend Benutzerfeedback erlernen müssen. Außerdem haben die Ansätze noch Probleme mit der Skalierbarkeit [GSCM07].

### 3.2.3 Modellierungsmethoden

In 2.2.2 kamen bereits die allgemeinen Modellierungsmethoden zur Sprache. Einige davon werden nun für die adaptive Suche konkretisiert. Darüber hinaus gibt es diesem Bereich noch ein paar erwähnenswerte Unterscheidungen von möglichen Methoden zur Modellierung, welche ebenfalls erläutert werden.

Modelle können manuell durch Benutzer oder Experten erstellt werden. Dies ist jedoch schwierig und zeitaufwändig. *Automatische Methoden zum Erstellen und Aktualisieren* sind daher sehr viel beliebter und darüber hinaus weniger aufdringlich für den Benutzer. Vor vollständig automatisierter Modellerstellung wird aber gewarnt und statt dessen empfohlen, wenigstens Benutzerfeedback mit minimalem Aufwand zu gewinnen [GSCM07].

Da in den Modellen für adaptive Suchsysteme zumeist Benutzerinteressen gespeichert sind, sind auch kaum Ansätze mit Stereotypen zu finden. Anand und Mobasher [AM05] nennen jedoch als einen möglichen Nachteil, dass so die Chance des zufälligen Entdeckens relevanter, aber nicht notwendigerweise zum eigenen Interensprofil passender Informationen verloren geht. Einer zu starken Fokussierung auf das eigene Modell kann mit Gruppenmodellen entgegengewirkt werden.

Beachtet werden müssen auch das *Latenzproblem* und das des „Cold Start“. Ersteres tritt auf, wenn ein Benutzer, noch bevor das System die nötigen Daten für eine sinnvolle Personalisierung sammeln konnte, die Benutzung wieder aufgibt. Dieses Problem kann auch für neue ins System gebrachte Objekte (Dokumente, Produkte, ...) entstehen und so verhindern, dass diese gefunden werden. Das *„Cold Start“ Problem* kann nach der Installation des Systems auftreten, wenn dieses noch keinerlei Informationen enthält [AM05]. In allen drei Fällen können sinnvolle Startwerte helfen, beispielsweise durch Gruppenmodelle oder Initialisierungen aus stereotypischen Modellen.

Zuvor schon kurz erwähnt wurde auch das Problem der *sparsity*, das bei Suchsystemen vermehrt auftritt. Aufgrund der großen Zahl an Produkten / Dokumenten, Schlüsselwörtern und

Themen in Taxonomien und Ontologien ist das Benutzermodell im Verhältnis zu seiner möglichen Größe auch nach viel Benutzerfeedback nur zu einem kleinen Teil gefüllt. Für neue Suchen könnten daher die nötigen Informationen fehlen [AM05]. Durch das *automatische Ableiten von Werten* im Benutzermodell aus schon vorhanden und ein dadurch erreichtes Vergrößern der Datenbasis (z. B. durch eine „interconcept interest propagation“) kann diesem Problem begegnet werden [BM07].

### 3.2.4 Informationen zum Benutzer sammeln

Eine der wichtigsten Aufgaben, um effektive Personalisierung bieten zu können, ist das Sammeln von Daten über den Benutzer und seine Präferenzen. Anhand von Benutzerfeedback können diese Informationen für die Modellierung gewonnen werden. Wie in Unterabschnitt 2.2.3 erwähnt, ist das Feedback auch bei der adaptiven Suche auf zwei Arten möglich: explizit oder implizit [MGSG07].

Implizites Sammeln von Informationen beeinträchtigt den Benutzer dabei nur wenig und hat daher bessere Chancen, auch genutzt zu werden. Studien haben außerdem gezeigt, dass diese Methode ähnlich gut funktioniert wie explizite Informationssammlung. Andere Ergebnisse deuten darauf hin, dass eine Kombination der beiden Methoden noch bessere Ergebnisse liefert. Ein Ansatz zum Kombinieren ist hierbei, das explizite Feedback stärker zu gewichten [GSCM07].

#### Explizite Informationen

Die in 2.2.3 beschriebenen, allgemeinen Kategorien expliziter Information finden mit Ausnahme von „Antworten auf Testfragen“ auch bei der adaptiven Suche ihre Anwendung. Benutzer können ihre Interessen, Intentionen und Information über sich bei der Registrierung oder während der Nutzung des Systems angeben. Gängige Methoden dazu sind Fragebögen und Bewertungen [MGSG07]. Spezifische Bewertungen, insbesondere der vom Suchsystem zurückgegebenen Ergebnisse, können von den Benutzern ebenfalls explizit vorgenommen werden [GSCM07]. Doch Studien zeigen, dass ohne greifbaren Nutzen für die Benutzer, diese mehr Ergebnisse betrachen als sie schlussendlich bewerten [AM05].

Alle expliziten Methoden der Informationsgewinnung haben den Nachteil, dass sie die Zeit des Benutzers in Anspruch nehmen und er bereit sein muss, mitzumachen. Aus diesem Grund, und auch aufgrund von Bedenken bezüglich ihrer Privatsphäre, können Benutzer es ablehnen, Informationen zu liefern. Ihre Angaben können auch ungenau sein und, sollte ihr Modell nicht regelmäßig aktualisiert werden, mit der Zeit ungenau werden. Manchmal können Benutzer aber durchaus Vergnügen daran finden, Feedback zu geben, beispielsweise auf Seiten für Film-, Musik- oder Buchkritiken [GSCM07].

### Implizite Informationen

Viele adaptive Suchsysteme verwenden implizite Informationen über den Benutzer durch die in 2.2.3 genannten „Natürlich auftretende Aktionen“. Hierbei wird das *Verhalten des Benutzers* beobachtet und mitverfolgt, ohne dass dieser bewusst sein Feedback gibt. Nutzungsdaten können dazu auf der Serverseite (Logs, Anfragen, Navigationspfade) und/oder am Client (Browserverlauf, Interaktionen wie Klicken oder Scrollen, ...) gesammelt werden [MGSG07].

Gauch et al. [GSCM07] nennen konkrete Quellen und ihre Vor- bzw. Nachteile, wobei die in allen Fällen mehr oder weniger stark gegebenen Risiken für die Privatsphäre nicht explizit erwähnt werden:

- *Browser Cache*: der Verlauf der besuchten Seiten kann mit ihm ermittelt werden. Dieser ist eine häufig genutzte Quelle von Information über die Interessen des Benutzers. Der Benutzer muss dazu nichts installieren, aber regelmäßig den Cache zur Analyse hochladen.
- *Proxy Server*: dieser kann die Internet-Aktivitäten des Benutzers verfolgen. Auch die Zeit, die der Benutzer auf den einzelnen Seiten verbringt, kann (mit einer gewissen Fehlerwahrscheinlichkeit) bestimmt werden. Die Methode ist ein guter Kompromiss, weil sie den Benutzer nur wenig beeinträchtigt aber trotzdem viele Informationen sammelt.
- *Browser-Agent*: auch hier können die Internet-Aktivitäten des Benutzers aufgezeichnet werden, sogar mit Informationen zu den Aktivitäten auf den einzelnen Seiten. Anand und Mobasher [AM05] erwähnen eine Studie, nach der die Verweilzeit und die Häufigkeit von Scrollen nützliche Indikatoren für das Interesse sind. Nachteil dieses Verfahrens ist, dass der Benutzer Software installieren und der Hersteller diese auch warten muss.
- *Arbeitsplatz-Agent*: dieser kann alle Interaktionen am Rechner des Benutzers aufzeichnen. Die gesamten Aktivitäten und alle geöffneten Dateien und Webseiten plus deren Inhalt stehen so zur Verfügung, eventuell sogar für mehrere adaptive Anwendungen. Auch hier muss der Benutzer spezielle Software installieren, die vom Hersteller erst entwickelt und anschließend gewartet werden muss.
- *Zugriffsprotokolle*: die Zugriffe der Benutzer auf eine bestimmte Website werden analysiert. So stehen gleich für mehrere Benutzer auf einmal Informationen zur Verfügung. Da diese jedoch nur von einer Website stammen, kann die tatsächliche Menge sehr gering ausfallen.
- *Suchprotokolle*: Suchaktivitäten werden bei einer Suchmaschine mitprotokolliert. Diese Informationen können sogleich für die Adaption verwendet werden. Der Benutzer muss dazu über ein Login oder Cookies eindeutig identifiziert sein, hat aber sonst keinen Aufwand.

Untersuchungen zeigten, dass das Benutzermodell umso genauer war, je mehr Informationen gesammelt wurden. Am genauesten war dieses mit Daten aus einem Arbeitsplatz-Agenten

über alle Dateien, gefolgt von einem ähnlichen Agenten für kürzlich betrachtete und bearbeitete Dateien und Informationen zu besuchten Webseiten. Das ungenaueste Modell entstand aus den Suchprotokollen, aber selbst dieses lieferte bessere Ergebnisse als eine nicht personalisierte Suche [GSCM07].

Ein Problem bei der impliziten Sammlung von Daten ist, dass die meisten Beobachtungen positiver Art sind. Das System muss eine Heuristik anwenden, um negative Rückmeldungen zu erkennen. Gerade bei adaptiven Systemen für Empfehlungen kann negatives Feedback die Effektivität stark verbessern [AM05].

## 3.3 Personalisierung

Das Benutzermodell kann laut Micarelli et al. [MGSG07] in drei verschiedenen Phasen in den den Suchprozess eingebunden werden. Die drei Methoden sind in Abbildung 3.1 auf der nächsten Seite dargestellt und arbeiten wie folgt:

- *Teil des Abfrageprozesses* (a): das Benutzermodell wird in einem einheitlichen Prozess verwendet, um Dokumente zu bewerten. Die direkte Einbindung der Personalisierung in die Suche erlaubt eine schnelle Antwort.
- *Umreihen* (b): nach dem Berechnen von nicht personalisierten Relevanzen für die Ergebnisse wird das Benutzermodell mit einbezogen. Dies erlaubt es dem Benutzer, die Personalisierung selektiv anzuwenden. Oft findet diese am Client statt, wo auch komplexe Modelle der Benutzerbedürfnisse berücksichtigt werden können.
- *Anfragemodifikation* (c): das Benutzermodell nimmt Einfluss auf die Formulierung der Informationsbedürfnisse (z. B. die Anfrage) und modifiziert bzw. erweitert sie durch Verändern oder Hinzufügen von Schlüsselwörtern, um die Bedürfnisse im Modell besser widerzuspiegeln oder Zweideutigkeiten zu beseitigen. Der große Vorteil dieses Ansatzes ist, dass der Aufwand gleich dem einer nicht personalisierten Suche ist.

Keenoy und Levene [KL05] merken zudem noch an, dass ein Algorithmus zur Personalisierung die besten Ergebnisse dann erzielt, wenn er zusätzlich zum Benutzermodell auch die traditionelle Relevanz der Ergebnisse in Bezug auf die Anfrage berücksichtigt, anstatt auf diese nur beim Auftreten von gleichen Reihungen zurückzugreifen.

## 3.4 Herausforderungen

Das Feld der adaptiven Suche ist ein Bereich mit aktueller und noch andauernder Forschung. Aus diesem Grund gibt es eine Reihe von Herausforderungen, deren Lösungen teils schon bekannt, teils noch aktiv gesucht werden. An dieser Stelle seien einige wichtige, in der Literatur genannte Problemfelder näher erläutert.

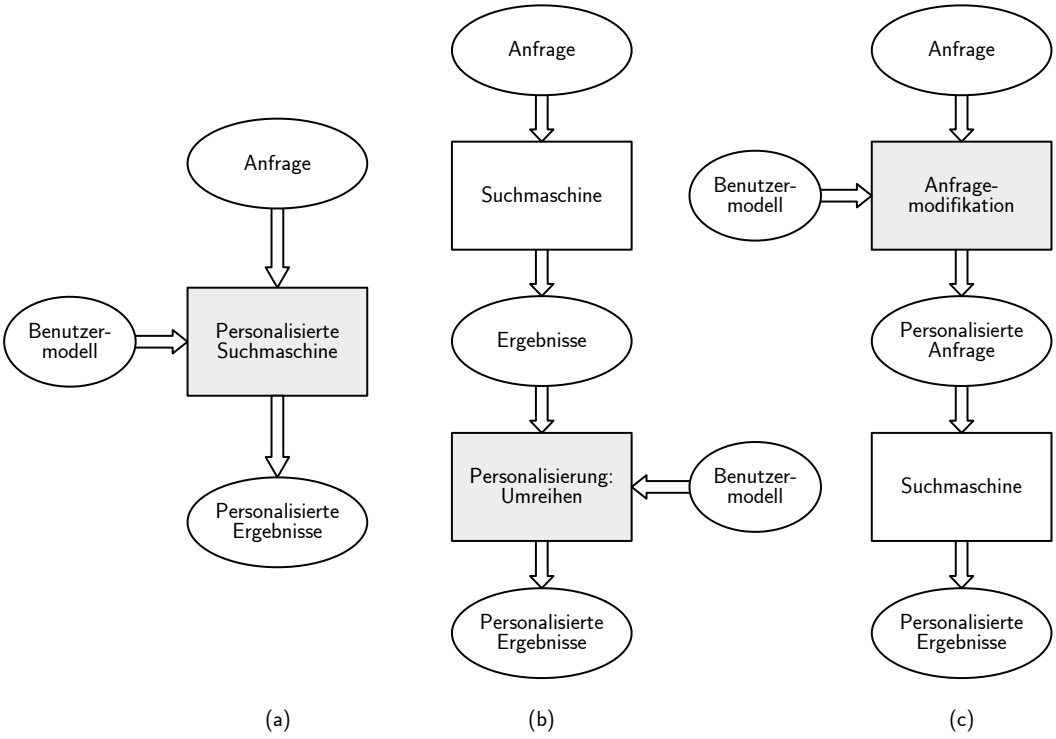


**Abbildung 3.1:** Personalisierungsprozesse bei der Suche nach Micarelli et al. [MGSG07]

Nur wenige Suchmaschinen bieten adaptive Funktionen. Micarelli et al. [MGSG07] erwähnen, dass die Systeme von Benutzern nicht einfach zu personalisieren sind, und dass mit steigendem Grad an Personalisierung mehr Schwierigkeiten in der Benutzung auftreten. Er gibt dafür drei mögliche Gründe: Erstens könnten die Benutzer nicht zu hundert Prozent damit einverstanden sein, dass *persönliche Informationen* zu ihren Informationsbedürfnissen, Interessen und Präferenzen in einem externen Suchsystem gespeichert sind. Zweitens könnten sie auch Schwierigkeiten damit haben, die *Personalisierung* als neue Funktion zu *verstehen und sinnvoll zu nützen*. Die dritte mögliche Ursache könnte eine zu *geringe Performanz* der rechenintensiven Personalisierung sein.

Auch anderweitig wird das Thema *Verarbeitungsgeschwindigkeit* als die größte Herausforderung adaptiver Suchmaschinen genannt. Die Mehrzahl dieser Systeme versucht sich am Durchsuchen des gesamten World Wide Web. Die Bewältigung der Benutzeranfragen in dieser nicht eingeschränkten Sammlung von Hypertextdokumenten ist anspruchsvoll [Bru01]. Die Hauptschwierigkeit ist die Skalierbarkeit für eine gewisse Zahl von gleichzeitig aktiven Benutzern [AM05].

Eine weniger technische als vielmehr psychologische Herausforderung betrifft das *Vertrauen der Benutzer* in eine Suchmaschine. Es konnte gezeigt werden [KOS08], dass weiter oben gereihte Ergebnisse unabhängig von ihrer tatsächlichen Relevanz öfter gewählt werden als andere. Um dies zu testen wurde die Reihung von Ergebnissen der Suchmaschine Google

systematisch umgekehrt. Sollten die Benutzer nichts davon merken, so muss es einen anderen Grund als die Relevanz für die Bevorzugung weiter vorne gereihter Ergebnisse geben.

Die Ergebnisse der Untersuchung zeigten, dass die Relevanz tatsächlich kaum Einfluss auf die Wahl der Ergebnisse durch die Benutzer hat. Nur manche Probanden suchten in den verkehrt gereihten Ergebnislisten nach den Ergebnissen mit der höchsten Relevanz. Eine mögliche Erklärung dafür ist ein starkes Vertrauen in die Korrektheit der Reihung einer gewissen Suchmaschine. Dies konnte jedoch noch nicht schlüssig gezeigt werden. Eine andere Hypothese für dieses Phänomen ist, dass Benutzer lediglich zufriedenstellende und nicht unbedingt die optimalen Ergebnisse wählen [KOS08].

Sollte das Vertrauen eine sehr große Rolle spielen, so werden es neue (adaptive) Suchmaschinen schwer haben, die Benutzer von ihren Ergebnissen zu überzeugen. Ein weiteres Problem könnte die *Verzerrung der Benutzermodelle* sein. Ein Ergebnis an erster Stelle wird unverhältnismäßig oft gewählt und dadurch könnte seine Chance, in Zukunft (bei anderen Suchen) wieder an der ersten Stelle aufzuscheinen, erhöht werden, was wiederum zu mehr Klicks führt. Diese möglicherweise inkorrekte Überbewertung kann einen negativen Einfluss auf das Benutzermodell haben [KOS08].

Umgekehrt könnten Änderung im System (z. B. des Reihungs-Algorithmus) oder an den Modellen dazu führen, dass plötzlich andere Ergebnisse als sonst an oberster Stelle aufscheinen. Loyale Benutzer würden diese ungeachtet ihrer Relevanz trotzdem wählen und könnten dadurch ihr Modell verzerren und später wenig zufriedenstellende Ergebnisse erhalten [CS07]. Überhaupt kann es für Benutzer von adaptiven Suchmaschinen verwirrend sein, wenn aufgrund von Änderungen im Modell für gleiche Suchanfragen jedes Mal eine andere Reihung oder überhaupt andere Ergebnisse zurück gegeben werden [Bru96b].

Eine Herausforderung ist auch, den Benutzern zu zeigen, warum ein Ergebnis als relevant erkannt wurde. In einer Untersuchung bevorzugten die Probanden eine Erklärung dazu, welche Prozentzahl der Benutzer bei der selben Anfrage das jeweilige Ergebnis gewählt haben. Eine textuelle Erklärung wurde dabei wegen ihrer Länge als nicht geeignet erkannt und durch geeignete graphische Elemente ersetzt [CS07].

## 3.5 Implementierungen und Systeme

Laut Keenoy und Levene [KL05] befindet sich die Personalisierung von Web-Suche noch in den Kinderschuhen. Noch 2003 fand eine Studie bei 60 öffentlich zugänglichen Web-Suchmaschinen keine oder wenig Personalisierung der Suchfunktionen. Bereits im Jahr 2001 kaufte Google Suchtechnologie von Outride, einer adaptiven Suchmaschine, die Anfragen auf der Basis der Lesezeichen im Browser und der betrachteten Seiten personalisiert. Wie in einem der nächsten Abschnitte zu sehen sein wird, verfolgte Google aber andere Pläne.

Mit einem Protokollieren der Suchanfragen und der gewählten Ergebnisse bei registrierten Benutzern besaßen Google, Yahoo! und Amazon mit seiner A9 Suchmaschine laut Coyle und

Smyth [CS07] im Jahr 2007 zumindest eine Grundlage für die Personalisierung von Suchergebnissen. Zu dieser Zeit wurde diese allerdings noch nicht genutzt. Als mögliche Ursache dafür nennt er Sorgen der Benutzer bezüglich ihrer *Privatsphäre*. Ein anderer Grund könnte sein, dass die Betreiber erst langfristige Studien durchführen wollten, bevor sie Personalisierung in den allgemeinen Betrieb integrierten. Gauch et al. [GSCM07] merken dazu an, dass beim Wechsel von der Forschung in den Echtbetrieb die Anforderungen an die *Genauigkeit der Benutzerprofile* steigt.

In der Forschung hingegen findet sich eine ganze Reihe von personalisierten Suchmaschinen mit den unterschiedlichsten Ansätzen. Besonders in den vergangenen paar Jahren wurden viele Arbeiten publiziert, wie eine *Meta-Studie* zeigt [SB08]. Im Folgenden werden zwei Systeme aus der Forschung und zwei im praktischen Einsatz näher betrachtet. Sie sollen als Beispiele für konkrete Umsetzungen der zuvor besprochenen Grundlagen dienen.

### 3.5.1 I-Spy

I-Spy [SB06] ist eine *kollaborative, personalisierte Meta-Suchmaschine*. Die grundlegende Hypothese ist, dass Suchende mit ähnlichen Interessen auch ähnliche Anfragen benutzen, um nach ähnlichen Informationen zu suchen, und anschließend ähnliche Ergebnisse auswählen. Das System reiht daher die Ergebnisse derart um, dass jene Seiten weiter oben aufscheinen, die in der Vergangenheit bei ähnlichen Suchanfragen durch andere Benutzer bevorzugt gewählt wurden. Auch Ergebnisse, die bei dieser speziellen Anfrage nicht enthalten sind, aber in der Vergangenheit beliebt waren, können in die endgültige Ergebnisliste aufgenommen und vorgereiht werden.

Das Sammeln von Daten über die Benutzer geschieht *implizit*, wenn diese eine Suchanfrage eingeben und ein Ergebnis auswählen. Es wird angenommen, dass diese Aktionen ein gewisses Maß an Interesse beim Benutzer anzeigen. Werden nun für eine bestimmte Anfrage öfters gewisse *Ergebnisse gewählt*, so wird ein gewisser Grad der Relevanz angenommen. Die Entwickler argumentieren, dass trotz mancher Ungenauigkeiten die verlässlichen Auswahl-Aktionen überwiegen werden [SB06]. Der Vorteil dieses Verfahrens ist außerdem, dass der eigentliche Inhalt der Ergebnisse nicht analysiert werden muss [MGSG07].

Gespeichert, wie oft ein Ergebnis für eine bestimmte Anfrage gewählt wurde, wird in der so genannten „hit-matrix“, die Suchanfragen und Ergebnisse über ihre „hits“ verknüpft. Eine Übereinstimmung der gefundenen Seiten und das Finden einer Ähnlichkeit bei Suchanfragen funktioniert allerdings nur in einem sehr engen Kontext. Der kollaborative Ansatz von I-SPY ist daher, dass die „hit-matrix“ von einer Gruppe von Benutzern in einem *spezifischen Interessensbereic*h gefüllt und benutzt werden soll. Das System ermöglicht daher das Anlegen mehrerer derartiger Matrizen und bietet Benutzergruppen somit eine Suche, die an ihr Gebiet und die bevorzugten Seiten angepasst ist [SB06]. Diese Technik nennt man *kollaborative Web-Suche* [CS07].

Bei kollaborativen Web-Suchmaschine dieser Art müssen sich die Benutzer nicht explizit einloggen und es werden auch keine individuellen Benutzerprofile gespeichert. Dies hat einige Vorteile: die Suchenden profitieren von der gemeinsamen Erfahrung ihrer Gruppenkollegen. Gleichzeitig müssen sie nicht besorgt sein, dass persönliche Information in einer ihnen zuordenbaren Form gespeichert wird. Außerdem werden die Daten in einer unaufdringlichen Weise gesammelt, sodass keine weiteren Eingaben der Benutzer nötig sind [CS07].

Wichtig im Zusammenhang mit den Matrizen ist auch die Frage der *Wartung der Modelle*. Ein Vorschlag der Entwickler von I-SPY ist, die Anzahl, wie oft eine Seite gewählt wurde, über die Zeit hinweg zu verringern. Dadurch müssen Seiten, die ihre wahre Relevanz womöglich verloren haben (z. B. weil sich ihr Inhalt geändert hat), wieder öfters gewählt werden, um ihre Reihung im Ergebnis zu behalten. Dies vermindert auch den Vorteil älterer Seiten gegenüber neueren, die noch nicht so oft gewählt wurden. I-SPY überprüft darüber hinaus auch regelmäßig die in den Matrizen verzeichneten Seiten und entfernt tote Links [SB06].

### 3.5.2 Google Personalized Search

Über die Jahre hinweg hat Google mehrere Aktivitäten gesetzt, seinen Suchdienst mit personalisierten Funktionen auszustatten. Die meisten dieser Versuche waren experimenteller Natur und lediglich angemeldeten oder überhaupt nur auserwählten Benutzern zugänglich. Mit der hier als „Version 2“ titulierten Methode bot Google erstmals einer breiten Öffentlichkeit Zugang zu personalisierten Ergebnissen. Eine andere Art der Personalisierung, die schon länger eingesetzt wird, basiert auf dem Standort des Benutzers und verwendet zur Lokalisierung dessen IP-Adresse [Goo08c].

#### Version 1 – Interessensbasiert

Die erste Version von Googles personalisierter Suche war im „Labs“-Bereich zu finden und als Testversion deklariert. Man konnte für sein Benutzermodell aus einer *vorgegebenen Menge von Themenbereichen* jene wählen, die einen interessierten. Während der Suche wurden die Ergebnisse in eine Taxonomie dieser Themenbereich klassifiziert und diese Information an den Browser mitgeliefert.

Der Benutzer erhielt die nicht personalisierte Ergebnisliste angezeigt und konnte mit einem Schieberegler den *Grad der Personalisierung festlegen* (siehe Abbildung 3.2 auf der nächsten Seite). Die Ergebnisse wurden dann entsprechend den vorher angegebenen Interessen umgereiht, sodass zum Benutzermodell passende Seiten weiter oben aufschienen [MGSG07].

#### Version 2 – Historienbasiert

In der zweiten Version der personalisierten Suche[3] setzte Google auf eine andere Strategie. Das Benutzermodell wird nun, wenn man eingeloggt ist, *implizit anhand der Suchanfragen*

[3] `http://www.google.com/psearch` (16. Oktober 2008)

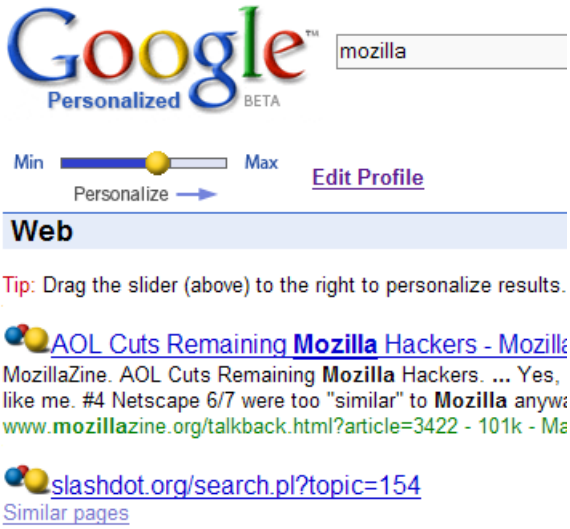


**Abbildung 3.2:** Personalisierte Google-Suche mit Schieberegler zum Steuern, wie stark die Reihung der Ergebnisse personalisiert werden soll.

*und gewählten Ergebnisse* als ein internes Abbildung der Benuzterinteressen aufgebaut. Der Benutzer hat Einsicht in diese „Web History“, kann einzelne Suchen entfernen und die Aufzeichnung temporär anhalten. Während der Suche adaptiert die Suchmaschine die Ergebnisse und reiht jene, die ähnlich zu schon betrachteten sind, höher [MGSG07].

Welche Ergebnisse umgereiht wurden, ist nicht sichtbar. Es wird lediglich ein Hinweis auf der Ergebnisseite angezeigt, wenn eine Personalisierung vorgenommen wurde. Interessant ist auch, dass eine Minimalversion dieses Verfahrens auch *für nicht eingeloggte Benutzer* angewendet wird. In einem Session-Cookie wird immer die letzte Suchanfrage gespeichert und bei einer erneuten Suche als Kontextinformation verwendet [Goo08c].

### Version 3 – Kollaborativ

Im Jahr 2007 startete Google erneut Experimente zur Personalisierung von Ergebnissen. Im ersten Prototypen[4] (siehe Abbildung 3.3 auf der nächsten Seite) konnten die Benutzer *Ergebnisse verschieben, entfernen und hinzufügen*. Das Betätigen der Schaltfläche mit einem nach oben gerichteten Pfeil neben einem Suchergebnis schob dieses ganz nach oben auf der Seite und markierte es als personalisiert. Die Schaltfläche mit dem Kreuz entfernte das jeweilige Ergebnis aus der Liste. Man konnte auch selbst Seiten hinzufügen; diese wurden ebenfalls als personalisiert markiert.

Die Änderungen an der Ergebnisseite wurden mit den verwendeten Suchbegriffen verknüpft und waren beim erneuten Suchen mit diesen wieder sichtbar. Gespeichert wurden sie pro Benutzer, sie hatten also keinen Einfluss auf die Suchen Anderer. Eine Möglichkeit zum Rückgängigmachen der Änderungen war ebenfalls vorgesehen.

[4] `http://www.google.com/experimental/a840e102.html` (16. Oktober 2008)


**Abbildung 3.3:** Personalisierte Google-Suche mit Hinweis auf Personalisierung (1a), Bewertungsschaltflächen (1b) und einer Möglichkeit, Seiten vorzuschlagen (1c).

Ein aktueller Prototyp[5] brachte einige Neuerungen gegenüber dem ersten. An der Stelle von entfernten Ergebnissen erscheint ein entsprechender Hinweis, und man kann am unteren Ende der Seite diese wieder anzeigen lassen. Zu Ergebnissen kann man *Kommentare hinzufügen* und die eigenen Kommentare auch verändern.

Man kann sich eine Übersicht aller Bewertungen und Kommentare, die man zu Ergebnissen abgegeben hat, anzeigen lassen. Diese ist nach Suchanfragen gegliedert. Die Ergebnisliste kann auch mit den Bewertungen und Kommentaren aller anderen Benutzer angezeigt werden. Die Zahl der positiven bzw. negativen Bewertungen wird bei jedem Ergebnis angezeigt, und diese sind diesen Werten entsprechend sortiert.

Auch die Modelle anderer Benutzer mit ihren Bewertungen und Kommentaren können eingesehen werden. Kommentare kann man in der Ergebnisliste bewerten. Inwieweit diese Bewertungen in die Reihung der Kommentare einfließt, ist nicht bekannt.

### 3.5.3 Eurekster

Eurekster ist eine *kollaborative Suchmaschine*, die Suchergebnisse umreihen kann. Für jede Suchanfrage speichert sie, welche Ergebnisse durch den Benutzer bewertet oder gewählt wurden. Diese Daten beeinflussen die anderen Mitglieder der Benutzergemeinde, die an dem

[5] `http://www.techcrunch.com/2008/07/16/is-this-the-future-of-search/` (16. Oktober 2008)

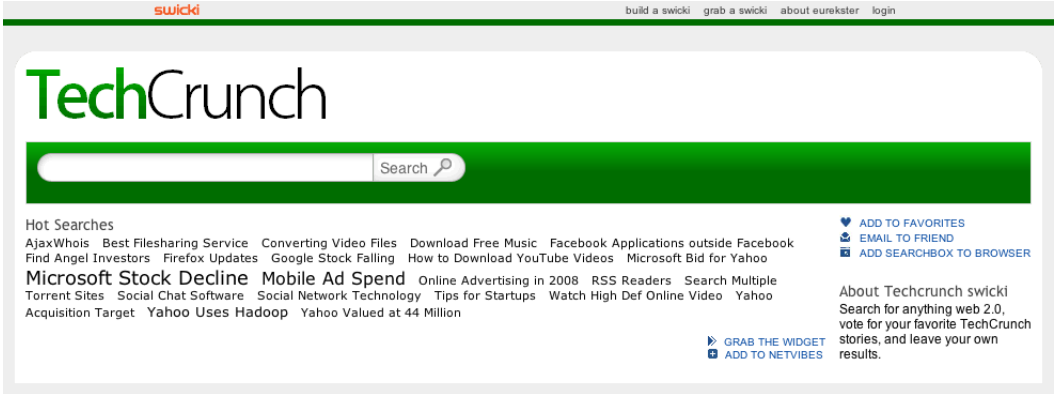


**Abbildung 3.4:** Eurekster – Swicki für TechCrunch mit „buzzcloud“ und Suchformular

selben Themengebiet interessiert sind. Gemeinden bilden sich rund um spezielle *Suchportale, genannt Swickis*. Diese können in eine Webseite oder einen Blog eingebaut werden und bieten den Besuchern somit eine Suchmaschine, die auf ein bestimmtes Thema fokussiert ist.

Ein Swicki wird im Browser als Suchformular mit einer „buzzcloud“ dargestellt (siehe Abbildung 3.4). Diese Begriffswolke (tag cloud) enthält eine Auswahl vorangegangener Suchen, oft genutzter Suchbegriffe und Schlüsselwörter zum Thema des Swickis. Durch Anwählen eines dieser Begriffe wird eine Suche mit diesem gestartet. Alternativ kann man Suchanfragen wie gewohnt im Eingabefeld des Suchformulars eingeben.

Die Ergebnisse einer Suche werden in einer einfachen Liste angezeigt (siehe Abbildung 3.5 auf der nächsten Seite). Jedes Ergebnis kann *positiv oder negativ bewertet* werden, selbst durch anonyme, nicht angemeldete Benutzer. Eingeloggte Benutzer können einzelne *Ergebnisse kommentieren* und *neue Schlüsselwörter* zum besseren Fokussieren der Suchmaschine vorschlagen. Das Ziel ist, dass das Swicki durch die Bewertungen und das Wählen von Ergebnissen durch die Benutzer implizit lernt und sich so an die Benutzergemeinde anpasst.

Bei Eurekster kann jeder kostenlos ein eigenes Swicki erstellen. Der Fokus eines Swickis wird vom erstellenden Benutzer durch Schlüsselwörter festgelegt. Suchergebnisse, die diese Schlüsselwörter enthalten, werden automatisch vorgereiht. Mit einer Whitelist und einer Blacklist für gewisse URLs kann man Seiten explizit priorisieren oder vom Suchergebnis ausschließen. Basierend auf den Schlüsselwörtern, dem Inhalt der „buzzcloud“ und den Seiten in der Whitelist ermittelt der Adaptionsalgorithmus Möbius automatisch neue Begriffe für die „buzzcloud“ und Seiten für eine Graylist (diese zuvor nicht speziell berücksichtigten Seiten werden in künftigen Suchen priorisiert). Besitzt die Seite, in die das Swicki eingebettet ist, HTML Meta-Tags in ihrem Quellcode, so werden die Begriffe aus den Schlüsselwörtern automatisch zur „buzzcloud“ hinzugefügt.

Zwei Gruppen beeinflussen also die Personalisierung der Swickis: die erstellenden Benutzer administrieren sie und geben den groben Fokus vor. Die suchenden Benutzer verfeinern die Ergebnisse durch ihre Suchen, Bewertungen und Navigation. In diesem Sinne sind in Eurekster

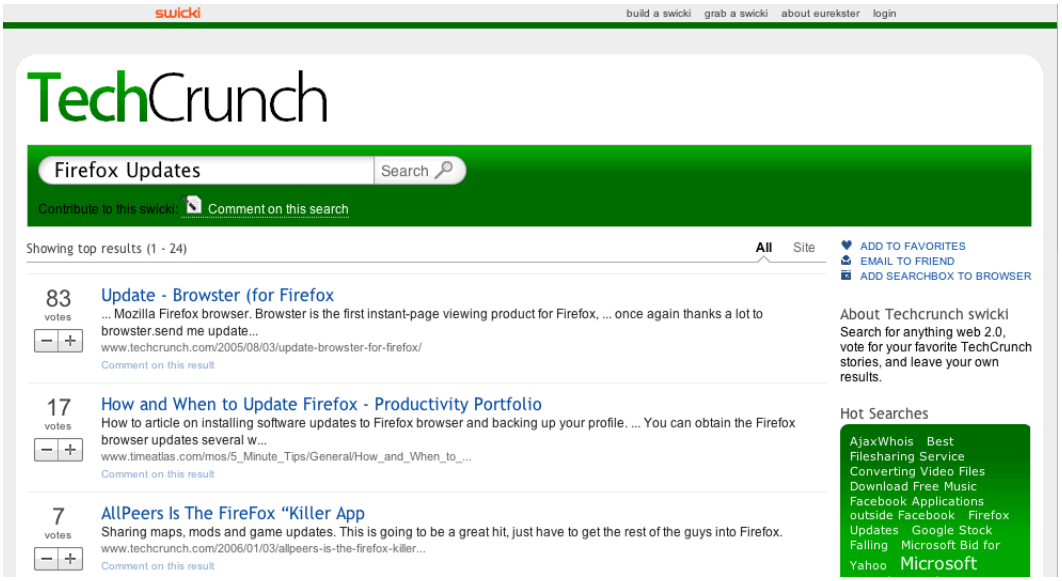


**Abbildung 3.5:** Eurekster – Suchergebnisse mit Bewertungen

sowohl Konzepte von I-SPY als auch der dritten Version von Googles personalisierter Suche umgesetzt.

### 3.5.4 Persona

Das System Persona [TM02] ist eine personalisierte Suchmaschine, die Interessen des Benutzers bei der Auswahl und Reihung der Ergebnisse berücksichtigt. Es baut auf den Daten des *Open Directory Project* auf und verwendet diese sowohl zum Finden von Suchergebnissen als auch in Form eines Overlay-Modells zum Speichern der Interessen eines Benutzers. Modelliert wird auf der Basis von *expliziten Rückmeldungen* des Benutzers.

Der grundsätzliche Ablauf einer Suche in Persona funktioniert wie folgt: Die Suchanfrage wird zuerst an *dmoz*, die Web-Schnittstelle des Open Directory Project, weitergegeben. Als Ergebnisse liefert diese eine Menge von URLs, die im ODP-Datenbestand enthalten und in der Ontologie klassifiziert sind. Mittels einer erweiterten Version des graphenbasierten Algorithmus HITS werden diese Ergebnisse auf der Basis der Informationen im Benutzermodell gereiht und dem Benutzer präsentiert. Dieser hat die Möglichkeit, einzelne Seiten positiv bzw. negativ zu bewerten und so sein Benutzermodell zu verfeinern.

Das Benutzermodell ist als Overlay über die gesamte ODP-Ontologie organisiert. Dies ermöglicht es, ein *sehr spezifisches Benutzermodell* zu erzeugen. Um die Größe des Modells handhabbar zu halten, werden nur Knoten zu jenen Themen gespeichert, zu denen der Benutzer eine Wertung abgegeben hat [GSCM07]. In jedem Knoten, der einem Thema der ODP-Ontologie entspricht, ist gespeichert, wie oft er in einer Suche bereits vorgekommen ist, die URLs, die ihm zugeordnet sind, und die Zahl deren positiver bzw. negativer Bewertungen.

Da aufgrund des Größe des ODP-Datenbestands die Interessen im Modell sehr feingliedrig gespeichert sind, muss eine Heuristik angewendet werden, um für Ergebnisse in Themen, die noch nicht im Modell verzeichnet sind, eine sinnvolle Bewertung zu finden. Persona berücksichtigt daher auch die Informationen in Knoten, die eine Ebene höher bzw. tiefer liegen als jener Knoten, der bei der Suche identifiziert wurde. Beim Reihen der Ergebnisse werden die Bewertungen der Seiten, die einem Knoten zugeordnet sind, dann verwendet, um die Suchergebnisse umzureihen.

In vielen Punkten ist Persona dem System Prospector ähnlich. Ein großer Nachteil ist jedoch, dass es auf die Seiten, die im ODP-Datenbestand klassifiziert sind, eingeschränkt ist. Diese Einschränkung bringt einige Erleichterungen mit sich, machen das System für den Einsatz in der Praxis aber nicht wirklich geeignet.

# Kapitel 4

# Open Directory Project

Der Blick in die Fachliteratur, insbesondere im Bereich Information Retrieval aber auch speziell bei adaptiven Systemen zeigt, dass das Open Directory Project (ODP) [ODP08] in einer Vielzahl von Systemen verwendet wurde und noch immer wird. Auch für das in dieser Arbeit beschriebene adaptive Suchsystem liefert es die ontologischen Metadaten zu den gefundenen Suchergebnissen. Diese sind für die Benutzermodellierung und die Personalisierung von zentraler Bedeutung.

In diesem Kapitel wird ein Überblick zum Open Directory Project, seiner Geschichte und der Verankerung in der Web-Gemeinde gegeben. Es werden Grundlagen zu den Begriffen Ontologie und Taxonomie besprochen und ihre Umsetzung im ODP erläutert. Abschließend werden die konkreten Daten des Projekts näher betrachtet, Statistiken zu deren Umfang und Verteilung angeführt und der Datenexport beschrieben.

## 4.1 Projekt

Das World Wide Web wächst mit einer rasanten Geschwindigkeit und überfordert seine Benutzer mit einer wahren Informationsflut. Suchmaschinen ermöglichen es, einen Teil dieser riesigen Datenmenge mittels Anfragen zu durchsuchen. Eine anderer Ansatz ist, die Web-Inhalte in so genannten *Web-Verzeichnisse* zu organisieren. Web-Verzeichnisse haben laut Cacheda und Baeza-Yates [CBY04] den Zusatznutzen, dass sie nicht nur durchsucht sondern auch manuell von den Benutzern durchgesehen werden können.

Das Open Directory Project ist ein derartiges Verzeichnis, ebenso wie das Yahoo!-Verzeichnis[1]. Laut eigenen Angaben ist es das *größte und umfassendste* von Menschen geschaffene Web-Verzeichnis. Aufgebaut und gewartet wird es von einer großen, globalen Gemeinde *freiwilliger Redakteure* [ODP08]. Der Inhalt kann über die Seite `http://www.dmoz.org` eingesehen und erweitert werden.

Das Verzeichnis gliedert die Verweise zu den einzelnen Web-Inhalten in *Themenbereiche*. Diese sind oberflächlich betrachtet baumartig strukturiert, was dazu führt, dass in der Literatur oft von einer thematischen Hierarchie gesprochen wird [PPP+04, LB07]. In Wirklichkeit handelt

[1] `http://dir.yahoo.com`

es sich, wie Cacheda und Baeza-Yates [CBY04] treffend erkennt, um eine *Ontologie*. Die Themen sind in einem gerichteten Graphen angeordnet, denn sie können neben keinen oder mehreren untergeordneten auch ein oder mehrere übergeordnete Themen besitzen. Web-Inhalte können darüber hinaus in mehr als einem Thema eingeordnet sein. Durch diese Struktur wird das Verzeichnis sehr mächtig und flexibel.

Da Menschen und nicht Maschinen das Verzeichnis aufbauen und warten, umfasst es Wissen, das nicht durch rein maschinelle Textklassifikation gewonnen werden kann. Es enthält das kollektive Wissen aller freiwilligen Redakteure [GM07]. Doch dieser manuelle Ansatz bedingt auch, dass nur ein beschränkter Bereich des World Wide Web katalogisiert werden kann, laut Cacheda und Baeza-Yates [CBY04] weniger als 1% aller Webseiten. Durch die große Zahl von bestehenden Themen und gewissen Inkonsistenzen ist ein Durchstöbern auf der Suche nach interessanten Information auch oft schwierig. Trotzdem wird eine große Vielzahl an Bereichen teilweise auch sehr detailliert abgedeckt [PPP+04]. Web-Verzeichnisse im Allgemeinen stellen mit ihrer Auswahl der Web-Inhalte *Qualität über Quantität* [CBY04].

### 4.1.1 Geschichtliche Entwicklung

Vor kurzem feierte das Open Directory Project seinen zehnten Geburtstag[2]. Ins Leben gerufen wurde es am 5. Juni 1998 von zwei Programmierern aus Kalifornien, Rich Skrenta und Bob Truel. Ursprünglich GnuHoo genannt, wurde es nach kurzer Zeit in NewHoo umbenannt, um Verwechslungen mit Aktivitäten des GNU-Projekts zu vermeiden [Sul98].

Schon von Beginn an war das Projekt darauf ausgerichtet, von freiwilligen Redakteuren aus dem gesamten World Wide Web getragen zu werden. Nach kurzer Zeit hatten 400 Redakteure bereits 31.000 Seiten in 3.900 Themenbereiche eingeordnet (Stand Juli 1998). Das Yahoo!-Verzeichnis verfügte zu dieser Zeit über 80 bezahlte Redakteure und umfasste 750.000 Seiten. Die Basis für die thematische Struktur des Verzeichnisses stammte übrigens von den Gruppen des Usenet und wurde händisch zur ersten Version der Hierarchie entwickelt [Sul98].

Später wurde das Projekt von der Netscape Communication Corporation übernommen, die es heute noch mit einem kleinen Mitgliederstab betreut [ODP08]. Bei diesem Schritt erhielt es auch seinen heutigen Namen.

### 4.1.2 Community und Lizenz

Laut eigenen Angaben arbeiteten über 70.000 freiwillige Redakteure aus der ganzen Welt am Open Directory Project. Im Durchschnitt waren zu jedem Zeitpunkt rund 6000 Zugangskonten aktiv. Jede Woche stoßen im Schnitt 78 neue Redakteure zum Projekt (Stand Oktober 2007)[Ope07].

Das Open Directory Project wurde im Geiste der Open Source-Bewegung gegründet und ist das einzige größere Verzeichnis, das *zu 100% frei* ist. Es wird versichert, dass Beiträge und

[2] `http://blog.dmoz.org/2008/06/05/dmoz-turns-10` (11.8.2008)

die Nutzung der Daten immer kostenlos sein werden. Jeder kann die Daten gratis benutzen, wenn er die Lizenz akzeptiert. Diese erlaubt es, die Daten zu kopieren, in anderen Arbeiten zu verwenden, zu verteilen und veröffentlichen. Wichtigste Bedingung ist, dass in allen diesen Fällen auf das ursprüngliche Projekt als Quelle zurückverwiesen und darauf hingewiesen wird, wenn Änderungen vorgenommen wurden [ODP08].

Diese freigiebige Lizenz macht es auch zur am weitesten verbreiteten Quelle von durch Menschen klassifizierte Web-Inhalte. Verwendet werden die Daten des ODP in mehreren hundert Projekten und Produkten, unter anderem in großen Suchmaschinen wie Google, AOL Search, Lycos und HotBot [ODP08].

## 4.2 ODP-Ontologie

In der Literatur wird die Struktur der Daten des Open Directory Project wiederholt als Taxonomie und Ontologie bezeichnet. Aus diesem Grund werden nun die Grundlagen zu diesen Begriffen erläutert. Anschließend wird beschrieben, wie diese Konzepte im ODP konkret umgesetzt wurden.

### 4.2.1 Grundlagen und Definitionen

Der Begriff Ontologie stammt aus einem Bereich der Philosophie, der sich mit dem Studium des Seins und der Existenz beschäftigt. Dort bezeichnet er eine *Theorie zum Wesen der Existenz*. Schon Aristoteles hat Kategorien wie Substanz und Qualität identifiziert, aus denen alles, was existiert, aufgebaut sein soll [Gru09]. Die philosophische Ontologie ist die *Wissenschaft von allem, was existiert*, den Arten und Strukturen von Objekten, Eigenschaften, Ereignissen, Prozessen und Beziehungen in jedem Bereich der Wirklichkeit. Der eigentliche Begriff wurde 1613 von einander unabhängig durch die zwei Philosphen Rudolf Göckel (im *Lexicon philosphicum*) und Jacob Lorhard (im *Theatrum philosphicum*) eingeführt [SW01].

Im Bereich der *Informatik* bezeichnet Ontologie ein Artefakt, das den Zeck hat, die Modellierung von Wissen über einen gewissen (realen oder imaginären) Bereich zu ermöglichen [Gru09]. Eine viel zitierte Definition dazu, die noch immer Gültigkeit besitzt [Smi04], lautet:

> *„A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose ... An ontology is an explicit specification of a conceptualization.“* [Gru95]

*Formal beschriebenes Wissen* basiert auf einer solchen Begrifflichkeit (conceptualization) und enthält die Objekte, Konzepte und anderen Gebilde, von denen man annimmt, dass sie in einem gewissen Bereich existieren, und die Beziehungen zwischen ihnen. Jede Wissensbasis und die darauf basierenden Systeme bedienen sich explizit oder implizit einer derartigen Begrifflichkeit [Gru95].

Eine Ontologie als Spezifikation einer Begrifflichkeit definiert ihrerseits Konzepte, Beziehungen und Unterscheidungen, die nötig sind, um den Bereich zu modellieren. Sie *spezifiziert Vokabular* wie Klassen oder Mengen, Attribute bzw. Eigenschaften und Beziehungen zwischen Klassen und zwischen Elementen einer Klasse. Sie enthält auch Informationen zu deren Bedeutung, Einschränkungen und logisch konsistenten Anwendung [Gru09].

Die Ontologie spezifiziert auf einer semantischen Ebene das Vokabular, mit dem Agenten wie beispielsweise Softwareprogramme Wissen und Abfragen austauschen können. Derart als *Spezifikation einer Schnittstelle* verstanden, stellt sie eine Sprache für die Kommunikation mit diesen Agenten dar [Gru09].

Der Begriff Ontologie wurde für eine große Zahl von Artefakten für die Strukturierung von Information verwendet, von Katalogen über Glossare und Thesauri bis zu Taxonomien und Mengen logischer Bedingungen [SW01]. Gruber geht sogar soweit zu sagen, dass Ontologien ein Werkzeug und Produkt der Technik sind und daher durch ihre Verwendung definiert sind [Gru09]. Auch werden sie oft mit Hierarchien von Klassen in Taxonomien gleichgesetzt, doch sie sind nicht auf diese Form beschränkt [Gru95]. Eine Definition von Smith et al. [SKSC06] zeigt jedoch, dass Taxonomien ein wichtiger Bestandteil einer Ontologie sind:

> *„An ontology is a representational artifact, comprising a taxonomy as proper part, whose representational units are intended to designate some combination of universals, defined classes, and certain relations between them.“*

Eine *Taxonomie* wird definiert als graphen-theoretisches, begriffliches Artefakt in Baumform. Die Knoten repräsentieren Klassen und Eigenschaften, die von allen dem Knoten zugeordneten Ausprägungen geteilt werden. Die Kanten zwischend den Knoten stellen „ist ein“- oder Untermengen-Relationen dar [SKSC06].

### 4.2.2 Struktur

Die Grundstruktur der Daten des Open Directory Project entspricht eine Taxonomie, wie sie im vorangegangenen Unterabschnitt definiert wurde. Die Knoten des Baums repräsentieren dabei als Klassen die einzelnen *Themengebiete*, beim ODP *Topic* genannt. Deren Umfang und Bedeutung sind mit einem Wort oder einer kurzen Phrase festgelegt. Auf oberster Ebene des Baums finden sich beispielsweise die Topics „Arts“, „Business“, „Computers“, „Regional“ oder „World“. In Abbildung 4.1 auf der nächsten Seite sind diese als Ovale dargestellt.

Die Kanten des Baums sind gerichtet und führen von übergeordneten Themen zu untergeordneten. In der Web-Oberfläche des ODP werden sie als Hyperlinks dargestellt. Bei den Kanten findet sich sowohl die Semantik einer „ist ein“-Beziehung als auch die einer Untermengen-Relationen. So sind beispielsweise „Cheese“ und „Meat“ ein „Food“ und daher diesem Thema untergeordnet. Das Thema „Europe“ enthält hingegen als thematisch untergeordnete Menge alle Staaten im geographischen Gebiet Europas. Doch nicht immer sind die Themen mathematisch so klar aufgeteilt. Gerade beim Thema „Europe“ finden sich neben den Staaten auch Unterthemen wie „Education“, „Weather“ oder „Government“, die Gesamteuropa betreffen.

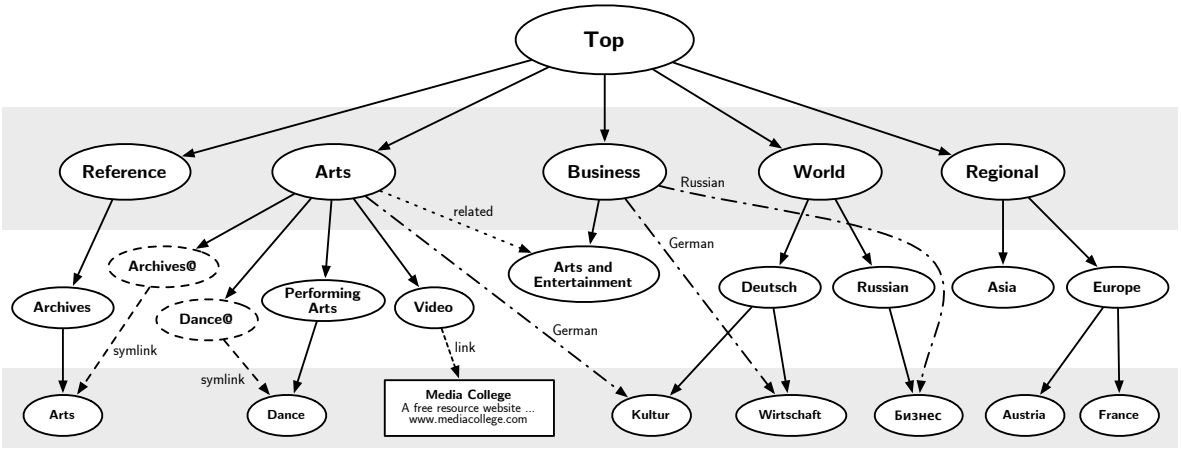


**Abbildung 4.1:** Teile der Struktur des ODP-Verzeichnisses

Innerhalb von Themen wird teilweise zwischen drei Ebenen von Verweisen auf untergeordnete Themen unterschieden. Diese Ebenen enthalten jeweils Unterthemen unterschiedlicher Relevanz und sind in der Weboberfläche durch einen horizontalen Strich getrennt. Rund ein Drittel der Themen enthält laut Gabrilovich und Markovitch [GM07] außerdem eine Beschreibung zu Bedeutung und Umfang, sowie Hinweise, welche Inhalte darin klassifiziert werden sollen.

Die konkreten Inhalte eines Themas sind seine *Verweise* (*link*) auf Web-Inhalte. Jeder dieser Verweise verfügt über einen Link in Form einer URL, einen Titel und eine kurzen Beschreibung des Inhalts. Theoretisch könnten in jeder Ebene Inhalte klassifiziert sein. In Wirklichkeit finden sich aber, mit Ausnahme der Hauptthemen „Health“, „News“ und „Regional“, erst in der zweiten oder dritten Ebene Verweise auf Web-Inhalte.

Neben den durch die Baumform bedingten Beziehungen zwischen übergeordneten und untergeordneten Themen gibt es auch Kanten zwischen Knoten, die mehr als eine Ebene voneinander entfernt sind oder überhaupt in einem anderen Teilbaum liegen. Es lassen sich drei Typen identifizieren:

- *Symbolische Verweise (symlink)*: diese Art von Verweisen ermöglicht den Benutzern mehrere Wege zu einem Thema zu wählen. Durch sie sind Inhalte oft auch mehr als einem Themenpfad zugeordnet. In der Web-Oberfläche sehen symbolische Verweise wie normale Themen aus, in ihrem Namen befindet sich am Ende jedoch ein @-Symbol. Möglich sind Verweise:
  - *im selben Teilbaum nach unten*: „Arts / Typography@“ zeigt auf „Arts / Graphic Design / Typography“.
  - *im selben Teilbaum nach oben*: erzeugt Zyklen, tritt aber ausgesprochen selten auf.
  - *über Teilbäume hinweg*: „Science / Museums@“ zeigt auf „Reference / Museums / Science“. Erlaubt eine Mehrfachklassifizierung von Inhalten und kann ebenfalls zu Zyklen führen.

- *Querverweise (related)*: sie verweisen auf andere, verwandte Themen. Beispielsweise findet sich in „Computers“ ein Verweis zu „Business / Information Technology“. In der Web-Oberfläche werden sie in einem getrennten Abschnitt mit dem Titel „See also:“ angezeigt.

- *Sprachverweise*: sie verweisen auf eine unter dem Hauptthema „World“ vorhandenen Zweig der jeweiligen Sprache und dort auf das entsprechende Theme in der Übersetzung. In „Recreation“ befindet sich so unter anderem der Verweis auf „World / Deutsch / Freizeit“. Auch sie werden in der Web-Oberfläche in einem getrennten Abschnitt mit dem Titel „This category in other languages:“ angezeigt.

Laut einer aktuellen Untersuchung [Per08] verdoppelt sich bei der Berücksichtigung symbolischer Verweise die durchschnittliche Zahl der einem Thema untergeordneten Themen. Den meisten Anteil daran haben mit mehr als 97% Verweise über Teilbäume hinweg. 89% dieser Verweise bleiben aber trotzdem im Teilbaum des jeweils obersten Themas. Bei 77% der Verweise sind zumindest die ersten zwei Ebenen gleich.

Diese *Abweichungen von der Baumform* sind der Grund, weshalb das Verzeichnis des Open Directory Project nicht als reine Taxonomie angesehen werden kann und dem umfassenderen Konzept einer *Ontologie* zugeordnet werden muss. Die anwendungsspezifische Semantik der Verweise ist ebenfalls ein Grund für diese Klassifizierung.

## 4.3 Daten in der ODP-Ontologie

Das Verzeichnis des Open Directory Project ist in 17 Hauptthemen geteilt: Adult, Arts, Business, Computers, Games, Health, Home, Kids and Teens, News, Recreation, Reference, Regional, Science, Shopping, Society, Sports, World (Stand Juli 2008). Das Thema Adult wird in der Weboberfläche nicht verlinkt, ist aber beim Export der Daten vorhanden bzw. kann durch explizites Eingeben der entsprechenden URL aufgerufen werden.

Einige der Themenbereich bedürfen einer kurzen Erklärung:

- *World*: enthält alle nicht englischsprachigen Einträge. Für jede vorhandene Sprache gibt es ein Unterthema. In diesem finden sich erneut die obigen Hauptthemen und entsprechende Unterthemen, jedoch in der jeweiligen Sprache.

- *Regional*: gliedert sich in geographische und politische Einheiten: zuerst nach Kontinenten und Regionen der Erde, dann nach Staaten, Bundesstaaten, Bezirken und Orten. Der Detaillierungsgrad ist dabei nicht in allen Bereichen gleich. Auf den unterschiedlichen Ebenen finden sich jeweils thematische Unterteilungen, beispielsweise in „Education“, „Business“ oder „Government“.

- *Kids and Teens*: besitzt ähnliche Unterthemen wie die Hauptthemen, enthält allerdings nur Inhalte, die für Kinder geeignet sind.

| **Thema** | **Unterthemen** | **Anteil** |
|---|---:|---:|
| Adult | 8.041 | 1,07% |
| Arts | 47.223 | 6,28% |
| Business | 12.208 | 1,62% |
| Computers | 8.432 | 1,12% |
| Games | 12.560 | 1,67% |
| Health | 7.069 | 0,94% |
| Home | 2.655 | 0,35% |
| Kids and Teens | 6.200 | 0,82% |
| News | 525 | 0,07% |
| Recreation | 11.194 | 1,49% |
| Reference | 12.090 | 1,61% |
| Regional | 309.138 | 41,08% |
| Science | 13.225 | 1,76% |
| Shopping | 5.381 | 0,72% |
| Society | 28.610 | 3,80% |
| Sports | 18.822 | 2,50% |
| World | 249.087 | 33,10% |
| **Gesamt** | **752.460** | **100,00%** |

**Tabelle 4.1:** Statistik zu den Themen oberster Ebene

### 4.3.1 Datenvolumen und Statistiken

Seit seiner Gründung durch zwei Privatpersonen hat das Open Directory Project ein starkes Wachstum hinter sich. Aus den anfänglichen 31.000 Verweisen auf Web-Inhalte in 3.900 Themenbereichen (siehe Unterabschnitt 4.1.1) wurden mittlerweile 4.604.652 Links in 752.460 Themen (Stand Juli 2008). Doch nicht das ganze Verzeichnis ist größenmäßig gleichförmig aufgebaut. Im Folgenden werden genauere Zahlen angeführt.

Die Verteilung der 752.460 Themen zeigt einige Besonderheiten. Eine Aufstellung der untergeordneten Themen, die jedes Thema der obersten Ebene enthält, ist in Tabelle 4.1 zu sehen. Auffallend ist, dass über 40% der Gliederung in „Regional“ und rund ein Drittel in „World“ liegen. Das lässt sich darauf zurückführen, dass in diesen beiden Bereichen in den 80 unterstützten Sprachen bzw. auf den Ebenen der geographischen und politischen Einheiten die grundlegende Themenstruktur dupliziert wird.

Kein sonderlich anderes Bild ergibt sich, wenn man die *Zahl der klassifizierten Web-Inhalte* auf die Hauptthemen aufteilt. Die Zahlen dazu sind in Tabelle 4.2 auf der nächsten Seite zu sehen. Neben der absoluten Zahl und dem Anteil sind auch die durchschnittliche Linkzahl und die Standardabweichung pro Themengebiet angegeben. Auch hier zeigt sich die Größe des regionalen Teilbaums und jenes in den Sprachen außer Englisch.

| **Thema** | **Links** | **Anteil** | $\mu$ | $\sigma$ |
|---|---|---|---|---|
| Adult | 37.316 | 0,81% | 7,26 | 10,78 |
| Arts | 265.365 | 5,76% | 6,77 | 14,47 |
| Business | 243.717 | 5,29% | 22,21 | 35,20 |
| Computers | 122.014 | 2,65% | 15,71 | 27,81 |
| Games | 58.523 | 1,27% | 5,61 | 7,57 |
| Health | 63.607 | 1,38% | 9,84 | 14,37 |
| Home | 29.605 | 0,64% | 12,13 | 17,05 |
| Kids and Teens | 45.632 | 0,99% | 8,02 | 9,52 |
| News | 9.019 | 0,20% | 19,95 | 28,78 |
| Recreation | 111.729 | 2,43% | 11,05 | 16,99 |
| Reference | 59.891 | 1,30% | 5,71 | 10,31 |
| Regional | 1.124.799 | 24,43% | 5,01 | 7,79 |
| Science | 109.053 | 2,37% | 8,78 | 13,98 |
| Shopping | 100.258 | 2,18% | 20,59 | 27,41 |
| Society | 251.387 | 5,46% | 9,92 | 25,71 |
| Sports | 104.730 | 2,27% | 6,43 | 11,68 |
| World | 1.868.007 | 40,57% | 8,63 | 14,41 |
| **Gesamt** | **4.604.652** | **100,00%** | **7,56** | **14,36** |

**Tabelle 4.2:** Link-Statistik zu den Themen oberster Ebene

In den Bereichen „Business“, „Shopping“ und „News“ sind im Schnitt besonders viele Inhalte pro Unterthema klassifiziert. Markant ist dies aber nur bei „Business“, da „Shopping“ und „News“ nicht sehr fein gegliedert sind. Doch auch die hohe Standardabweichung lässt auf große Schwankungen zwischen den einzelnen Unterthemen schließen.

Die meisten *Inhalte* finden sich übrigens zu einer katholischen Enzyklopaedie, da für diese der gesamte Index im Verzeichnis eingetragen ist. So kommt es, dass in diesem Thema 1278 Links im Unterthema zum Buchstaben A klassifiziert sind. An zweiter Stelle steht der Buchstabe C, an dritter Stelle liegen Nachrichtenartikel zu den Anschlägen vom 11. September 2001. In weiterer Folge wechseln sich mit mehr als 500 Links die Enzyklopaedie, persönliche Homepages, niederländische Schilderkunst, Netzwerkdienstleistungen, Musikbands, eine arabische Seite, Web-Designer, Video-Produzenten und Schilderhersteller ab.

Die *Verteilung der Zahl der Links pro Thema* ist in Abbildung 4.2 auf der nächsten Seite als Histogramm dargestellt. Die gewählten Themen haben zwischen einem und 50 Links. Dies entspricht 98,4% aller Themen, die überhaupt einen Link besitzen. 143.693 Themen, also rund 19% haben keinen Link. Im Diagramm sieht man, dass rund 30% der Themen nur einen Link besitzen. Die Zahl fällt dann rasch ab, und die Verteilung verflacht sich mit steigender Anzahl an Links.

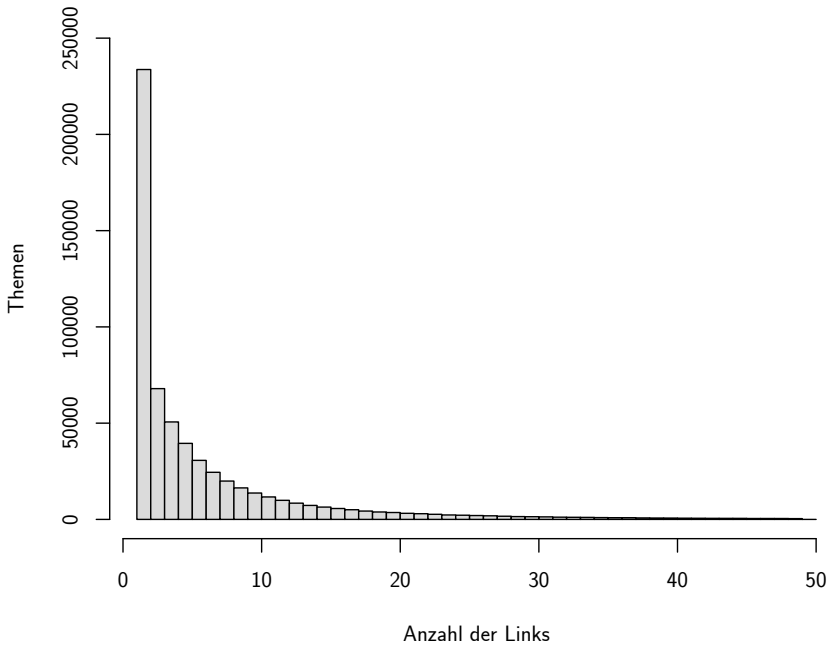


**Abbildung 4.2:** Anzahl Links pro Thema für Themen mit weniger als 50 Links. Dies umfasst 598.898 Themen und entspricht somit 98,4% all jener Themen, die überhaupt einen Link besitzen.

### 4.3.2 Export der Ontologie

Das Open Directory Project ermöglicht den *kostenlosen Download* aller seiner Daten. Jede Woche wird der gesamte Datenbestand exportiert und kann heruntergeladen werden. Zu diesem Zweck sind die Struktur und der Inhalt des Verzeichnisses getrennt. Gespeichert sind beide Teile in einem dem Resource Description Framework ähnlichen Format.

Das *Resource Description Framework (RDF)* ist eine unter anderem in XML ausdrückbare Sprache zur Beschreibung von Ressourcen. Man kann damit in einer standardisierten Form Metadaten zu beliebigen Ressourcen definieren. Festgelegt ist RDF in einer durch das World Wide Web Consortium (W3C) entwickelten Spezifikation. Es ist einer der Grundbausteine für das Semantic Web [Wor04].

Die Daten des ODP geben in ihrem Kopfelement zwar an, dass sie im RDF-Format sind, doch Versuche, sie mit einem entsprechenden Parser einzulesen, schlugen fehl. Auf der Seite des Downloads findet sich hierzu leider kein Hinweis, lediglich in einem Protokoll zu Änderungen des Formats [ODP08].

Die oben besprochene Struktur der Daten (siehe Unterabschnitt 4.2.2) wird im Struktur-Dokument wiedergegeben. Ein Auszug daraus findet sich in Code 4.1 auf der nächsten Seite. In ihm werden die Themenbereiche oberster Ebene definiert. Zu diesem Zweck gibt es das Wurzel-Thema „Top“. Es besitzt die ID mit der Zahl 1 und den Titel „Top“. Mit dem Element `narrow` werden die Beziehungen zu den untergeordneten Themen angegeben.

```
<Topic r:id="Top">
  <catid>1</catid>
  <d:Title>Top</d:Title>
  <lastUpdate>2008-02-11 19:39:42</lastUpdate>
  <narrow r:resource="Top/Adult"/>
  <narrow r:resource="Top/Arts"/>
  <narrow r:resource="Top/Business"/>
  ...
  <narrow r:resource="Top/Home"/>
  ...
  <narrow r:resource="Top/Society"/>
  <narrow r:resource="Top/Sports"/>
  <narrow r:resource="Top/World"/>
</Topic>
```

**Code 4.1:** ODP-Datenexport – Definition der obersten Themen

Code 4.2 zeigt die Definition eines der Hauptthemen. Bei diesem sieht man zusätzlich einen alternativen Namen für die Anzeige (`dispname`) und für die AOL Suche (`aolsearch`), sowie eine kurze Beschreibung des Themenbereichs.

```
<Topic r:id="Top/Home">
  <catid>7</catid>
  <aolsearch>home and garden</aolsearch>
  <dispname>Home and Garden</dispname>
  <d:Title>Home</d:Title>
  <d:Description>Welcome to the Home categories of the Open Directory
      Project. If you are interested in home improvement, redecorating,
      remodelling or do it yourself home repair, tax preparation,
      consumer information, gardening, apartment living, family and
      parenting sites, recipes or fun places for kids, the Home category
       has the sites for you.
  </d:Description>
  <narrow1 r:resource="Top/Home/Apartment_Living"/>
  <narrow1 r:resource="Top/Home/Consumer_Information"/>
  <narrow1 r:resource="Top/Home/Cooking"/>
  <narrow1 r:resource="Top/Home/Do-It-Yourself"/>
  ...
  <narrow r:resource="Top/Home/News_and_Media"/>
  ...
  <symbolic1 r:resource="Pets:Top/Recreation/Pets"/>
  ...
  <related r:resource="Top/Regional/Europe/United_Kingdom/
      Recreation_and_Sports/Home_and_Garden"/>
  <related r:resource="Top/Science/Social_Sciences/
      Family_and_Consumer_Science"/>
  ...
  <altlang r:resource="German:Top/World/Deutsch/Zuhause"/>
  <altlang r:resource="French:Top/World/Francais/Maison"/>
```

```
  ...
  <lastUpdate>2007-11-01 08:55:33</lastUpdate>
</Topic>

<Alias r:id="Pets:Top/Recreation/Pets">
  <d:Title>Pets</d:Title>
  <Target r:resource="Top/Recreation/Pets"/>
</Alias>
```

**Code 4.2:** ODP-Datenexport – Definition eines Themas im Detail

Es werden wiederum Beziehungen zu untergeordneten Themen definiert. Das Element `narrow1` ist hierbei ähnlich zu `narrow`, hat jedoch eine höhere Priorität. Diese Themen werden in der Web-Oberfläche weiter oben angezeigt. Ein symbolischer Verweis (`symbolic1`) wird definiert. Er verweist auf ein `Alias`-Element weiter unten, in dem spezifiziert wird, dass das Thema „Pets“ auf „Top / Recreation / Pets“ zeigen soll.

Auf verwandte Themen zeigen Querverweise, die mit dem Element `related` definiert werden. Sie werden im Gegensatz zu symbolischen Verweisen nicht als eigene Themen angesehen und besitzen daher auch keinen Namen sondern nur ein Zielthema. Ebenfalls definiert werden mit `altlang` die Verweise zu dem selben Thema in verschiedenen Sprachen.

```
<Topic r:id="Top/Home/Apartment_Living">
  <catid>84890</catid>
  <link r:resource="http://ths.gardenweb.com/forums/apt/"/>
  <link r:resource="http://www.dogbreedinfo.com/apartment.htm"/>
  <link r:resource="http://www.rentaldecorating.com"/>
  <link r:resource="http://www.apartmentreviews.net"/>
  <link r:resource="http://www.apartment-ideas.com"/>
  ...
</Topic>

<ExternalPage about="http://ths.gardenweb.com/forums/apt/">
  <d:Title>Apartment Living</d:Title>
  <d:Description>A discussion forum for those living in apartments,
      condominiums and co-ops. Topics range from roommate problems,
      maintenance issues, leases to decorating.</d:Description>
  <topic>Top/Home/Apartment_Living</topic>
</ExternalPage>
```

**Code 4.3:** ODP-Datenexport – Definition von Verweisen auf Web-Inhalte

In Code 4.3 sieht man schließlich, wie im Inhalt-Dokument die zu den jeweiligen Themen klassifizierten Web-Inhalte spezifiziert werden. Die Ressource wird in einem `link`-Element definiert und anschließend in einem `ExternalPage`-Element noch mit einem Titel und einer Beschreibung versehen. Das Element `topic` gibt an, für welchen Pfad diese Definition gilt. Bei Inhalten, die in mehreren Themen klassifiziert sind, ist diese Zuordnung wichtig.

# Kapitel 5

# Prospector

In Kapitel 1 wurde Prospector bereits vorgestellt und aus Sicht eines Anwenders beschrieben. Nach einern anfänglichen Erläuterung der dem System zugrunde liegenden Idee aus einer anderen Perspektive wird nun zuerst die bisherige Entwicklung von Prospector umrissen. Näher beschrieben werden die ersten beiden Versionen mit ihren jeweiligen Lösungsansätzen, was die Datenquellen, Algorithmen und die Benutzerschnittstelle betrifft.

Nachdem in den Kapiteln 2 und 3 die Grundlagen adaptiver System und personalisierter Suche besprochen wurden, können Idee und Funktionsweise der zweiten Version von Prospector nun auch im Detail behandelt werden. Beschrieben wird somit jener Stand, der als Ausgangspunkt für die im nächsten Kapitel erläuterten Erweiterungen diente. Dabei wird insbesbesondere auf die Mechanismen der Benutzermodellierung und der Personalisierung eingegangen. Die Algorithmen und ihre mathematischen Grundlagen werden ebenfalls erläutert.

## 5.1 Grundidee

Prospector realisiert eine *adaptive Meta-Suche* für Suchmaschinen wie beispielsweise Google. Das System arbeitet auf einer über diesen Suchmaschinen angeordneten Schicht und kapselt sie. Dabei ist Prospector nicht auf Web-Suche beschränkt, sondern modular gestaltet und erweiterbar. Der Algorithmus und die unterstützenden Programmstrukturen sind großteils unabhängig von der darunterliegenden Suchmaschine und den Eigenschaften der durch sie gelieferten Ergebnisse.

Prospector kann mit jedem Suchsystem, für das semantische Informationen zu den Ergebnissen in Form einer Ontologie vorliegen, eingesetzt werden. Ein mögliches Einsatzgebiet wäre beispielsweise mit einem *Suchsystem in Bibliotheken*, das zu jedem Ergebnis auch eine Klassifizierung in einem gewissen Schema liefert. Ebenfalls vorstellbar ist der Einsatz bei der *Bildersuche*, wo Dienste wie iStockphoto[1] eine umfangreiche Kategorisierung der angebotenen Bilder vornehmen. Auch bei *Musiksuchmaschinen* können zu den Titeln mit einer Ontologie zusätzliche Metadaten wie Genre, Herkunftsland oder Entstehungszeit festgelegt sein.

[1] `http://www.istockphoto.com`

Vom Suchsystem übernimmt Prospector die Ergebnisse, reichert sie selbständig mit semantischen Informationen an und personalisiert sie für den Benutzer durch Umreihen. Die Grundlage für diese Personalisierung liefern das Benutzermodell und die Modelle der Gruppen, deren Mitglied der Benutzer ist. Insofern bedient sich Prospector zweier Quellen zum *Verbessern der Reihung* der Suchergebnisse: der semantische Informationen zu den einzelnen Ergebnissen der Suche und der Präferenzen der Benutzer beziehungsweise deren Aggregation in Gruppenmodellen. Details zu den adaptiven Funktionalitäten von Prospector finden sich in Abschnitt 5.4.

## 5.2 Bisherige Entwicklung

Von Prospector gibt es mittlerweile schon die dritte Version. Diese ist es auch, deren Entwicklung in der vorliegenden Arbeit beschrieben wird. Im Folgenden werden die beiden vorangegangenen Versionen, ihre Entwicklung, ihre Neuerungen und deren Evaluierung beschrieben.

### 5.2.1 Version 1 – Erstentwicklung

Den Anstoß zur Entwicklung der ersten Version von Prospector gab eine Übungsaufgabe in der Lehrveranstaltung *Adaptive Web-based Systems* im Wintersemester 2004. Als Projektarbeit sollte ein einfaches adaptives web-basiertes System erstellt werden. Gemeinsam mit meinem Kollegen Christian Schwendtner starteten wir die Entwicklung einer adaptiven Suchmaschine, die wir zu dieser Zeit noch *Google Prospector* nannten. Mehr Details zu den hier angeführten Erklärungen finden sich in der zu dieser Version veröffentlichten Publikation [SKP06].

#### Ziele

Das Ziel der Entwicklung war, *passendere Suchergebnisse* zu liefern. Dies sollte einerseits nach den persönlichen Vorlieben des Benutzers und andererseits situationsbezogen und anonym möglich sein. Benutzer sollten zudem von den Bewertungen anderer Benutzern profitieren und ihr Benutzermodell einsehen und bearbeiten können. Aufgrund der beschränkten Zeit, die für das Projekt zur Verfügung stand, sollte dieses so einfach wie möglich realisiert werden. Nichtsdestotrotz waren bereits die Grundlagen aller wichtigen Funktionalitäten der aktuellen Version vorgesehen.

#### Datenquelle

Die Suchergebnisse lieferte in dieser ersten Version ausschließlich *Google*. Die Wahl fiel auf diesen Anbieter, weil sich zu jener Zeit Google als einzige große Suchmaschine über eine Schnittstelle programmatisch abfragen ließ. Möglich war und ist dies mittels der von Google

selbst veröffentlichten, früher frei erhältlichen *Search API*. Mit dieser Java-Bibliothek lassen sich SOAP-Aufrufe mit Suchanfragen zum Webservice von Google machen.

Als Ergebnis erhält man eine Liste von Ergebnissen mit unter anderem Titel und URL der Seiten, dem kurzen Abriss des Inhalts genannt *Snippet*, sowie einer *Klassifizierung der Seite* in der Ontologie des ODP. Jedes der vier Informationsstücke für sich (aber auch Kombinationen) hätte als Basis für die Modellierung von Benutzerinteressen dienen können. Die Wahl fiel auf die Klassifizierung, da diese mit dem geringsten Aufwand die meiste semantische Information liefert.

Aus der URL ließen sich anhand der Top-level Domain das Land bzw. die Organisationsform (gemeinnützig, kommerziell, akademisch, militärisch, ...) der Ergebnisseite erkennen. Unter Umständen könnten noch Worte zur Klassifizierung der Seite aus der URL extrahiert werden. Wesentlich mehr Worte lassen sich aus dem Titel und dem Snippet extrahieren. Im Gegensatz zur schon bestehenden Klassifizierung durch ODP stellt dies aber einen höheren Aufwand dar und ist zudem unsicherer. Besonders das Snippet wird oft so gestaltet, dass es nicht einen allgemeinen Abriss des Inhalts der Seite wiedergibt sondern Textpassagen rund um die gefundenen Suchbegriffe enthält. Einziger Vorteil, den Titel und Snippet hätten, wäre ihre universelle Verfügbarkeit. Anders als die Klassifizierung durch ODP sind diese Informationen bei jedem Ergebnis vorhanden.

### Algorithmus

In dieser ersten Version verwendete der Algorithmus für die Gewichtungen in den Benutzer- und Gruppenmodellen noch ein *Punktesystem*. Je nach Bewertung durch den Benutzer wurden in den einzelnen Themengebieten der Ontologie, nach denen das Benutzermodell aufgebaut war, Punkte addiert oder subtrahiert. Für die Berechnung der Relevanz eines Ergebnisses wurde die Summe der Punkte jener Themen, in denen das Ergebnis klassifiziert war, genommen. Nicht klassifizierte Suchergebnisse, also Seiten, die nicht durch das ODP erfasst sind, erhielten 0 Punkte. Beim Umreihen der Ergebnisliste wurde zuerst nach der Punktezahl eines jeden Ergebnisses sortiert und bei Punktegleichstand nach der ursprünglichen Reihenfolge von Google.

### Benutzerschnittstelle

Wie von der Übungsaufgabe gefordert, handelte es sich bei der ersten Version von Prospector um eine *Web-Anwendung*. Realisiert wurde sie mit Servlet-Technologie und Java Server Pages (JSP) auf Apache Tomcat und unter Verwendung des Struts Framework zur Erzeugung von Web-Anwendungen nach dem Model-View-Controller (MVC) Prinzip. Die Web-Schnittstelle unterstützte alle zuvor beschriebenen Funktionalitäten von Prospector, eine Login- und Logout-Funktion, Blättern durch die Ergebnisliste und eine Möglichkeit zum Bewerten der Ergebnisse.


**Abbildung 5.1:** Prospector Version 1 – Suchmaske

In Abbildung 5.1 sieht man die einfach gehaltene Suchmaske, Abbildung 5.2 auf der nächsten Seite zeigt eine Liste mit Ergebnissen. Besonders zu beachten sind hierbei die zuvor schon angesprochenen Relevanzpunkte, deren *Wertebereich nicht beschränkt* war und die, wie in Abbildung 5.3 auf der nächsten Seite und Abbildung 5.4 auf der nächsten Seite zu sehen ist, heftig schwanken konnten, sowohl innerhalb einer Ergebnisliste als auch zwischen zwei Listen. Ebenfalls zu sehen in Abbildung 5.3 ist die Auswahl der Gruppen, nach deren Modell (hier „profile“ genannt) umgereiht werden konnte. Diese Umreihung konnte auch wieder aufgehoben werden („disable“), und es wurden, je nachdem ob man eingeloggt war oder anonym suchte, die personalisierte oder die unveränderte Reihung wiederhergestellt.

Die *Bewertung von Suchergebnissen* war auf zwei Arten möglich: in der Ergebnisliste konnten die Benutzer die Schaltfläche „Unsuitable“ mit dem roten Kreuz verwenden, um das jeweilige Ergebnis aus der Liste zu entfernen und negativ zu bewerten (siehe Abbildung 5.3). Klickte man auf den Link im Titel eines Suchergebnisses, so wurde die Seite des Ergebnisses und darüber ein Bereich mit Schaltflächen zum Bewerten angezeigt (siehe Abbildung 5.5 auf Seite 58). Diese Schaltflächen ermöglichten positive und negative Bewertungen und ließen den Benutzer wahlweise zur Ergebnisliste zurückkehren oder entfernten den Bewertungsbereich, wenn er die Suche mit diesem Ergebnis abschließen wollte.

Die Seiten zum Festlegen der *Gruppenzugehörigkeit* und zum Betrachten und Bearbeiten des *Benutzermodells* sieht man in Abbildung 5.6 auf Seite 58 und Abbildung 5.7 auf Seite 59. Auch hier zeigen sich die Gewichtungen in Form von in ihrem Wertebereich nicht beschränkten Werten. Durch Anwahl von „Disable subtree“ in einer Zeile des Baums im Benutzermodell konnten die Benutzer diesen Ast aus dem Modell entfernen.

### Evaluierung

Die erste Version von Prospector wurde einer informellen und auf Gebrauchstauglichkeit (usability) ausgerichteten, *heuristischen Evaluierung* unterzogen. Gleichzeitig benutzen Mitglieder des Instituts das System testweise für einen Tag. Trotz der kurzen Zeit und dem Fehlen einer Kontrollgruppe konnten einige wichtige Erkenntnisse gewonnen werden.

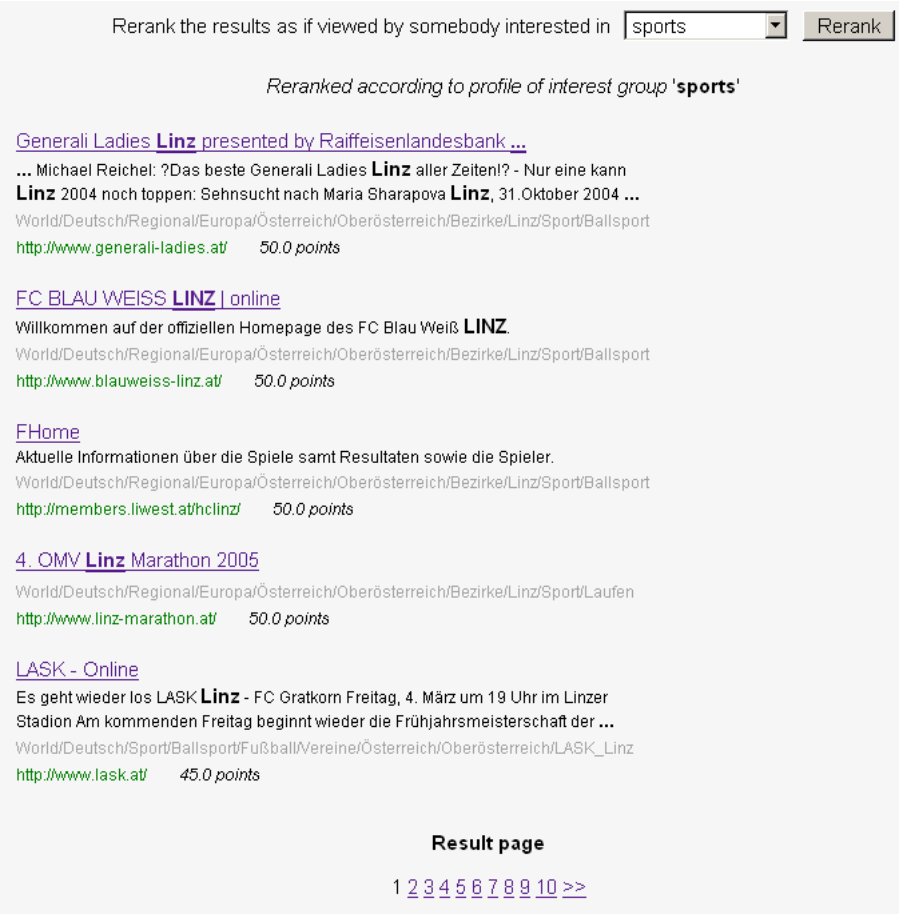


**Abbildung 5.2:** Prospector Version 1 – Ergebnisliste

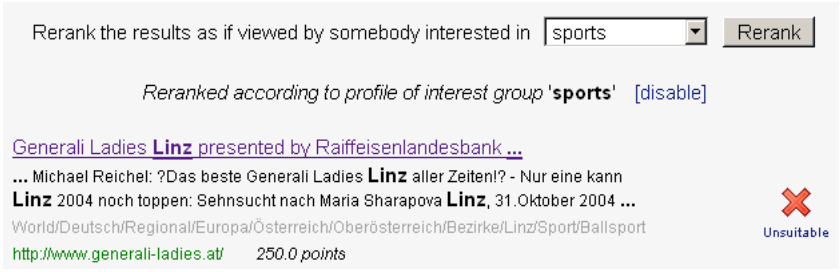


**Abbildung 5.3:** Prospector Version 1 – Unbeschränkter Wertebereich für Relevanzpunkte

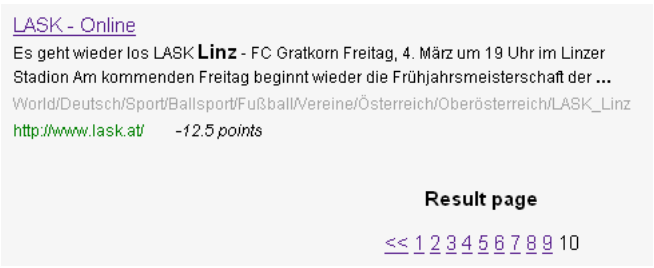


**Abbildung 5.4:** Prospector Version 1 – Negative Punkte markierten Ergebnisse in einer Kategorie, die vom Benutzer zuvor schon negativ bewertet wurde

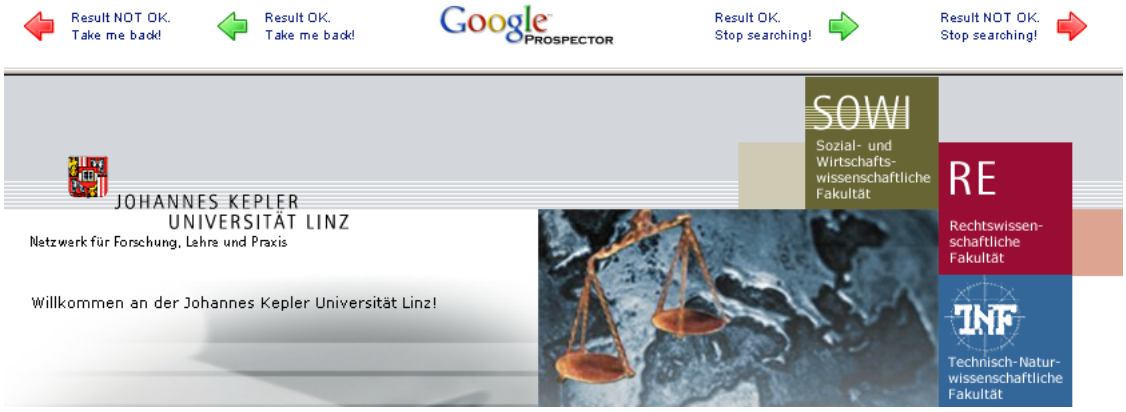


**Abbildung 5.5:** Prospector Version 1 – Ergebnisanzeige mit Möglichkeit zum Bewerten

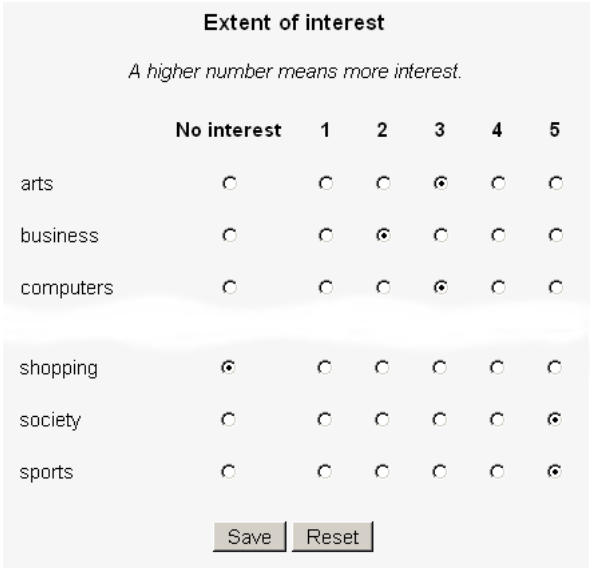


**Abbildung 5.6:** Prospector Version 1 – Stärke der Gruppenzugehörigkeit festlegen

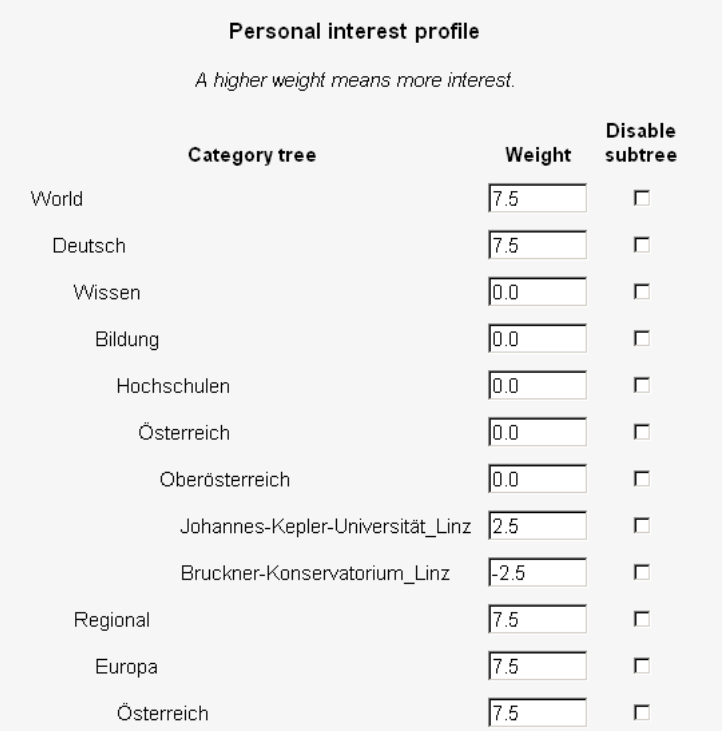


**Abbildung 5.7:** Prospector Version 1 – Benutzermodell betrachten und bearbeiten

Die Benutzer waren im Allgemeinen Prospector gegenüber *positiv* eingestellt und zufrieden mit den Umreihungen. Die Modellierung der Interessen und Berechnung der Relevanz mit in ihrem Wertebereich nicht eingeschränkten Gewichten und Punkten waren hingegen schwer zu verstehen. Die *Werte schwankten*, wie zuvor bereits erwähnt, *stark* und machten einen Vergleich zwischen einzelnen Ergebnissen auf Basis der Punkte schwierig. Die hypothetische Frage „Wie mehr relevant ist eine Ergebnis mit 50 Punkten als eines mit 5 Punkten? Zehn mal so relevant?“ soll dieses Problem illustrieren.

Kritisch war auch, dass dadurch ein *Wertebereich* für die manuelle Modelländerung *fehlte* (siehe Abbildung 5.7). Die Wahl eines Wertes für die Gewichtung einer bestimmten Kategorie war nicht trivial und für Laien kaum zu durchschauen. Darüber hinaus war der genaue Effekt eines Wertes auf den Prozess der Umreihung nicht klar. Die Benutzer fanden auch die Benennung der Gruppen nach den obersten Kategorien der Ontologie des ODP verwirrend.

### 5.2.2 Version 2 – Probabilistischer Ansatz

Die Ergebnisse der Evaluierung der ersten Version von Prospector deuteten auf klare Verbesserungspotenziale hin. Aus diesem Grund wurde eine zweite Version entwickelt, die anstelle der unbeschränkten Gewichte und Punkte mit einem *probabilistischen Ansatz* (Details folgen) arbeitete. Auch bei der Klassifizierung der Ergebnisse gab es Neuerungen, ebenso bei der Benutzerschnittstelle. Diese Version wurde erneut einer (diesmal größer angelegten) Eva-

luierung unterzogen. Mehr Informationen enthält neben den folgenden Erläuterungen auch die entsprechende Publikation [PKSvV08].

### Ziele

Das Hauptziel dieser Version war, die Modellierung und Relevanzberechnung in den *Wertebereich zwischen Null und Eins* zu verlegen und dadurch für die Benutzer in Form von Prozentzahlen leichter verständlich zu machen. Entsprechende Änderungen der Benutzerschnittstelle waren ebenfalls geplant. Mit Feinarbeiten am Algorithmus sollte das Verhalten des Systems zusätzlich durchschaubarer und flexibler werden.

### Datenquelle

Auch in dieser Version stammten die Suchergebnisse von Google und wurden über dessen *Search API* abgefragt. Aufgrund einer Änderung durch Google enthielten die Ergebnisse aber nicht mehr die Klassifizierung nach ODP. In einem ersten Schritt wurde daher versucht, mittels *screen scraping* Daten von der Seite des Open Directory Project `http://www.dmoz.org` zu beziehen. Dabei wurden die Anfragen direkt an das CGI-Skript zur Suche gesendet und die zurückgegebenen HTML-Seite mit einem Parser analysiert. Dieser Ansatz stellte sich aber als wenig effizient, fehleranfällig und nicht zukunftssicher dar.

Wie bereits in Unterabschnitt 4.3.2 erwähnt, bietet ODP die Daten seiner Ontologie in einem zum RDF ähnlichen Format frei zum Herunterladen an. Mit dem Import dieser Daten in eine *lokale Datenbank* war Prospector fortan unabhängig von den Klassifizierungsdaten in den Google-Ergebnissen und der Online-Verfügbarkeit des ODP-Dienstes.

Obwohl Webseiten auch in mehr als einer Kategorie klassifiziert sein können, verwendete der Algorithmus als Vereinfachung in dieser Version von Prospector nur jene Kategorie mit den meisten Einträgen. Der Gedanke dahinter war, dass eine Kategorie mit mehr Einträgen (also mehr Webseiten, die in ihr klassifiziert sind) relevanter ist als jene mit weniger. Wie die Evaluierung zeigte, hielt diese Annahme leider nicht.

### Algorithmus

Der Algorithmus in dieser Version von Prospector basierte auf *Wahrscheinlichkeiten*, intern ausgedrückt in Fließkommazahlen im einschließenden Intervall zwischen Null und Eins und dem Benutzer als Prozentzahl zwischen 0% und 100% angezeigt. Sowohl für die Gewichtungen im Modell als auch die errechneten Relevanzen war die Idee für die Bedeutung der Wahrscheinlichkeit folgende: 0,5 (50%) bedeutet Indifferenz beim Interesse bzw. keine klare Aussage bei der Relevanz. Werte über 0,5 zeigen ein verstärktes Interesse am damit gewichteten Thema an bzw. bezeichnen Ergebnisse mit höherer Relevanz. Bei Werten unter 0,5 ist die Bedeutung umgekehrt.


**Abbildung 5.8:** Prospector Version 2 – Farbliche Unterstützung bei der Angabe der Interessen

Der verbesserte Algorithmus sorgte zusätzlich mit einer *nicht-linearen Veränderung der Wahrscheinlichkeiten* beim Bewerten eines Ergebnisses dafür, dass sich die Gewichte im Modell rund um 0,5 schnell verändern, zu den Rändern hin jedoch immer langsamer. Damit sollte verhindert werden, dass schon länger bestehende Interessen durch kurzfristige Bewertungen unverhältnismäßig stark verändert werden. Noch nicht gewichtete Themen sollten jedoch schnell in eine gewisse Richtung tendieren können. Details zum Algorithmus von Prospector finden sich weiter unten in Abschnitt 5.5.

### Benutzerschnittstelle

Bis auf die farbliche Unterstützung bei der Angabe der Interessen und somit dem Grad der Zugehörigkeit zu einer Gruppe (siehe Abbildung 5.8) waren die meisten Änderungen der Benutzerschnittstelle durch die Umstellung auf den probabilistischen Ansatz bedingt. In Abbildung 5.9 auf der nächsten Seite sieht man die Anzeige der Relevanz nun als Prozentzahl.

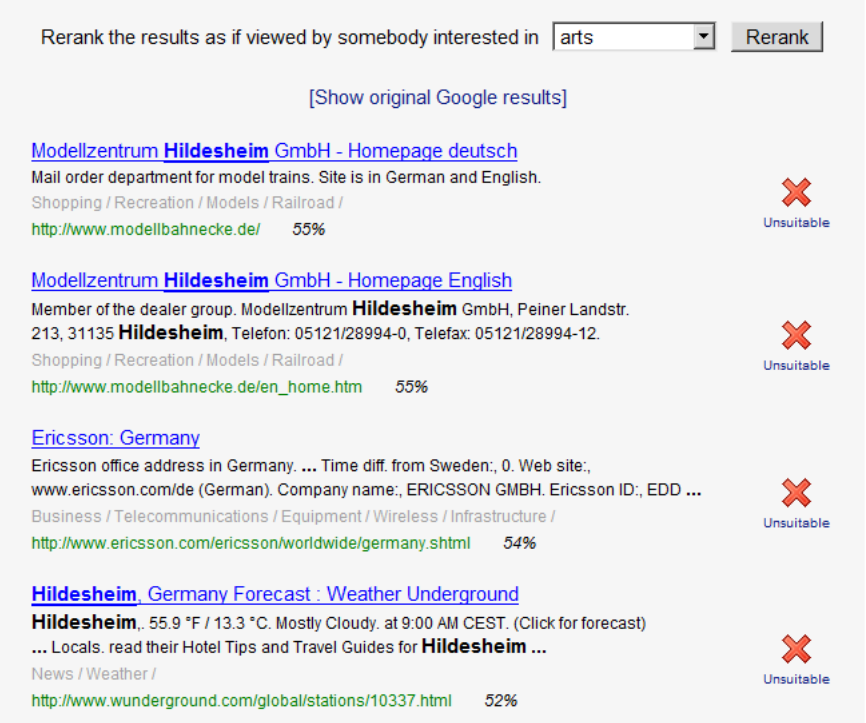


**Abbildung 5.9:** Prospector Version 2 – Anzeige der Relevanz in Prozent

Der nunmehr fixierte Wertebereich bei den Gewichtungen der Themenbereiche im Benutzermodell machte es auch möglich, die Änderung derselben benutzerfreundlicher zu gestalten. Abbildung 5.10 auf der nächsten Seite zeigt die dazu verwendeten Schieberegler, mit der ein Wert von 0% bis 100% eingestellt werden konnte. Ebenfalls neu war, dass die Ebenen der Themen-Hierarchie durch Steuerungselemente, wie man sie von anderen baumartigen Strukturen (z. B. ein Ordnerbaum im Dateimanager) kennt, ein- und ausgeklappt werden konnten.

### Evaluierung

Diese zweite Version von Prospector wurde in den Niederlanden in einer kontrollierten Umgebung (Labor) evaluiert (Details zur Evaluierung finden sich in Abschnitt 7.1). Eingebettet war diese Evaluierung in eine größere Studie zur Evaluierung von adaptiven Systemen. Die *Ziele* speziell für das Prospector-Projekt waren, zu erforschen, ob und warum die personalisierte Suche effektiv ist. Weiters sollten Probleme mit der Benutzerschnittstelle und dem Interaktionsdesign sowie jene bei der Implementierung der Personalisierungsfunktionen identifiziert werden.

Die *Ergebnisse* der Evaluierung gaben gute Hinweise, wo Prospector noch weiter verbessert werden musste. Die Mehrheit der Teilnehmer hatte grundsätzlich eine positive Einstellung gegenüber der Personalisierung. Sie sahen den Mehrwert vor allem für persönliche Suchen und Suchen nach etwas, das sie wirklich interessiert. Die Informationen zu ihren Interessen

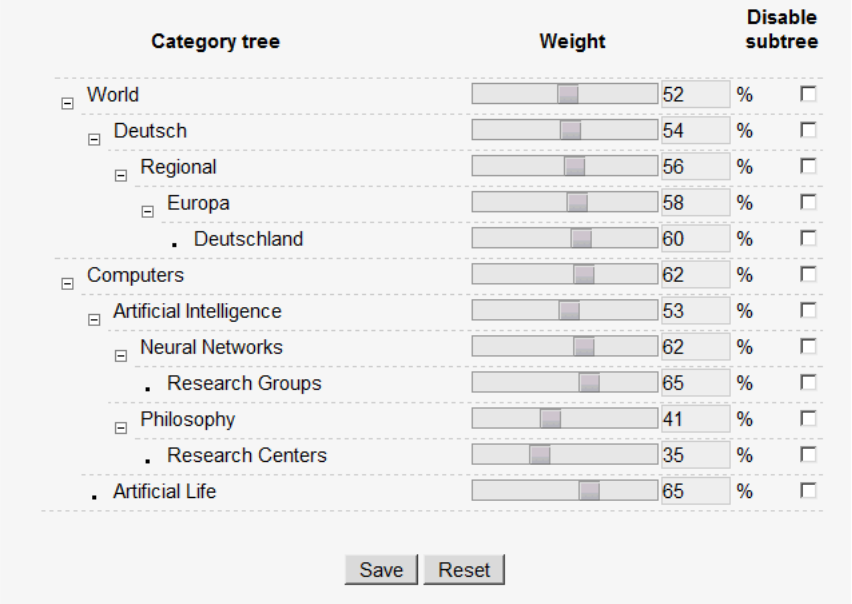


**Abbildung 5.10:** Prospector Version 2 – Modellbearbeitung mit Schiebereglern

im Modell erachteten sie als nicht sensibel und sahen daher ihre Privatsphäre auch nur wenig beeinträchtigt.

*Probleme* bereitete der Bereich zum Bewerten von Ergebnissen (siehe Abbildung 5.5 auf Seite 58). Dieser war als HTML-Frame gestaltet und manche Seiten (z. B. Wikipedia) verhinderten seine Anzeige, weil sie sich selbst als oberste Seite im Browserfenster setzten („frame breakout“). Darüber hinaus wurde der Bereich von einigen Benutzern nicht korrekt oder überhaupt nicht verwendet. Es konnte beobachtet werden, wie Benutzer die „Zurück“-Schaltfläche des Browsers nutzten, um von einer Ergebnisseite zurück zur Ergebnisliste zu gelangen, und anschließend die Schaltfläche „Unsuitable“ wählten, um das Ergebnis negativ zu bewerten.

Schwierigkeiten gab es auch beim Verständnis der Skala zum Angeben es Interesses an einer Gruppe. Ein Teilnehmer verstand diese nicht als Abstufung des Interesses von wenig bis viel sondern von Desinteresse bis starkes Interesse. Auch die Prozentzahlen bei der Skala im Benutzermodell sorgten für Verwirrung. Ebenfalls als problematisch identifiziert werden konnte die Heuristik, die für jede klassifizierte Seite nur das relevanteste Thema liefern soll. Sie funktionierte nicht immer wie gewünscht und führte zu offensichtlichen Fehlklassifikationen.

Insgesamt konnten viele Punkte gefunden werden, die verbesserungswürdig waren. Einige davon wurden in der dritten Version von Prospector behoben. An den jeweiligen Stellen wird daher auf die Evaluierung zurück verwiesen werden.

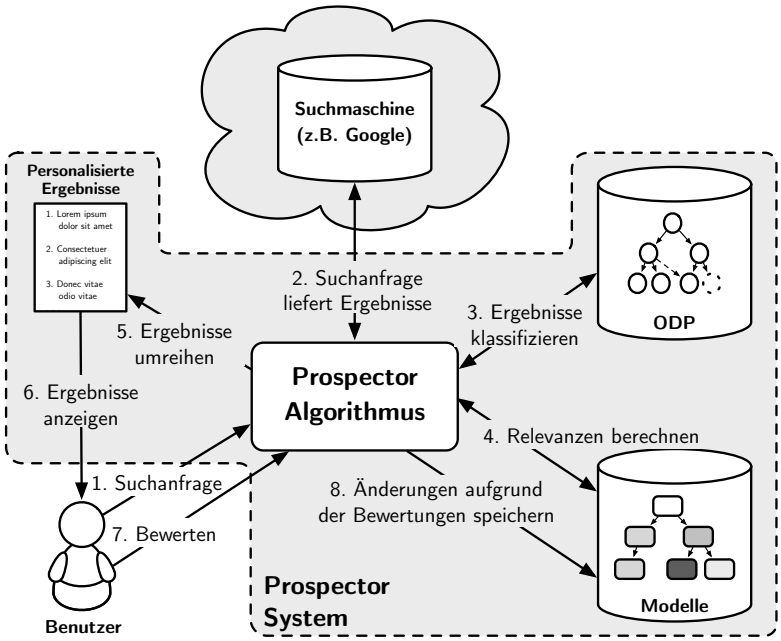


**Abbildung 5.11:** Grundlegender Ablauf von Suchanfragen und Bewertungen

## 5.3 Suchablauf

Wichtig für die tiefergehenden Erklärungen zu Aufbau und Funktion von Prospector ist das Verständnis des grundlegenden Ablaufs von Suchanfragen und Bewertungen. Im Folgenden werden diese, unterstützt durch Abbildung 5.11 Schritt für Schritt beschrieben:

1. *Suchanfrage stellen*: Der Benutzer übermittelt an Prospector eine Suchanfrage. Im Fall der existierenden Implementierung erfolgt dies über eine Web-Oberfläche. Die Suchanfrage enthält die einzelnen Suchbegriffe, die in den Ergebnissen vorkommen sollen.

2. *Suchanfrage weiterleiten*: Prospector leitet die Suchanfrage an eine Suchmaschine (z. B. Google) weiter und erhält die unpersonalisierten Suchergebnisse.

3. *Ergebnisse klassifizieren*: Für jedes Ergebnis wird eine Klassifikation anhand einer Ontologie versucht, sofern die Suchmaschine derartige Daten nicht bereits zur Verfügung gestellt hat. Zur Zeit greift Prospector auf die Daten des Open Directory Project zurück.

4. *Relevanzen berechnen*: Anhand der Klassifikation und den in den Modellen verzeichneten Benutzerinteressen wird für jedes Ergebnis dessen Relevanz berechnet.

5. *Ergebnisse umreihen*: Die Ergebnisliste wird so umgereiht, dass die Ergebnisse mit höherer Relevanz weiter oben stehen.

6. *Ergebnisse anzeigen*: Das Resultat der Suche wird dem Benutzer präsentiert. In der Web-Oberfläche werden die Ergebnisse in der vielseits verwendeten Form mit Seitentitel als Link, Kurzabriss der Seite und URL angezeigt. Informationen zur Klassifizierung und der berechneten Relevanz werden ebenfalls ausgegeben.

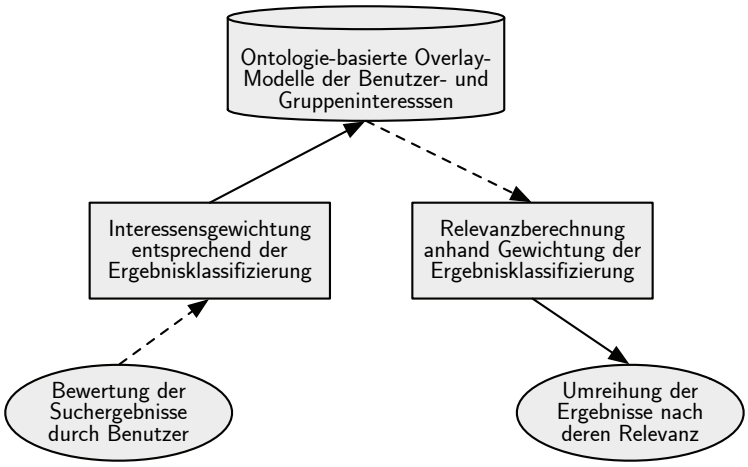


**Abbildung 5.12:** Verarbeitungsschema in Prospector nach Jameson [Jam03]

7. *Bewerten*: Der Benutzer bewertet eines oder mehrere Ergebnisse positiv oder negativ.

8. *Änderungen aufgrund der Bewertungen speichern*: Die Bewertungen werden in das Benutzermodell und relevante Gruppenmodell eingearbeitet, um die damit ausgedrückte Interessenslage abzubilden.

## 5.4 Adaptivität

In Abschnitt 2.1 wurde bereits Jamesons [Jam03] *Schema des Verarbeitungsablaufs* in einem Benutzer-adaptiven System vorgestellt. Gemäß diesem Schema können auch die adaptiven Funktionen von Prospector schematisch dargestellt werden (siehe Abbildung 5.12). Die Eingaben sind die positiven bzw. negativen Bewertungen des Benutzers, die Ausgabe eine umgereihte Ergebnisliste; beides durch Ovale dargestellt. In den Rechtecken finden sich die Benutzermodellierung und die Personalisierung, der Zylinder stellt die gespeicherten Modelle dar. Entlang gestrichelten Pfeilen werden Informationen wie Bewertungen und Gewichtungen übergeben, entlang durchgezogenen Pfeilen Ergebnisse wie Gewichtungsänderungen und berechnete Relevanzen.

Gemäß dem in Abschnitt 2.1 beschriebenen *Schema von Dieterich et al.* [DMKSH93] mit den zwei möglichen Akteuren (System, Benutzer) und seiner Aufteilung des Adaptionsprozesses in vier Schritte (Initiative, Vorschläge, Entscheidung, Ausführung) lässt sich Prospector als System mit Selbst-Adaption klassifizieren. Der Benutzer muss, sofern er angemeldet ist, nicht explizit die Personalisierung aktivieren, die Initiative liegt also beim System. Auch die Vorschläge zur Art der Adaption stammen in Form der Benutzer- und Gruppenmodelle vom System. Die Entscheidung für das Benutzermodell und die entsprechenden Gruppenmodelle wird standardmäßig durch das System getroffen. Die Ausführung der Adaption, das eigentliche Umreihen, übernimmt ebenfalls das System.

Beim *Fokussieren der Suche* besteht hingegen für den Benutzer eine Möglichkeit, in diesen automatischen Prozess einzugreifen. Die Initiative übernimmt weiterhin das System, da es die Möglichkeit zur Adaption eigenständig anbietet. Es liefert auch die Vorschläge für die konkrete Adaption, indem es die Namen der Gruppen als mögliche Fokussierungen auf den durch sie repräsentierten Interessensbereich zur Auswahl stellt. Die Entscheidung, gemäß welchem Gruppenmodell die Umreihung tatsächlich stattfinden soll, wird durch den Benutzer gefällt. Die Ausführung übernimmt wiederum das System.

Anhand der von Jameson [Jam03] genannten erstrebenswerten Eigenschaften für Benutzeradaptive Systeme lassen sich die Ansätze von Prospector grundlegend bewerten. Eine umfassendere Evaluierung findet sich in Kapitel 7.

- *Berechenbarkeit* (predictability): Sofern vorhanden wird die Klassifikation der Ergebnisse innerhalb der Ontologie bei der Ausgabe angezeigt. Der Benutzer kann somit bei einer Bewertung abschätzen, welche Teile seines Interessensmodell wie beeinflusst werden. Da von Suchbegriffen selten auf die gesamte Ergebnisliste geschlossen werden kann und sich diese aufgrund der Dynamik des Internet auch ständig verändert, ist die Berechnbarkeit in Bezug auf die eigentlichen Ergebnisse gering. Lediglich die Tendenzen der Umreihungen lassen sich anhand des Benutzermodells prinzipiell vorhersagen.
- *Verständlichkeit* (transparency): Das Anzeigen der berechneten Relevanzen bei den Suchergebnissen und die Möglichkeit, das Benutzermodell einzusehen, sollen den Benutzern helfen, das System und seine Funktionsweise zu verstehen.
- *Steuerbarkeit* (controllability): Durch ein Ändern der Gewichtungen im Benutzermodell und der Affinitäten zu den Gruppen kann der Benutzer seine Interessen anpassen und die Umreihung beeinflussen. Wenig Einfluss hat er jedoch auf den Inhalt der Gruppenmodelle.
- *Unaufdringlichkeit* (unobtrusiveness): Prospector verlangt nur insofern Aufmerksamkeit, als der Benutzer freiwillig Ergebnisse bewerten kann und beim Registrieren seine Interessen angeben muss.
- *Privatheit* (privacy): Durch die Bewertungen wird ein Modell der Interessen des Benutzers aufgebaut und so eventuell seine Privatsphäre verletzt.
- *Erlebnisreichtum* (breadth of experience): Durch das Umreihen werden zwar keine Ergebnisse entfernt, doch mögliche interessante, nicht zum Benutzermodell passende Ergebnisse können so aus dem unmittelbaren Blickfeld nach hinten verschoben werden und die Vielfalt herabsetzen.

### 5.4.1 Benutzermodellierung

Prospector verwaltet am Server gespeicherte, *dynamische Benutzermodelle* [GSCM07], die als Benutzerdaten das *Interesse* an bestimmten Themengbieten modellieren [MGSG07] und

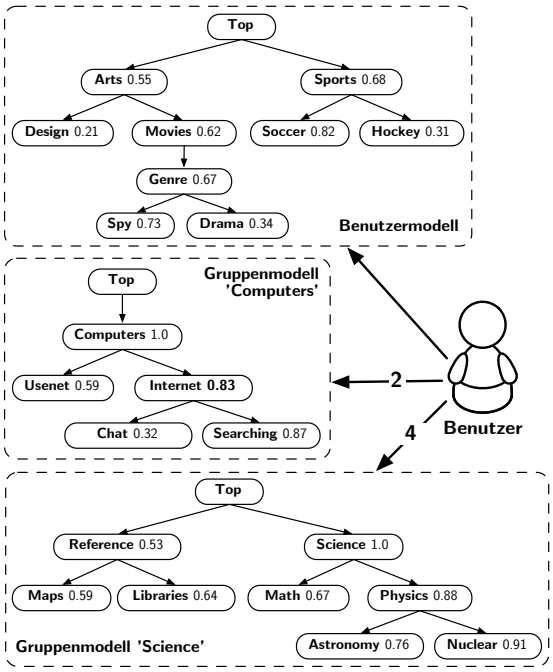


**Abbildung 5.13:** Benutzermodell und Gruppenmodelle mit Affinität $> 0$

während des Systemlaufs („online“) aktualisiert werden. Die Struktur entspricht dem in Abschnitt 3.2 erwähnten *Taxonomie-Ansatz*, und das Modell ist ein *Overlay* über eine bestehende Modellierung des Anwendungsgebiets, in diesem Fall die Ontologie des ODP. In Abbildung 5.13 werden die Komponenten der Benutzermodellierung beispielhaft dargestellt.

Die gespeicherten Werte waren in der ersten Version von Prospector numerisch; nun entsprechen sie Wahrscheinlichkeiten des Interesses und enthalten somit eine *Ungewissheitskomponente* [BM07]. Die Gewichtung 0 entspricht hierbei überhaupt keinem Interesse, 1 bedeutet volles Interesse und 0,5 wird verwendet, wenn keine Informationen zum Interesse vorliegen oder keine Präferenzen bestehen.

### Gruppenmodelle

Nach dem gleichen Prinzip sind Gruppenmodelle aufgebaut. Es existiert eine endliche Zahl solcher Modelle, und sie enthalten die gemeinsamen Interessen ihrer Gruppenmitglieder. Benannt sind diese *thematischen Modelle* nach Themengebieten der darunterliegenden Ontologie, also jener des ODP. Gewählt wurden hierzu die Themengebiete oberster Ebene, mit Ausnahme von „Adult“, „Reference“, „Regional“ und „World“.

Verbunden mit den Gruppenmodellen sind die Benutzer über ihre *Affinität* zu diesen Themengebieten. Auf einer Skala von 0 bis 5 können Benutzer zu jedem Gebiet und somit zur entsprechenden Gruppe ihr Interesse angeben. 0 bedeutet kein Interesse; der Benutzer ist dann nicht Mitglied der Gruppe. Die Affinitäten von 1 bis 5 stehen für wenig bis sehr großes Interesse. Mit den Affinitäten wird der Einfluss der Bewertungen eines einzelnen Benutzers

auf das Gruppenmodell gesteuert. Umgekehrt geben sie auch die Stärke des Einflusses an, den die Gewichtungen im Gruppenmodell auf die Relevanzberechnungen bei der Personalisierung haben. Skaliert wird in beiden Fällen mit dem Anteil, den der Benutzer am kumulierten Interesse aller Gruppenmitglieder hat. Somit ist gewährleistet, dass bei steigender Gruppengröße die Stärke der Änderungen normiert ist und nicht ebenfalls steigt.

Zum Einsatz kommen die Gruppenmodelle auf zwei Arten: beim Speichern von Bewertungen und beim Berechnen der Relevanz von Suchergebnissen. In beiden Fällen werden im Benutzermodell fehlende Gewichtungen von Themengebieten aus den Gruppenmodellen abgeleitet, so sie dort vorhanden sind. In diesem Sinne handelt es sich um eine *Kombination von merkmalsbasierter Benutzermodellierung mit Stereotypen.* Die stereotypisch verwendeten Gruppenmodelle dienen zur Initialisierung der individuellen, merkmalsbasierten Benutzermodelle und lösen so das Problem des leeren Benutzermodells nach einer Neuregistrierung [BM07].

#### Informationen zu den Benutzern sammeln

In der Web-Schnittstelle von Prospector wird zur Identifizierung der Benutzer auf eine *explizite Anmeldung* und auf *Cookies* gesetzt. Die Anmeldung sorgt für die nötige Verlässlichkeit, und durch die Cookies erfolgt die weitere Identifizierung in der Arbeitssitzung bzw. auch darüber hinaus transparent für den Benutzer. Außerdem kann der Benutzer so Prospector von mehreren Rechnern aus benützen. Laut Gauch et al. [GSCM07] stellt diese Kombination einen guten Kompromiss dar.

Die konkreten Daten für die Modelle erhält Prospector durch *implizite und explizite Informationsgewinnung* beim Benutzer (Unterscheidung in Unterabschnitt 2.2.3 und Unterabschnitt 3.2.4). Beim expliziten Typ werden zwei von Jameson [Jam03] beschriebene Methoden in unterschiedlichen Bereichen eingesetzt:

- *Selbstbeurteilung zu allgemeinen Bereichen*: die Benutzer können, wie oben beschrieben, auf einer Bewertungsskala zu den von den Gruppen repräsentierten Themengebieten ihre Interessen angeben. Durch die Allgemeinheit der Bereiche hält sich der kognitive Aufwand in Grenzen, doch unklare Bedeutungen zeigten sich in der Praxis bereits als Problem.

- *Selbstbeurteilung durch spezifische Bewertungen*: die einzelnen Ergebnisse können durch die Benutzer positiv oder negativ bewertet werden. Hierbei kann der kognitive Aufwand höher sein, und erschwerend kommt auch hinzu, dass die Bewertung nicht ein notwendiger Teil der eigentlichen Suche ist.

Implizit gibt der Benutzer eine Bewertung ab, wenn er ein Ergebnis aus der Liste entfernt, weil es „unpassend“ erscheint. Bei dieser für den Benutzer nicht als eigentliche Bewertung erkennbaren Vorgang wird im System eine negative Rückmeldung registriert.

#### Optimierungen

Das in Unterabschnitt 3.2.3 bereits erwähnte Problem der *sparsity* [AM05] muss auch in Prospector berücksichtigt werden. Würde für jedes Ergebnis nur das tatsächliche Thema, in dem es in der ODP-Ontologie klassifiziert ist, beim Speichern einer Bewertung oder dem Berechnen einer Relevanz herangezogen, käme es selten zu „Treffern“. Die Chancen, dass zwei Seiten eines Ergebnisses bei der großen Menge an möglichen Themengebieten in dem selben klassifiziert sind, ist äußerst gering. Eine Möglichkeit wäre, nur die obersten Ebenen des großteils hierarchisch strukturierten ODP-Verzeichnisses zu verwenden. Dieser Ansatz, der bereits in einigen System umgesetzt wurde [GSCM07], führt aber nur zu einer sehr groben Modellierung der Interessen.

Um alle Ebenen der ODP-Ontologie für das Overlay verwenden zu können und gleichzeitig ein ansonsten spärlich besetztes Benutzermodell zu vervollständigen, wendet Prospector zwei Methoden an. Die eine wurde zuvor schon beschrieben: *Gewichtungen* für Themengebiete im Overlay werden aus den Gruppenmodellen *abgeleitet*, wenn sich im Gruppenmodell keine Informationen finden. Bei der anderen wird die Gewichtung für ein Thema aus den Gewichtungen übergeordneter Themen abgeleitet. Hierbei wird die Tiefe in der Ontologie berücksichtigt und je weiter das zu gewichtende Thema von jenem mit bekannter Gewichtung entfernt ist, umso mehr tendiert das schlussendliche Gewicht gegen den neutralen Wert 0,5 und drückt damit die steigende Unsicherheit der Ableitung aus. Dieser Ansatz kann als eine mögliche Implementierung für das von Busilovsky und Millan [BM07] beschriebene Konzept der *interconcept interest propagation* (siehe auch Unterabschnitt 3.2.3) gesehen werden.

### 5.4.2 Personalisierung

Die Personalisierung in Prospector geschieht durch die in Abschnitt 3.3 erwähnte Methode des *Umreihens der Ergebnisse*, die von der darunterliegenden Suchmaschine als Antwort auf die Suchanfrage des Benutzers zurückgegeben wurden. Die Reihung erfolgt absteigend nach der Relevanz, die für jedes einzelne Ergebnis anhand des Benutzermodells und/oder der Gruppenmodelle berechnet wurden. Bei gleichen Relevanzen wird nach der ursprünglichen Reihenfolge gereiht. Ansonsten spielt diese, entgegen einer Empfehlung von Keenoy und Levene [KL05], keine Rolle.

Der Benutzer hat die Wahl zwischen *zwei Arten der Adaption*: für ihn individuell personalisiert oder auf das Modell einer bestimmten Gruppe fokussiert. Bei der ersten Möglichkeit werden das Benutzermodell und die Modelle aller Gruppen, für die der Benutzer eine Affinität größer als 0 angegeben hat, berücksichtigt. Alternativ kann er bei der zweiten Möglichkeit den Themenbereich einer bestimmten Gruppe (z. B. „Sports“) wählen und nur nach deren Modell umreihen lassen. Diese Vorgehensweise erlaubt es dem Benutzer, die Personalisierung selektiv anzuwenden und für spezielle Suchen mit der Wahl eines Themenbereichs den nötigen Kontext zu schaffen.

Hinter der *Fokussierung der Suche* mit einem Gruppenmodell steht die Annahme, dass nur jene Benutzer Mitglieder einer Gruppe sind, die auch wirklich Interesse an deren Thema haben. Über den Grad ihrer Affinität beeinflussen sie die Gewichtungen in diesem Gruppenmodell und erzeugen somit das Interessensprofil eines fiktiven Benutzers mit Interesse an diesem Thema. Der Umfang des Modells ist dabei nicht auf Themen unterhalb des gleichnamigen Themas erster Ebene im ODP-Verzeichnis beschränkt. In dem Gruppenmodell für „Arts“ können also nicht nur Themen unter „Arts“ mit ihren Gewichtungen verzeichnet sein, sondern beliebige andere Zweige.

Wie schon im vorangegangenen Unterabschnitt beschrieben, kann das Overlay im Benutzermodell bei der Verwendung großer Grundstrukturen wie beispielsweise die des ODP sehr weitläufig werden und viele spezialisierte Themen enthalten. Gauch et al. [GSCM07] schlagen hier vor, beim Bewerten von Suchergebnissen auf der Basis der Informationen im Modell auf Informationen in höheren Ebenen der Themenhierarchie zurückzugreifen und deren Gewichtung abzuleiten. Analog zum oben beschriebenen Ableiten von Gewichtungen beim Aufbau der Modelle können auch beim Berechnen der Relevanz fehlende Gewichtungen aus höheren Ebenen des Modells abgeleitet werden.

## 5.5 Algorithmen

Die in Prospector angewendeten Strukturen und Konzepte wurden bislang aus der Sicht des Benutzers und in diesem Kapitel detaillierter im Hinblick auf die Benutzermodellierung betrachtet. Vor diesem Hintergrund werden nun die konkreten Algorithmen und ihre Wirkung auf die Modelle beschrieben.

### 5.5.1 Initialzustand der Modelle

Beim ersten Start des Systems werden die *Gruppenmodelle* angelegt. Für jedes Themengebiet oberster Ebene des ODP mit Ausnahme von „Adult“, „Reference“, „Regional“ und „World“ wird ein Modell erzeugt. In diesem wird ein gleichnamiger Overlay-Knoten mit der Gewichtung 1,0 erzeugt. Mit dieser Maßnahme wird dem „cold start“-Problem [AM05] entgegengewirkt (siehe auch Unterabschnitt 3.2.3). Die Modelle sind dann nicht einfach leer sondern enthalten eine gewisse Initialbefüllung. Konkret hilft dies in Zusammenhang mit der Art, wie die Gruppenmodelle verwendet werden, gegen das Latenzproblem. Dieses tritt auf, wenn ein neu registrierter Benutzer mangels erkennbarer Personalisierung schnell frustriert ist und beschließt, das System nicht mehr zu verwenden, noch bevor sich ein geeignetes Benutzermodell bilden konnte. Durch die Initialisierung der Gruppenmodelle verfügt das System allerdings schon zu Beginn über Möglichkeiten, Relevanzen für Suchergebnisse zu berechnen.

Ein *Benutzermodell* wird beim Registrieren eines neuen Benutzers leer angelegt. Einzig ein virtueller Wurzelknoten, unter dem die Knoten für die Gewichtung der Themengebiete oberster

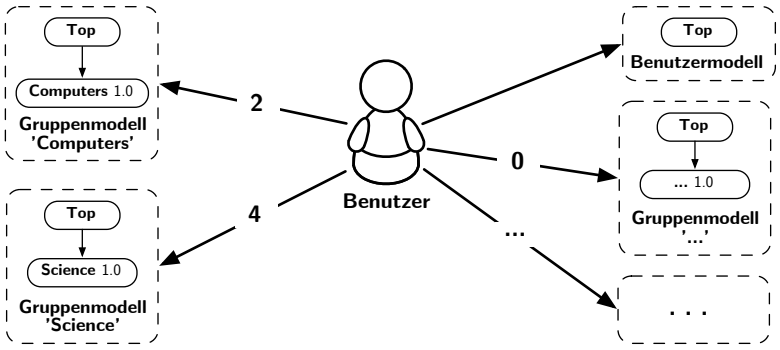


**Abbildung 5.14:** Benutzer- und Gruppenmodelle nach der Initialisierung

Ebene liegen, wird erzeugt. Gleich nach dem Registrieren bekommt der Benutzer die Möglichkeit, seine Affinität zu den einzelnen Gruppen bekannt zu geben. Nachdem diese gespeichert wurden, sind die Modelle für den neuen Benutzer fertig initialisiert. Zu sehen ist eine derartige Konfiguration mit noch „leeren“ Gruppenmodellen in Abbildung 5.14.

### 5.5.2 Ergebnisbewertung durch Benutzer

Gibt der Benutzer wie in 5.4.1 beschrieben eine Bewertung eines Suchergebnisses ab, so muss diese Information zu seiner Interessenslage gespeichert werden. Von dem Ergebnis ist, dank der vorhergehenden Klassifikation, bekannt, welchem Themengebiet der verwendeten Ontologie es zugeordnet ist. Im folgenden Beispiel handelt es sich bei dem bewerteten Ergebnis um eine Seite über Programmiersprachen, deren Klassifizierung über den Pfad „Computers / Programming / Languages“ angegeben wird.

Das Speichern der Bewertung erfolgt in drei Schritten:

1. *Pfad im Modell erzeugen*: Jene Teile des Pfades, für die noch keine Gewichtungen im Modell gespeichert sind, werden als Knoten mit leerer Gewichtung erzeugt.
2. *Gewichtungen ableiten*: Für die Knoten mit leeren Gewichtungen werden aus den Gruppenmodellen und Gewichtungen übergeordneter Knoten passende Werte abgeleitet.
3. *Gewichtungen anpassen*: Entsprechend der Bewertung (positiv oder negativ) werden die Gewichtungen eines jeden Knotens des Pfades vermindert oder erhöht.

Dargestellt ist der Ablauf für eine positive Bewertung beispielhaft in Abbildung 5.15 auf der nächsten Seite. Die genauen Vorgänge in Schritt zwei und drei werden im Folgenden beschrieben.

#### Ableitung von Gewichtungen

Bei der Ableitung von Gewichtungen für einen Themen-Knoten im Modell wird versucht, aus der Gewichtung des gleichlautenden Themas in den diversen Gruppenmodellen, zu denen

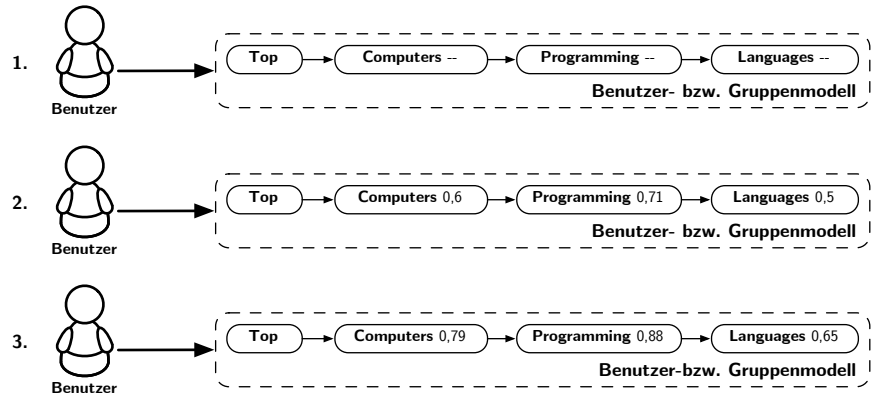


**Abbildung 5.15:** Bewertung speichern: 1. Pfad erzeugen, 2. Gewichtungen ableiten, 3. Gewichtungen verändern (Hierarchie ausgeflacht zu besseren Darstellung)

der Benutzer eine positive Affinität hat, und den Gewichtungen übergeordneter Knoten einen *Wert vorherzusagen*. Dies Vorgehensweise wurde bereits in Unterabschnitt 5.4.2 vorgeschlagen. Die Berechnung der abgeleiteten Gewichtung $G_a(b,t)$ für einen Benutzer $b$ und ein Thema $t$ erfolgt gemäß Formel 5.1.

$$G_a(b,t) = \begin{cases} 0,5 & G_g(b,t) = 0 \wedge G_v(b,t) = 0 \\ G_g(b,t) & G_g(b,t) > 0 \wedge G_v(b,t) = 0 \\ G_v(b,t) & G_g(b,t) = 0 \wedge G_v(b,t) > 0 \\ \lambda \cdot G_g(b,t) + (1-\lambda) \cdot G_v(b,t) & \text{andernfalls} \end{cases} \tag{5.1}$$

Grundlage der Berechung sind die *abgeleitete Gewichtung* aus den Gruppenmodellen $G_g(b,t)$ und die von den übergeordneten Knoten *vererbte Gewichtung* $G_v(b,t)$. Beide Funktionen liefern 0, wenn keine Gewichtung bestimmt werden konnte. Im Fall von $G_g(b,t)$ passiert dies, wenn der Benutzer zu keiner Gruppe eine positive Affinität hat, oder in keinem Modell seiner Gruppen das entsprechende Thema gefunden wurde. $G_v(b,t)$ liefert hingegen 0, wenn kein übergeordnetes Thema im Benutzermodell gespeichert ist, von dem eine Gewichtung abgeleitet werden könnte.

Wenn beide Funktionen 0 zurückgeben, so liegt keine Information vor, auf der die Ableitung der Gewichtung basieren könnte. In diesem Fall wird auf die neutrale Gewichtung 0,5 zurückgegriffen. Liefert eine der zwei Funktionen einen Wert ungleich 0, so wird dieser als abgeleitete Gewichtung verwendet. Liefern beide eine positiven Wert, so werden sie mit dem Faktor $\lambda$ gewichtet summiert. $\lambda$ kann im Wertebereich [0, 1] gewählt werden. Bis jetzt wurde der Faktor auf 0,75 gesetzt und damit den Gruppenmodellen mehr Gewicht eingeräumt. Eine experimentelle Überprüfung dieser Wahl ist noch ausständig.

Die Berechnung von $G_v(b,t)$ erfolgt gemäß Formel 5.2 auf der nächsten Seite. Die Gewichtungen $G(b,t_i)$ der übergeordneten Themen $t_i$ werden mit ihrer $Tiefe(t_i)$ im Modell im Verhältnis zur $Tiefe(t)$ des betrachteten Themas gewichtet und aufsummiert. Die Tiefe berechnet sich

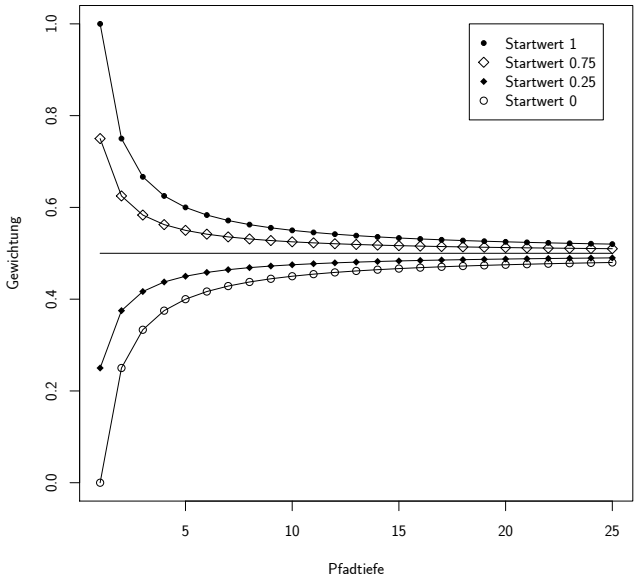


**Abbildung 5.16:** Abgeleitete Gewichte in Abhängigkeit der Pfadtiefe und der Gewichtung des obersten Knotens als Startwert. Klar erkennbar ist die asymptotische Annäherung an die neutrale Gewichtung 0,5.

anhand der Länge des Klassifikationspfads: $Tiefe(Top/Computers/Programming) = 3$. Die Summe wird durch die Zahl der übergeordneten Knoten $N$ dividiert und somit genormt. Die Summenterme mit 0,5 haben den Zweck, mit steigender Pfadtiefe die abgeleitete Gewichtung immer näher an die neutrale Gewichtung 0,5 zu bringen (siehe Abbildung 5.16). Dies soll ausdrücken, dass mit steigender Tiefe der Ableitung die Unsicherheit bezüglich der Vorhersage des Werts zunimmt.

$$G_v(b,t) = \frac{\sum_{i=1}^{N} \left( (G(b,t_i) - 0{,}5) \cdot \frac{Tiefe(t_i)}{Tiefe(t)} + 0{,}5 \right)}{N} \tag{5.2}$$

Die Ableitung aus den Gruppenmodellen $G_g(b,t)$ wird mit Formel 5.3 berechnet. Die Gewichtungen $G(g_i,t)$ des Themas $t$ in den Modellen all jener Gruppen $g_i$, zu denen der Benutzer eine positive Affinität haben, werden nach der Affinität geteilt durch die maximale Affinität 5 gewichtet aufsummiert und anschließend durch die Zahl $N$ dieser Gruppen geteilt. Die Summenterme mit 0,5 haben wiederum den Zweck bei sinkender Affinität positive wie auch negative Gewichtungen in Richtung 0,5 zu bewegen.

$$G_g(b,t) = \frac{\sum_{i=1}^{N} \left( (G(g_i,t) - 0{,}5) \cdot \frac{Affinität(b,g)}{5} + 0{,}5 \right)}{N} \tag{5.3}$$

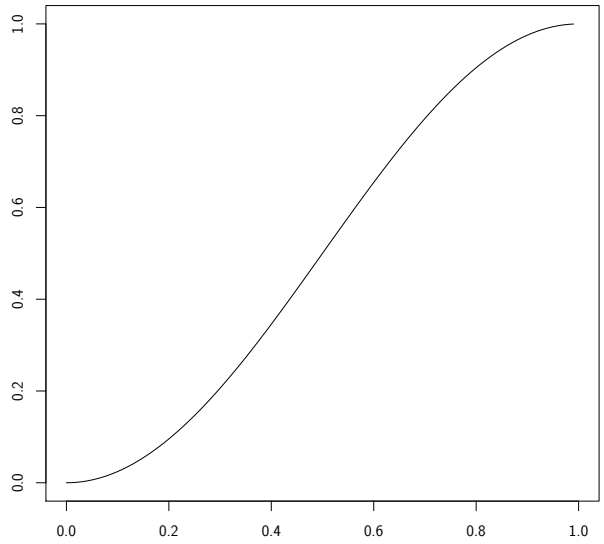


**Abbildung 5.17:** Nicht-lineare Funktion zur Veränderung von Gewichtungen

### Veränderung von Gewichtungen

Positive bzw. negative Bewertungen werden durch Erhöhen bzw. Vermindern von Gewichtungen modelliert. Da die Gewichtung eines Themas die Wahrscheinlichkeiten des Interesses an ihm abbilden soll, muss sie im Wertebereich zwischen 0 und 1 gehalten werden. Ein primitiver Ansatz wäre, die Gewichtung immer um einen fixen Wert (z. B. 0,1) zu verändern, außer sie hat 0 oder 1 erreicht. Eine zweite wünschenswerte Eigenschaft lässt sich damit allerdings nicht realisieren: die Werte sollen sich um 0,5 schneller ändern, gegen 0 bzw. 1 aber immer mehr Bewertungen benötigen. Damit sollen unbewertete Themen schnell als interessant bzw. uninteressant eingestuft werden können, aber auf längere Zeit relativ stabil so verbleiben.

Aus diesem Grund wird in Prospector eine nicht-lineare Funktion zur Veränderung von Gewichtungen verwendet. Die Funktion ist in Formel 5.4 angeführt und zeigt einen Verlauf wie in Abbildung 5.17. Wie gewünscht ist diese im Bereich um 0,5 steiler und sorgt so für eine schnellere Veränderung der Gewichtungen. Zu den Rändern bei 0 und 1 hin wird sie flacher, was für eine Stabilisierung der Gewichtung sorgt.

$$f(x) = \frac{\cos((1-x)\cdot\pi}{2} + 0{,}5 \tag{5.4}$$

Die konkrete Änderung von Modellen nach einer Bewertung erfolgt nach folgendem Schema:

1. *Gewichtung bestimmen*: die Gewichtung des Themas, in welches das Ergebnis klassifiziert ist, wird aus dem Benutzermodell gelesen. Sollte der entsprechende Pfad nicht vorhanden sein, so wird er erzeugt, und die Gewichtungen werden mit den Formeln des vorangegangenen Unterabschnitts abgeleitet.

2. *Gewichtung rückführen*: mittels der Umkehrfunktion der in Formel 5.4 auf der vorherigen Seite genannten Funktion wird das Gewicht auf einen Wert auf der x-Achse rückgeführt.

3. *Bewertung umsetzen*: bei einer positiven Bewertung wird zu diesem Wert 0,1 addiert, bei einem negativen Wert 0,1 subtrahiert. Werden durch diese Operation 0 oder 1 erreicht, so werden diese Werte nicht überschritten.

4. *Neue Gewichtung berechnen*: mit der Funktion in Abbildung 5.17 auf der vorherigen Seite wird aus dem x-Wert die neue Gewichtung berechnet.

5. *Änderung nach oben propagieren*: auch die dem aktuellen Knoten $t$ übergeordneten werden durch die Änderung der Gewichtung $\Delta G(b,t)$ für das Modell des Benutzers $b$ beeinflusst. Gemäß Formel 5.5 ergibt sich die Änderung der Gewichtung $\Delta G(b,t')$ eines übergeordneten Knotens $t'$ durch eine Skalierung mit dessen Tiefe.

$$\Delta G(b,t') = \Delta G(b,t) \cdot \frac{Tiefe(t')}{Tiefe(t)} \tag{5.5}$$

6. *Änderung in Gruppenmodellen vornehmen*: auch in den Gruppenmodellen, zu denen der Benutzer eine positive Affinität hat, wird eine Änderung vorgenommen. Der Ablauf erfolgt gemäß den zuvor genannten Punkten eins bis fünf. Der jeweilige Wert, um den sich ein Knoten ändert, wird aber zusätzlich noch durch einen Faktor $Einfluss(b,g)$, der den Einfluss eines Benutzers $b$ auf eine Gruppe $g$ angibt, skaliert. Wie in Formel 5.6 zu sehen ist, wird dieser Wert aus dem Anteil der Affinität des Benutzers an der Summe aller Affinitäten von Benutzern in Bezug auf diese Gruppe berechnet.

$$Einfluss(b,g) = \frac{Affinität(b,g)}{\sum_{i=1}^{N} Affinität(b_i,g)} \tag{5.6}$$

Der in Schritt 3 erwähnte Wert 0,1 bedarf noch einer Erklärung: Er bestimmt, wie schnell sich die Gewichtung bei einer Bewertung in Richtung der beiden Extreme 0 und 1 bewegt. In der konkreten Ausprägung mit 0,1 lägen zwischen diesen Randwerten zehn positive bzw. negative Bewertungen. Sollte das System langsamer bzw. schneller reagieren, so kann der Wert auch verkleinert bzw. vergrößert werden. Aufgrund von Beobachtungen des Systemverhaltens hat sich der Wert 0,1 als brauchbar erwiesen.

### 5.5.3 Relevanzberechnung für Suchergebnisse

Jedes Ergebnis, das die darunterliegende Suchmaschine an Prospector liefert, wird gemäß der verwendeten Ontologie (in dieser Implementierung das ODP-Verzeichnis) klassifiziert. Zur Berechnung der Relevanzen, die den Wahrscheinlichkeiten des Interesses des Benutzers entsprechen, wird bei den Ergebnissen anschließend folgende Unterscheidung getroffen:

- *Unklassifiziert*: konnte ein Ergebnis nicht klassifiziert werden, so wird die Relevanz mit 0,5 gewählt. Das Ergebnis wird somit weder bevorzugt noch benachteiligt gereiht.
- *Verzeichnet*: wenn das Thema, in welches das Ergebnis klassifiziert wurde, bereits im Benutzermodell vorhanden ist, so wird dessen Gewichtung als Relevanz genommen.
- *Abgeleitet*: andernfalls wird versucht, eine Relevanz aus den Gewichtungen der Gruppenmodelle und übergeordneter Themen im Benutzermodell abzuleiten. Dazu werden die selben Berechnungen wie jene in „Ableitung von Gewichtungen“ auf Seite 71 verwendet.

Die berechneten Relevanzen dienen beim anschließenden Umreihen der Ergebnisse als Sortierkriterium. Ergebnisse mit gleichen Relevanzen werden in der originalen Reihenfolge gereiht.

## 5.6 Zusammenfassung

Zum Abschluss dieses Kapitels seien noch einmal stichwortartig die wichtigsten Eigenschaften von Prospector zusammengefasst:

- *Meta-Suchmaschine*: Prospector verwendet die Suchergebnisse anderer Suchmaschinen, um sie für seine Benutzer zu personalisieren.
- *Klassifizierung*: Suchergebnisse werden in der Ontologie des Open Directory Project klassifiziert und so in Themengebiete eingeordnet. Dadurch werden sie mit semantischen Metadaten angereichert.
- *Benutzermodellierung*: mit positiven und negativen Bewertungen können Benutzer ihr (Des-)Interesse an Suchergebnissen angeben. Anhand der Klassifizierung wird ein entsprechendes Benutzermodell aufgebaut.
- *Umreihung*: Prospector berechnet anhand des Benutzermodells und der Gruppenmodelle die Relevanz von klassifizierten Ergebnissen und reiht sie entsprechend.
- *Gruppenmodelle*: über eine festgelegte Zahl thematischer Gruppenmodelle können Interessenslagen zwischen mehreren Benutzern geteilt und neuen Benutzern schnell personalisierte Ergebnisse geliefert werden.
- *Affinität zu Gruppen*: Benutzer können mit der Affinität die Stärke ihres Interesses an einer bestimmten Gruppe festlegen. Dies bestimmt, wie stark das Gruppenmodell seine Reihung beeinflusst, und wie stark seine Bewertungen das Gruppenmodell verändern.
- *Umreihung nach Gruppen*: Suchergebnisse können auch speziell nach einem Gruppenmodell umgereiht und so auf dessen thematisches Gebiet ausgerichtet werden. Diese Funktion steht auch anonymen, nicht angemeldeten Benutzern zur Verfügung.
- *Zugriff auf das Benutzermodell*: Benutzer können ihr persönliches Modell einsehen und bearbeiten. Sie können detaillierte Änderungen bei den Gewichtungen einzelner Interessensgebiete vornehmen.

# Kapitel 6

# Entwicklung und Implementierung

In diesem Kapitel wird die Weiterentwicklung von Prospector und die Implementierung neuer Funktionalitäten behandelt. Zu Beginn wird das Carrot$^2$-Framework vorgestellt, da Prospector in dieses integriert werden sollte und es die Basis für das Verständnis der weiteren Erklärungen bildet. Anschließend wird die Software-Umgebung erläutert, in der entwickelt und getestet wurde. Es folgt der Teil zur Integration von Prospector in Carrot$^2$, gefolgt von der Beschreibung der Erweiterungen des Systems. Neu in Prospector ist auch der große Teil der Systemüberwachung, dem ein eigener Abschnitt gewidmet ist. Abschließend werden die Maßnahmen zum Testen und Verbessern der Leistung des Systems beschrieben.

## 6.1 Carrot$^2$-Framework

Eine der Hauptaufgaben bei der Weiterentwicklung von Prospector war, einen *soliden architektonischen Unterbau* zu schaffen und gleichzeitig die Gebrauchstauglichkeit der Bedienoberfläche zu verbessern. Die Wahl zur Grundlage für die Umsetzung dieser Anforderungen fiel auf das Carrot$^2$-Framework. Dieses stellt eine grundlegende Architektur für den Aufbau einer Meta-Suchmaschine zur Verfügung, liefert bereits fertige Komponenten zum Abfragen diverser bestehender Suchmaschinen und ist nach dem Open Source-Prinzip entwickelt und frei erweiterbar. Im Folgenden wird das Projekt näher beschrieben.

### 6.1.1 Idee und Zielsetzung

In Abschnitt 1.1 wurden einige Unzulänglichkeiten aktueller Suchmaschinen in Bezug auf die Informationsmenge im Internet genannt. Hauptproblem war hierbei die Frage der für den Benutzer sinnvollen Reihung einer großen Menge von Suchergebnissen. Doch auch wenn es darum geht, einen Überblick zu einem Thema oder eine tiefergehende Analyse eines Gebiets zu liefern, können gängige Suchmaschinen noch keine passende Information liefern. Sie sollten für eine Anfrage nicht nur die relevantesten Dokumente anzeigen sondern auch einen umfassenden *Überblick der Themen*, die damit abgedeckt sind, bieten [SW03].

Ein vielversprechender Ansatz ist hierbei, Suchergebnisse automatisch in thematische Kategorien, so genannte Cluster zu gruppieren. Durch eine aussagekräftige Benennung dieser

*Cluster* erhält der Benutzer einen Überblick zu den gefundenen Themenbereichen und kann Zeit sparen, anstatt mit dem Sichten irrelevanter Ergebnisse beschäftigt zu sein. Im Gegensatz zu einer (Offline-) Klassifizierung in vordefinierte Kategorien, wie dies beim Open Directory Project der Fall ist, wird dieses Clustering (online) bei jeder Suche dynamisch und nur mit den Informationen aus den Ergebnissen vorgenommen [SW03]. Mit dieser Technik des Web Mining könnten Ergebnisse der Anfrage „tiger“ beispielsweise in die Themen „Mac OS“, „Tiger Woods“, „Security Tool“ und „Tiger Attack Helicopter“ gruppiert werden [OW07].

Das Ziel des Carrot$^2$-Projekts, welches ursprünglich am Institute of Computing Science der Poznan University of Technology in Polen ins Leben gerufen wurde, ist die Bereitstellung eines offenen, flexiblen und freien *Frameworks zum Entwickeln von Clustering-Systemen* für Suchergebnisse. Es soll die Vergleichbarkeit von unterschiedlichen Algorithmen erleichtern und neue Ideen zur Visualisierung von Suchergebnissen hervorbringen [SW03]. Das Framework dient jedoch nicht nur zum Experimentieren sondern ist auch geeignet, voll funktionsfähige Anwendungen zu erstellen. Zu diesem Zweck enthält Carrot$^2$ bereits eine Vielzahl von Bausteinen zum: [Car08]

- *Abfragen von Dokumenten*: für Google, Yahoo!, MS Live Search, eTools Meta-Suche, Alexa, PubMed, OpenSearch, Lucene, Apache Solr. Carrot$^2$ durchsucht (wie für eine Meta-Suchmaschine üblich) nicht selbst den Dokumentenbestand, sondern greift auf bestehende Dienste zurück.
- *Verarbeiten der Ergebnisse*: sprachabhängige Tokenizer, Stemmer und Stoppwortlisten.
- *Clustern*: fünf funktionsfähige Clustering-Algorithmen (STC, HAOG-STC, FuzzyAnts, Rough k-means, Lingo, Lingo3G) sind inkludiert.
- *Ausgeben der Ergebnisse und Cluster*: Carrot$^2$ bietet eine Web-Anwendung, eine graphische Anwendung, in der man die Parameter der Clustering-Algorithmen feineinstellen kann und einen Document Clustering Server, der über HTTP/REST ansprechbar ist und die Ergebnisse im XML- oder JSON-Format liefert.

### 6.1.2 Lizenz

Die Entwickler von Carrot$^2$ erlauben in ihrer *BSD-ähnlichen Lizenz*[1] die Weiterverbreitung als Quellcode oder kompiliert, mit oder ohne Änderungen, sofern das Copyright klar erwähnt ist und die folgenden Bedingungen und Haftungsausschlüsse der Lizenz beiliegen. Der Name der Technischen Universität Poznan in Polen und die Namen der zum Projekt Beitragenden dürfen gemäß diesen Bedingungen ohne schriftliche Erlaubnis nicht zur Bewerbung von abgeleiteten Produkten verwendet werden. Außerdem soll der Dokumentation und/oder dem Produkt ein Hinweis beiliegen, dass dieses Software des Carrot$^2$-Projekts enthält. Schlussendlich dürfen keine Algorithmen oder technischen Lösungen des Projekts patentiert oder als proprietär erklärt werden.

[1] `http://project.carrot2.org/license.html` (7. Oktober 2008)

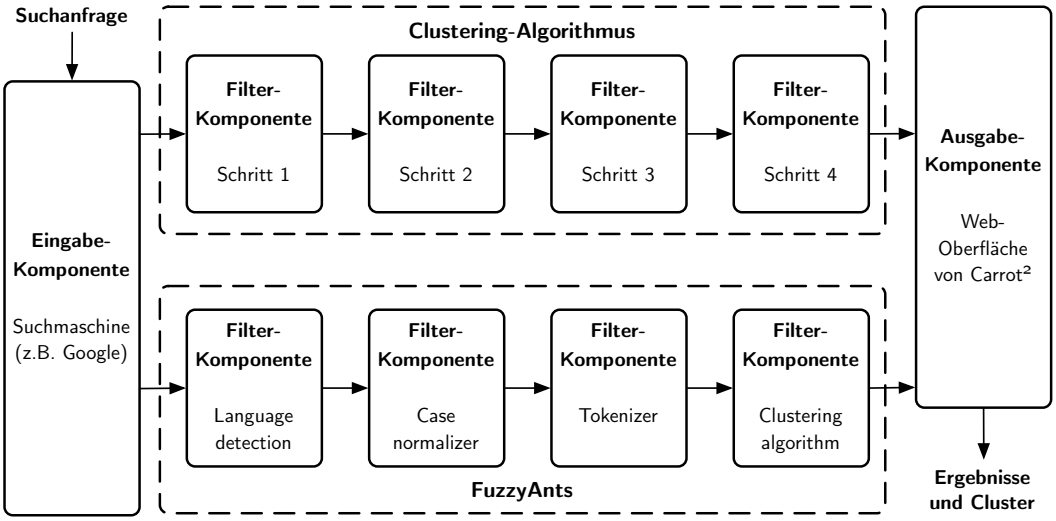


**Abbildung 6.1:** Pipe-and-Filter-Architektur von Carrot$^2$

Carrot$^2$ kann laut den Entwicklern in jeder Art von Projekt (Open Source, Forschung, kommerziell) verwendet werden, solange man (auf der Projekt-Webseite und in der Dokumentation) auf seine Verwendung entsprechend hinweist und die Entwickler davon in Kenntnis setzt [Car08]. Dank dieser *wenig restriktiven Lizenz* konnte es auch bereits erfolgreich in einigen kommerziellen und akademischen Anwendungen eingesetzt werden. Wie später noch näher erklärt wird, verfügt die neue Version von Prospector über einen immer sichtbaren Hinweis auf das Copyright und die Verwendung von Carrot$^2$.

### 6.1.3 Architektur und Datenfluss

Das Carrot$^2$-Framework ist in Java entwickelt und realisiert eine Pipe-and-Filter-Architektur (siehe Abbildung 6.1). In dieser werden unterschiedliche Komponenten zu einer Verarbeitungskette verbunden. Diese Kette, genannt Pipeline, wird an ihrem Anfang mit Daten versorgt, welche durch die Komponenten geschleust, von ihnen verarbeitet und am Ende wieder zurückgegeben werden. Die Beschreibung der Architektur erfolgt zunächst anhand der vorhandenen Komponenten, erklärt dann die Daten und deren Verarbeitung und geht zum Schluss auf die Steuerung dieses Prozesses ein.

#### Komponenten

In Carrot$^2$ gibt es drei grundlegende *Typen* von Komponenten [Car08]:

1. *Eingabe-Komponenten*: erhalten die Suchanfrage des Benutzers und erzeugen entsprechende Ergebnisse bzw. erhalten diese von externen Quellen (z. B. Suchmaschinen).

2. *Filter-Komponenten*: verändern, transformieren oder reichern die Daten an, beispielsweise durch Vereinheitlichung der Groß- und Kleinschreibung, Entfernung von Stoppwörtern oder das eigentliche Erzeugen von Clustern aus der Menge der Ergebnisse.

3. *Ausgabe-Komponenten*: sammeln die Ergebnisse und die erzeugten Cluster und ermöglichen die Darstellung für den anfragenden Benutzer.

Alle Komponenten sind vom Typ `LocalComponent`. Dieses Präfix „Local“ kommt übrigens in vielen Bereichen von Carrot$^2$ vor und stammt aus einer Zeit, als noch ein paralleler Ansatz mit Remote-Kommunikation zwischen verteilten Komponenten verfolgt wurde. Das grundlegende Interface `LocalComponent` definiert unter anderem Methoden zum Steuern der Verarbeitung in den Komponenten: `startProcessing(RequestContext requestContext)` startet die Verarbeitung, übergibt die Anfrageparameter und muss bei der jeweils nachfolgenden Komponente ebenfalls aufgerufen werden. `endProcessing()` signalisiert, dass die Vorgängerkomponente ihre Verarbeitung abgeschlossen hat und keine Daten mehr liefern wird. Die Komponente muss nun ihre Arbeit fertigstellen und die Methode ihrerseits bei der nachfolgenden aufrufen.

Die oben genannten drei Typen von Komponenten leiten sich als drei Interfaces von `LocalComponent` ab. `LocalInputComponent` verfügt über eine Methode `setQuery(String query)`, um der Eingabekomponente die Suchanfrage zu übergeben. In einer `LocalOutputComponent` kann man mit der Methode `getResult()` das Ergebnis der Pipeline abfragen. `LocalFilterComponent` und `LocalInputComponent` haben zudem noch die Methode `setNext(LocalComponent next)` zum eigentlichen Zusammenhängen der Komponenten zu einer Verarbeitungskette [Car08].

### Daten

Die bisher beschriebenen Teile ermöglichen das Erstellen einer Pipeline, die Prüfung der Kompatibilität zwischen aufeinanderfolgenden Komponenten und das Steuern der Verarbeitung. Das eigentliche Weiterreichen der Daten und deren Format ist davon getrennt modelliert, um den Ansatz flexibel zu halten und nicht einzuschränken. In Carrot$^2$ sind zwei *Typen* von Daten vordefiniert:

- *Dokument*: gemäß den Methoden im Interface `RawDocument` verfügt ein Dokument über ID, URL, Relevanz (zwischen 0 und 1), Titel und die, Snippet genannte, Zusammenfassung des Dokuments bzw. jener Textpassage, die auf die Suchanfrage passt. Darüber hinaus ist mit `getProperty(String name)` und `setProperty(String propertyName, Object value)` ein generischer Mechanismus vorgesehen, um weitere Attribute zu speichern. Das Interface ist in der Klasse `RawDocumentBase` implementiert.

- *Cluster*: laut dem Interface `RawCluster` hat ein Cluster eine der Wichtigkeit nach absteigend sortierte Liste von ihn beschreibenden Begriffen, eine Menge von Dokumenten, eine Menge von untergeordneten Clustern und eine numerische Relevanz. Die Implementierung in der Klasse `RawClusterBase` bietet auch Methoden zum bequemen Verwalten eines Clusters.

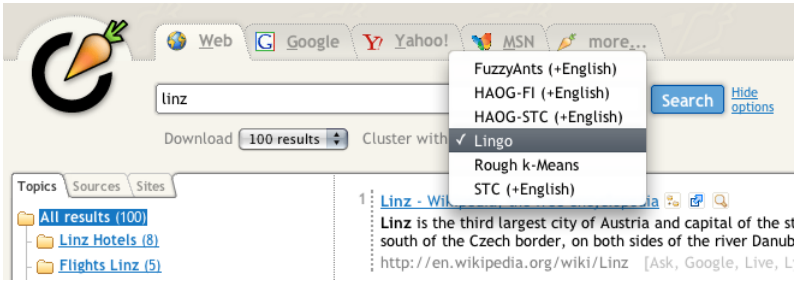


**Abbildung 6.2:** Carrot$^2$ – Eingabefeld für Suchanfrage mit Wahl von der zu verwendenden Suchmaschine und des Clustering-Algorithmus

Die *Datenübertragung* zwischen den Komponenten ist über Kontrakte in Interfaces geregelt. Implementiert eine Komponente die Interfaces `RawDocumentsProducer` oder `RawClustersProducer`, so zeigt sie an, dass sie Dokumente bzw. Cluster als Ausgabe hat. Diese Interfaces enthalten keine Methoden, sind also nur so genannte „marker interfaces“. Als Gegenstück gibt es die Interfaces `RawDocumentsConsumer` und `RawClustersConsumer`, die die Methoden `addDocument(RawDocument doc)` bzw. `addCluster(RawCluster cluster)` zur Annahme von Daten definieren.

#### Steuerung

Die Logik zum Zusammenfügen von Komponenten in einer Verarbeitungskette, Überprüfen der Kompatibilitäten und Steuern der Verarbeitung (Starten, Stoppen) liegt in einem *Prozess*, verkörpert durch die Schnittstelle `LocalProcess` bzw. die mitgelieferte Implementierung `LocalProcessBase`. Der Prozess enthält selbst keine Komponenten, sondern kann diese nur steuern. Er wird gemeinsam mit den Komponenten in einem `LocalController` (bzw. `LocalControllerBase`) zusammengefasst [Car08].

Dieser Controller kann eine Menge von Komponenten (und ihre Factories) verwalten, vorzugsweise in einem Pool, in dem diese nach ihrer Verwendung für spätere Verarbeitungen gespeichert und dann wieder verwendet werden können. Der Controller kann auch mehrere Prozesse verwalten. Diese stehen für unterschiedliche Verarbeitungsketten, also für die genauen Abfolgen von Filter, die für die jeweiligen Clustering-Verfahren benötigt werden [Car08].

### 6.1.4 Web-Schnittstelle

Carrot$^2$ verfügt über eine voll funktionsfähige Web-Schnittstelle zum *Suchen und Anzeigen* von Dokumenten und der Cluster, in die sie gruppiert wurden. Unterschiedliche Suchmaschinen als Quelle und der zu verwendende Clustering-Algorithmus können vom Benutzer gewählt werden. Dargestellt ist dies in Abbildung 6.2. Die einzelnen Suchmaschinen sind über Register (tabs) wählbar, der zu verwendenden Clustering-Algorithmus mittels einer als Combobox dargestellten, standardkonformen HTML-Auswahl (select).

Die *Anzeige der Ergebnisse* erfolgt in zwei IFRAME-Bereichen unterhalb des Suchformulars (siehe Abbildung 6.3). Links werden die berechneten Cluster und die Zahl der jeweils pro Cluster enthaltenen Dokumente dargestellt. Es hängt vom Algorithmus ab, ob die Cluster als einfache Liste oder in einer Hierarchie organisiert werden. Durch Anwählen eines Clusters werden die entsprechenden Ergebnisse im rechten IFRAME angezeigt.

### Implementierung

Die Web-Schnittstelle von Carrot$^2$ ist als *Java Web-Anwendung* implementiert. Sie baut auf zwei gängige Basistechnologien auf:

- *Servlets*: die gesamte Kommunikation zwischen Browser und der Web-Anwendung geht über das `QueryProcessorServlet`. Bei dessen Initialisierung werden alle unterstützten Clustering-Algorithmen, ihre Verarbeitungsketten mit den Komponenten, die entsprechenden Prozesse und der Controller geladen. Das `QueryProcessorServlet` kann sowohl Anfragen zur Darstellung der gesamten Seite als auch der einzelnen Teile (Cluster- bzw. Dokumente-IFRAME) verarbeiten.

  Gemäß den Spezfikationen in den Such-Parametern verwendet es die entsprechende Quell-Suchmaschine und Verarbeitungskette und lässt deren Ergebnisse von Serialisierungskomponenten (`FancyDocumentSerializer`, `XMLClustersSerializer`) in HTML- oder XML-Code umwandeln. Wirklich durch die Verarbeitungskette laufen die Dokumente aber nur bei einer Anfrage für das Cluster-IFRAME. Die Dokumente werden direkt von der Quell-Suchmaschine an die Serialisierungskomponente übergeben. Zwischen diesen zwei HTTP-Anfragen werden die nur einmal abgefragten Ergebnisse in einem Cache gespeichert.

- *XSL Transformations (XSLT)*: der Datenfluss vom Servlet zum Browser wird durch Filter geleitet. Liefert die Serialisierungskomponente XML-Code, wie dies bei allen Antworten außer der Ergebnisliste der Fall ist, so wird dieser mittels XSLT in HTML-Code

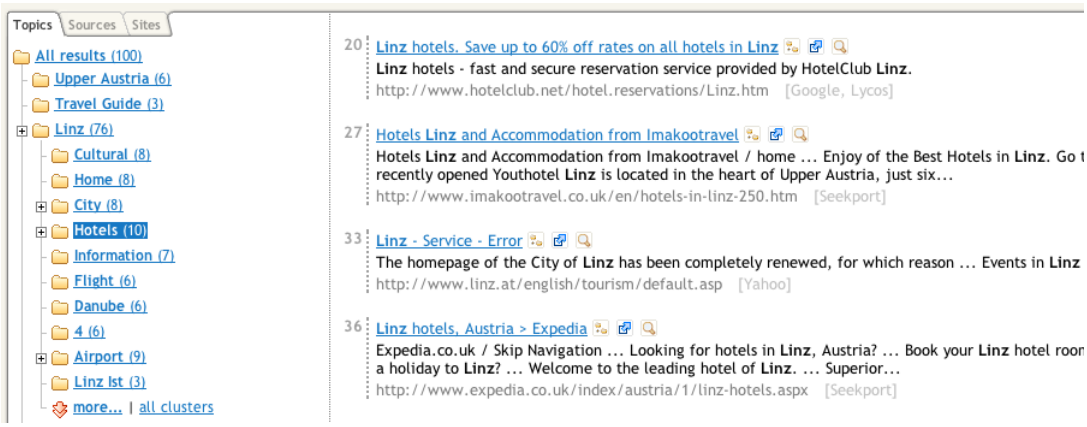


**Abbildung 6.3:** Carrot$^2$ – Cluster und Dokumente

transformiert. Die Definition der Transformation in einem XSLT-Dokument macht es einfach, ohne Eingriff in den Code Änderungen an der Ausgabe vorzunehmen.

Wie bereits erwähnt, werden sowohl Eingabe- und Filter-Komponenten als auch ganze Verarbeitungsketten und Prozesse vom `QueryProcessorServlet` bei dessen Initialisierung vorbereitet. Die Informationen, wie diese einzelnen Teile konfiguriert und verbunden sind, finden sich in Konfigurationsdateien, die außerhalb des Codes der Web-Anwendung liegen. Alle Eingabe-Komponenten (z. B. für Google, Yahoo! oder MS Live Search) sind im Verzeichnis `web/inputs` als kurze BeanShell-Skripte definiert. Die Verwendung von BeanShell[2] als eingebetteten Skript-Interpreter mit Java-Syntax und der Möglichkeit, Java-Objekte zu verarbeiten, lässt eine einfache Konfiguration der Komponenten zu, ohne Code neu kompilieren zu müssen.

Auch Filter-Komponenten werden mit BeanShell-Skripten konfiguriert. Alle im Verzeichnis `web/algorithms` mit „`filter-`“ beginnenden Dateien werden dazu berücksichtigt. Bereits vordefiniert sind die Eingabe-Komponente „`input-demo-webapp`“ und die Ausgabe-Komponente „`output-demo-webapp`“. Erstere liefert die Ergebnisse von der in der Web-Oberfläche ausgewählten Quell-Suchmaschine. Letztere gibt die Ergebnisse der Verarbeitungskette an die Web-Anwendung zurück, wo sie angezeigt werden. Ein Beispiel für eine derartige Filter-Definition findet sich in Code 6.3 auf Seite 89.

Ebenfalls in diesem Verzeichnis können durch XML-Dateien mit dem Präfix „`alg-`“ Prozesse und ihre Verarbeitungsketten definiert werden. Für den Prozess wird ein Namen, eine Beschreibung und die genaue Abfolge der zuvor definierten Komponenten definiert. Ein derart konfigurierter Prozess ist in der Web-Oberfläche bei der Auswahl des Algorithmus sichtbar. In Code 6.4 auf Seite 90 ist ein Beispiel für die Definition eines Prozesses zu sehen.

## 6.2 Entwicklungsumgebung

Bevor auf die eigentliche Entwicklungstätigkeit eingegangen wird, sei an dieser Stelle noch auf die Umgebung, in der diese stattfand, verwiesen. Diese setzte sich aus folgenden relevanten Komponenten zusammen:

- *Betriebssystem*: Die Entwicklung erfolgte hauptsächlich unter *Mac OS X* 10.4 und 10.5. Getestet und evaluiert wurde Prospector unter *Windows XP* mit Service Pack 2.
- *Programmiersprachen/-umgebungen*: Als primäre Sprache kam *Java* mit dem Java Development Kit (JDK) 1.5 zum Einsatz. Im Bereich der Web-Oberfläche wurde *JavaScript* mit der Bibliothek *jQuery 1.2.6* (`http://www.jquery.com`) verwendet.
- *Persistenz und Datenbanken*: Die Persistierung der Daten erfolgte über den objekt-relationale Persistenz- und Abfrage-Dienst *Hibernate* (`http://www.hibernate.org/`) in der Version 3. Die eigentliche Speicherung übernahmen die SQL-Datenbanken *Hyperthreaded Structured Query Language Database (HSQLDB)* (`http://www.hsqldb.org/`) und *MySQL 5* (`http://dev.mysql.com/`).

[2] `http://www.beanshell.org/`

- *Servlet- und JSP-Container*: Die Web-Oberfläche mit ihren Servlets, JSP-Seiten und Filtern lief unter *Tomcat 6* (`http://tomcat.apache.org/`).
- *Entwicklungsumgebungen*: Der Großteil der Entwicklung an Prospector erfolgte in *Eclipse 3* (`http://www.eclipse.org/`). Aufgrund der guten Unterstützung für Performance-Messungen und der einfachen Möglichkeiten zum Verpacken von Web-Anwendungen in Web Application Archive (WAR)-Dateien wurde auch auf *Netbeans 6.1* (`http://www.netbeans.org/`) zurückgegriffen.
- *Browser und andere Werkzeuge*: Entwickelt wurde an der Web-Oberfläche in *Firefox 2* und *3* mit der *Web Developer Extension* (`http://chrispederick.com/work/web-developer/`) und *Firebug* (`http://www.getfirebug.com/`). Tests und Anpassungen wurden auch in *Safari 3*, *Opera 9* und *Internet Explorer 6* und *7* vorgenommen. Für das Testen von einzelnen Komponenten und der gesamten Web-Anwendung wurde *Apache jMeter 2.3.2* (`http://jakarta.apache.org/jmeter/`) verwendet.

## 6.3 Integration von Prospector in Carrot$^2$

Nach dieser einleitenden Beschreibung von Architektur und Oberfläche des Carrot$^2$ Clustering Frameworks wird nun die Integration von Prospector in dieses System erläutert. Die Integration erfolgte in zwei Schritten: zuerst wurde der Algorithmus als ein Clustering-Algorithmus definiert, der Ergebnisse umreiht und deren unterschiedliche Klassifizierungen gemäß der verwendeten Ontologie als hierarchische Menge von Clustern darstellt. Doch Prospector liefert noch zusätzliche Ausgaben (z. B. Relevanz eines Ergebnisses) und benötigt auch die Rückmeldungen der Benutzer. Unter anderem aus diesen Gründen musste in einem zweiten Schritt auch eine Anpassung der Web-Oberfläche vorgenommen werden.

### 6.3.1 Prospector-Algorithmus

Im ersten Schritt sollte der Prospector-Algorithmus als ein Clustering-Algorithmus für Carrot$^2$ adaptiert werden. Da dessen Web-Oberfläche nicht zwischen den einzelnen Benutzern unterscheiden kann, wurden für die Entwicklung mit dem ursprünglichen Prospector-System ein Benutzer und entsprechende Modelle erzeugt und diese fix im Algorithmus definiert. Dies hatte auch den Vorteil, dass sich die Parameter des Algorithmus nicht änderten, und so beim Testen korrekte Ergebnisse ohne viel Mühe erkannt werden konnten.

Das Carrot$^2$ Framework basiert auf der Architektur des *schrittweisen Verarbeitens von Daten* und unterstützt ein Austauschen der einzelnen Verarbeitungskomponenten. Da Prospector flexibel mit unterschiedlichen Datenquellen arbeiten können soll, wurde der Algorithmus in drei unabhängige Schritte zerlegt:

1. *Ergebnisse klassifizieren*: entsprechend der Ontologie wird jedes Ergebnis klassifiziert und so in die Struktur, die den Overlay-Modellen zugrunde liegt, eingeordnet.

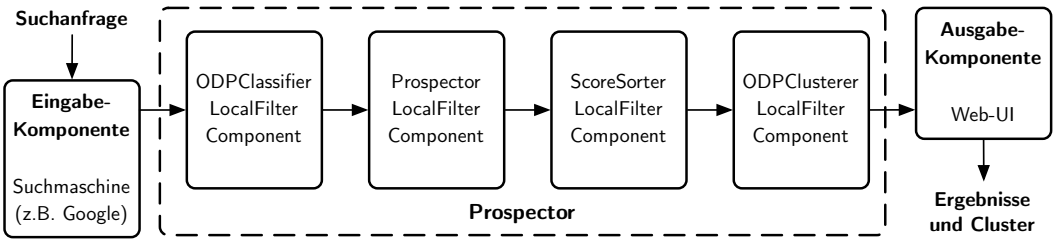


**Abbildung 6.4:** Carrot$^2$-Verarbeitungskette mit den Komponenten von Prospector

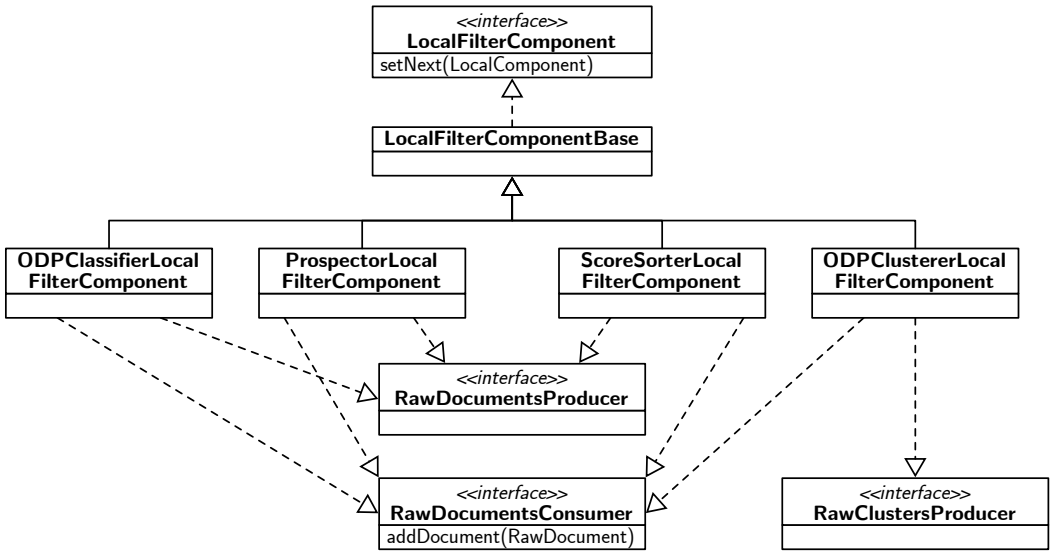


**Abbildung 6.5:** Carrot$^2$ Filter-Komponenten der Verarbeitungskette für Prospector

2. *Relevanzen berechnen*: mit den Informationen aus den Modellen wird für das klassifizierte Ergebnis die vorhergesagte Relevanz, dass dieses den Benutzer interessiert, berechnet.

3. *Umreihen*: die Ergebnisse werden nach absteigender Relevanz umgereiht.

Jeder dieser Schritte lässt sich in eine Komponente fassen und wird somit austauschbar. Die verwendete Ontologie, die Art wie die Relevanzen berechnet werden, und deren Einfluss auf die Reihung lassen sich damit auswechseln. Konkret implementiert wurden die Schritte in den Klassen `ODPClassifierLocalFilterComponent`, `ProspectorLocalFilterComponent` und `ScoreSorterLocalFilterComponent`, die in Abbildung 6.4 in der Carrot$^2$-Architektur gezeigt und Abbildung 6.5 in einem UML-Klassendiagramm zu Teilen des Frameworks in Beziehung gesetzt werden. Sie alle leiten sich von der Basisimplementierung einer Filterkomponente ab. Jene Komponenten, die Dokumente als Eingabe nehmen, implementieren `RawDocumentsConsumer`. Liefern sie Dokumente als Ausgabe, so implementieren sie `RawDocumentsConsumer`. Alle drei Komponenten, sowie auch die bisher nicht genannte `ODPClustererLocalFilterComponent`, werden im Folgenden kurz beschrieben.

#### ODPClassifierLocalFilterComponent

Als Eingabe erhält diese Komponente über die Methode `addDocument(RawDocument doc)` Daten vom Typ `RawDocument`. Anhand der URL des jeweiligen Dokuments wird dieses in der Ontologie des ODP klassifiziert. Ergebnis davon ist eine Liste mit den Namen jener Themengebieten, die den Pfad von der Wurzel bis zu jenem Thema, in dem das Dokument schlussendlich klassifiziert ist, bilden.

Wenn die Klassifizierung erfolgreich war, so muss diese Information zu dem Dokument-Objekt hinzugefügt werden. Das Interface `RawDocument` sieht dazu die Methode `setProperty(String propertyName, Object value)` vor, mit der Name-Wert-Paare gespeichert werden können. Damit auch von anderen Komponenten aus sicher auf die Liste der Themengebiete zugegriffen werden kann, wird der Wert der in der Hilfsklasse `ProspectorHelper` definierten Konstante `DOCUMENT_ODP_CLASSIFICATIONS` als Name verwendet. Anschließend wird das Dokument über die `addDocument()`-Methode an die nachfolgende Komponente weitergegeben.

#### ProspectorLocalFilterComponent

Diese Komponente übernimmt die klassifizierten Dokumente und berechnet für jedes die Relevanz im Bezug auf das Benutzermodell und die Gruppenmodelle. Zu diesem Zweck greift es auf die schon bestehende Implementierung des Prospector-Algorithmus zurück. Zuvor muss jedoch noch das `RawDocument`-Objekt in das intern in Prospector verwendete Ergebnisobjekt `SearchResultItem` umgewandelt werden. Dieses stammt noch aus den vorangeganenen Versionen, doch seine Verwendung sichert auch die Unabhängigkeit des eigentlichen Algorithmus von Carrot[2].

Zum Umwandeln wurde die Methode `getSearchResultItem(RawDocument doc, int index)` in `ProspectorHelper` geschaffen. Sie gibt ein entsprechendes `SearchResultItem`-Objekt zurück. Intern wird dazu eine neu erstellte Unterklasse `GenericSearchResultItem` verwendet. Diese verfügt über die Möglichkeit, beliebige Name-Wert-Paare zu speichern und kann so auch Daten im `RawDocument`-Objekt, die noch nicht in `SearchResultItem` vorgesehen waren, übernehmen. Nach dem Berechnen der Relevanz und dem Abspeichern im Ergebnisobjekt wird dieses durch `getRawDocument(SearchResultItem item)` wieder in ein `RawDocument` umgewandelt, welches anschließend an die nächste Komponente weitergegeben wird.

#### ScoreSorterLocalFilterComponent

Ursprünglich wurde das Umreihen der Ergebnisse ebenfalls durch den Prospector-Algorithmus erledigt. Es erschien aber als einheitlicher, das Konzept der klaren Aufteilung der einzelnen Schritte in eigenständige Komponenten auch hier anzuwenden. Somit ist es möglich, alternative Algorithmen zum Umreihen auf Basis der Relevanzen zu finden. Wie weiter unten noch beschrieben wird, kam es im Lauf der Entwicklung tatsächlich zu einer derartigen Änderung.

Die Implementierung dieser Komponente unterscheidet sich von den bisher beschriebenen, da nicht wie beim Klassifizieren oder Berechnen der Relevanz jedes Dokument einzeln bearbeitet werden kann. Zum Umreihen wird die gesamte Liste von Ergebnissen benötigt. Aus diesem Grund wird in `startProcessing(RequestContext requestContext)` eine leere Liste angelegt, zu der bei jedem Aufruf von `addDocument(RawDocument doc)` das entsprechende Dokument hinzugefügt wird. Wenn von der vorangegangenen Komponente `endProcessing()` aufgerufen wird, so signalisiert sie damit, dass keine weiteren Dokumente mehr kommen. In dieser Methode findet dann das eigentliche Umreihen statt.

Zu diesem Zweck wird, wie in Code 6.1 gezeigt, die in der Java-Bibliothek standardmäßig vorhandene, statische Methode `Collections.sort(List l, Comparator c)` verwendet. Diese sortiert die übergebene Liste anhand der Ordnungsrelation, die durch den `Comparator` definiert wird. In diesem Fall wird ein von `Comparator` abgeleitete anonyme Klasse erzeugt und die Methode `compare(Object o1, Object o2)` entsprechend implementiert. Die eigentliche Berechnung der Ordnungsbeziehung zwischen den zwei Objekten (0 wenn beide gleich sind, negativer Integer-Wert wenn `o1` kleiner und positiver wenn `o1` größer als `o2` ist) wird mit den Relevanzen (im Falle von Carrot[2] „score“ genannt) an die entsprechende Vergleichsfunktion der Klasse `Float` delegiert.

```
public void endProcessing() throws ProcessingException {
   Collections.sort(documents, new Comparator() {
      public int compare(Object obj1, Object obj2) {
         if (obj1 != null && obj2 != null &&
               obj1 instanceof RawDocument &&
               obj2 instanceof RawDocument) {

            RawDocument d1 = (RawDocument) obj1;
            RawDocument d2 = (RawDocument) obj2;
            return Float.compare(d2.getScore(), d1.getScore());

         } else return 0;
      }
   });
   Iterator it = documents.iterator();
   while (it.hasNext()) {
      RawDocument doc = (RawDocument) it.next();
      rawDocumentsConsumer.addDocument(doc);
   }
   super.endProcessing();
}
```

**Code 6.1:** Umreihen in `ScoreSorterLocalFilterComponent`

`Collections.sort(List l, Comparator c)` führt eine *stabile Sortierung* durch. Das heißt, dass Listenelemente mit gleicher Wertigkeit (d. h. gleicher Relevanz) ihre ursprüngliche Reihung zueinander nicht verlieren. Da die Dokument so die Verarbeitungskette durchlaufen, wie sie von der Quell-Suchmaschine zurückgegeben wurden, sorgt dies bei gleichen Relevanzen au-

tomatisch dafür, dass diese originale Reihung berücksichtigt wird. Dies ist insofern relevant, wenn Ergebnisse nicht klassifiziert werden konnten. In diesem Fall erhielten sie die neutrale Relevanz 0,5. All derartigen Ergebnisse bleiben in ihrer ursprünglichen Reihung. Nachdem die Liste sortiert wurde, muss in einer abschließenden Schleife noch jedes Dokument einzeln an die nachfolgende Komponente weitergegeben werden.

### ODPClustererLocalFilterComponent

Diese Komponente erzeugt jene Struktur, die in der Web-Oberfläche als *Cluster-Hierarchie* dargestellt wird. Diese enthält die einzelnen Cluster und Untercluster sowie die Verweise auf die darin enthaltenen Dokumente. Verwendet werden dazu `RawClusterBase`-Objekte, die das bereits besprochene `RawCluster`-Interface implementieren. Beim Start der Verarbeitung in `startProcessing(RequestContext requestContext)` werden ein Wurzel-Cluster und ein Rest-Cluster erzeugt. Ersterer sammelt zur leichteren Handhabung alle Cluster oberster Ebene (entspricht ODP-Themengebieten oberster Ebene) als Untercluster.

```
...
List classificationPath =  (List) doc.getProperty(
                        ProspectorHelper.DOCUMENT_ODP_CLASSIFICATIONS);

if (classificationPath != null && !classificationPath.isEmpty()) {
   String topicPath = "";
   RawClusterBase parentCluster = rootCluster;

   for (int i = 0; i < classificationPath.size(); i++) {
      String topic = (String) classificationPath.get(i);

      // Don't insert empty ancestors (if doc is in top-level topic)
      if (topic.length() > 0) {
         topicPath = topicPath + "/" + topic;
         RawClusterBase cluster = (RawClusterBase) map.get(topicPath);

         // If not in map we have to create a new cluster
         if (cluster == null) {
            cluster = new RawClusterBase();
            cluster.addLabel(topic);
            parentCluster.addSubcluster(cluster);
            map.put(topicPath, cluster);
         }
         parentCluster = cluster;
      }
   }
   parentCluster.addDocument(doc);
}
```

**Code 6.2:** Erstellen der Cluster-Hierarchie für die Klassifizierung eines Dokuments

Der Rest-Cluster enthält alle Dokumente, die nicht klassifiziert werden konnten. Er wird dem Wurzel-Cluster als Untercluster hinzugefügt. Ebenfalls angelegt wird eine Abbildung (Map) zwischen Themengebiet-Pfaden (z. B. „Computers/Programming/Languages/Java“) und dem entsprechenden Cluster am Ende dieses Pfades. Dies erspart das Durchlaufen des Baumes, um den richtigen Cluster für ein klassifiziertes Dokument zu finden.

In `addDocument(RawDocument doc)` wird zuerst das Dokument unverändert an die nachfolgende Komponente weitergegeben. Anschließend wird dessen gespeicherte Klassifizierung geladen (siehe Code 6.2 auf der vorherigen Seite). Es handelt sich um eine geordnete Liste von Themennamen, die im ODP-Datenbestand den Pfad zum eigentlichen Thema des Dokuments angeben. Dieser Pfad wird durchlaufen und für jede Ebene versucht, mit der Abildung das entsprechende Cluster-Objekt zu erhalten. Dieses wird immer als aktueller Eltern-Cluster in einer Variable gehalten. Sollte die Abbildung kein Cluster-Objekt finden, so wird ein neues mit dem Namen des aktuellen Themas erzeugt, als Untercluster des Eltern-Clusters gespeichert und als neuer Eltern-Cluster verwendet.

Nach dem vollständigen Durchlaufen des Klassifizierungspfades entspricht der Eltern-Cluster jenem Thema, in welches das Dokument im ODP klassifiziert wurde. Diesem Cluster wird es dann zugeordnet. Beim Beenden der Verarbeitung in `endProcessing()` werden schließlich alle Cluster oberster Ebene (also auch der Rest-Cluster) einzeln mit `addCluster(cluster)` an die nachfolgende Komponente weitergegeben.

### Einbindung der Komponenten

Um Prospector in Carrot$^2$ verwenden zu können, müssen die Komponenten konfiguriert und zu einem Algorithmus/Prozess verbunden werden. Im Verzeichnis `web/algorithms` der Web-Anwendung wurden daher entsprechende BeanShell-Skripte angelegt. Als Beispiel sei die Definition der Prospector-Filterkomponente in Code 6.3 gegeben.

```
import org.carrot2.filter.prospector.*;
import org.carrot2.core.*;
import org.carrot2.core.controller.*;

factory = new LocalComponentFactory() {
    public LocalComponent getInstance() {
      return new ProspectorLocalFilterComponent();
    }

    public String getName() { return "Prospector"; }
};

// Return the ID of the component and its factory.
return new LoadedComponentFactory("filter-prospector", factory);
```

**Code 6.3:** Definition der Prospector-Filterkomponente in `filter-prospector.bsh`

Die Verbindung der Eingabe-, Filter- und Ausgabekomponenten zur Verarbeitungskette des Prospector-Prozesses (in der Web-Oberfläche als Algorithmus bezeichnet) erfolgt in einer XML-Datei im selben Verzeichnis (siehe Code 6.4). Sofern alle benötigten Klassen im Java-Klassenpfad zu finden sind, lädt durch diese Definitionen die Carrot[2] Web-Anwendung beim Start des Servlet-Containers automatisch Prospector als Standard-Algorithmus.

```
<?xml version="1.0" encoding="UTF-8" ?>

<local-process id="Prospector">
   <name>Prospector</name>
   <description>Prospector Personalized Search Algorithm</description>

   <input component-key="input-demo-webapp" />
   <filter component-key="filter-odpcategorizer" />
   <filter component-key="filter-prospector" />
   <filter component-key="filter-scoresorter" />
   <filter component-key="filter-odpclusterer" />
   <output component-key="output-demo-webapp" />

   <attribute key="process.default" value="true" />
</local-process>
```

**Code 6.4:** Definition des Prospector-Prozesses in `alg-prospector.xml`

Wie bereits erwähnt, werden die Ergebnisse nur für die Anzeige der Cluster-Hierarchie durch diese geleitet, nicht jedoch für die Anzeige der Dokument-Liste. Da Prospector aber nicht wie die ursprünglichen Algorithmen nur Cluster erzeugt, sondern auch die Reihung der Ergebnisse verändert, musste im `QueryProcessorServlet` vor dem Serialisieren der Dokumente Code (vom Teil für Cluster-Anfragen) zum Benützen der Verarbeitungskette eingefügt werden. Wie der dadurch entstehende Overhead (Umreihen bei Cluster-Anfragen bzw. Clustern bei Dokument-Anfragen) gemindert werden konnte, wird in Unterabschnitt 6.6.1 beschrieben.

### 6.3.2 Anpassung der Web-Oberfläche

Bei der Integration des Algorithmus wurde die *Identität des Benutzers* zunächst noch nicht berücksichtigt, da sie für die grundlegende Implementierung keine Rolle spielt. Auf der Ebene der Web-Oberfläche kann diese jedoch festgestellt und zur Parametrisierung der Ergebnisverarbeitung herangezogen werden. Ebenfalls relevant ist hier die Anzeige der zusätzlichen Informationen (z. B. Relevanzen, Cluster), die Prospector für jedes Ergebnis liefert. Für die Modellierung der Benutzerinteressen musste in der Oberfläche ein Weg geschaffen werden, Rückmeldungen zu den Ergebnissen zu geben, die Affinitäten der Benutzer zu den Gruppenmodellen einzustellen und das Modell zu betrachten und zu verändern. Auch das Umreihen der Ergebnisse nach einem bestimmten Gruppenmodell wurde in der Web-Oberfläche implementiert.

Die Entwickler von Carrot[2] hatten bereits einige Elemente dynamischer Web-Programmierung integriert, beispielsweise für die Anzeige und Bedienung der Cluster-Hierarchie. Auch bei der Integration von Prospector sollte auf aktuelle Technologien gesetzt werden, um eine funktionell und optisch ansprechende Benutzeroberfläche zu schaffen. Zu diesem Zweck wurde insbesondere JavaScript verwendet, um im Browser Code auszuführen und nicht immer mit HTTP-Anfragen auf Server-Ressourcen zurückgreifen zu müssen.

### JavaScript-Bibliothek

Mit zwei Problemen haben JavaScript-Entwickler immer wieder zu kämpfen: Inkompatibilitäten zwischen Browsern und sehr allgemeine, auf niedriger Ebene angesiedelte Programmierschnittstellen. Das erste Problem lässt sich mit den Unterschieden bei der Unterstützung von Hyper-Text Markup Language (HTML) und Cascading Style Sheets (CSS) vergleichen. In beiden Fällen sind für die einzelnen Browser(versionen) und Betriebssysteme Fallunterscheidungen nötig, die den Code schlechter lesbar machen und die Wartbarkeit herabsetzen. Bei Funktionalitäten, die in allen Browsern funktionieren, handelt es sich oft um den kleinsten gemeinsamen Nenner, und es tritt Problem zwei auf: derartige Funktionen erledigen nur sehr grundlegende und feingranulare Aufgaben. Umfangreichere Operationen im Document Object Model (DOM) sind meist nur in bestimmten Browsern verfügbar oder müssen selbst programmiert werden.

Abhilfe können hier JavaScript-Bibliotheken bzw. Toolkits wie Yahoo! User Interface Library (YUI)[3], jQuery[4] und dojo[5] schaffen. Sie bieten eine große Zahl von häufig benötigten aber auch exotischen Funktionalitäten. Außerdem kapseln sie die Unterschiede zwischen den Browsern und bieten so eine einheitliche Schnittstelle. Eine weite Verbreitung stellt sicher, dass sie in den unterschiedlichsten Konfigurationen und Umgebungen getestet werden. Bei selbst entwickelten Bibliotheken könnte man hier nur schwer mithalten und hätte einen Qualitätsnachteil.

Für die Entwicklung an Prospector wurde die freie Bibliothek *jQuery* eingesetzt [jQu08]. Sie ist sehr schlank gehalten (nur ca. 15 KB groß), man kann aber neue Funktionalitäten durch Erweiterungen hinzufügen. Nichtsdestotrotz bietet auch schon die Basisversion umfangreiche Möglichkeiten in den Bereichen:

- *Selektoren*: Elemente der Seite können mit CSS3-Selektoren und grundlegenden XPath-Ausdrücken ausgewählt werden.
- *Attribute*: Zugriff auf und Veränderung von HTML-Attributen, CSS-Klassen, innerem HTML-Code, Text und Wert von Elementen.
- *DOM-Baum durchlaufen*: Suchen (Kinder, Eltern, Geschwister, Nachkommen, ...) und Filtern (Index, Untermenge, Komplement, ...) von Elementen.

[3] `http://developer.yahoo.com/yui`
[4] `http://www.jquery.com`
[5] `http://www.dojotoolkit.org`

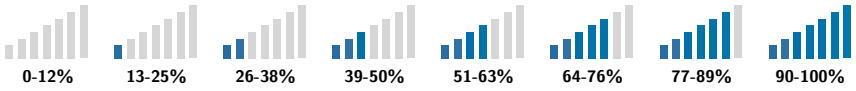


**Abbildung 6.6:** Darstellung der Relevanz-Prozentzahl mit leeren und gefüllten Balken

- *DOM-Baum verändern*: Einfügen (innerhalb/außerhalb, vorher/nachher, Umschließen, Ersetzen), Entfernen und Kopieren von Elementen.
- *CSS*: Zugriff auf und Änderung von CSS-Eigenschaften.
- *Ereignisse*: (De-)Registrieren von Funktionen zur Ereignisbehandlung.
- *Effekte*: Anzeigen/Verstecken, Ein-/Ausrollen, Ein-/Ausblenden von Elementen und benutzerdefinierte Animation von Eigenschaften.
- *Asynchronous JavaScript and XML (AJAX)*: HTML-Code, Scripts und JavaScript Object Notation (JSON)-Dokumente asynchron laden bzw. Daten senden.

jQuery unterstützt mit Internet Explorer ab Version 6, Mozilla Firefox ab Version 2, Apple Safari ab Version 2 und Opera ab Version 9 alle gängigen Browser. Ausschlaggebend war auch, dass sich mit dieser Bibliothek im Gegensatz zu anderen (z. B. Yahoo! User Interface Library (YUI)) sehr kompakter Code schreiben lässt, der durch diese Kürze aber nicht an Lesbarkeit verliert, sondern im Gegenteil intuitiver und mehr wie natürlichsprachliche Sätze zu lesen ist.

### Relevanzanzeige

Um dem Benutzer ein Gefühl zu geben, wie hoch die von Prospector berechnete Relevanz eines Ergebnisses ist, wurde bisher immer die genaue Prozentzahl neben dem Ergebnis angezeigt. Es stellte sich jedoch die Frage, ob diese von den Benutzern verstanden würde. Ohne das Wissen, dass 50% Indifferenz, alles darüber mehr und darunter weniger Interesse bedeutet, ist die Prozentzahl nicht intuitiv zu interpretieren.

Aus diesem Grund wurde beschlossen, die Relevanz durch eine Reihe leerer bzw. gefüllter Balken aufsteigender Größe darzustellen, *ähnlich der Anzeige der Signalstärke bei Mobiltelefonen*. Um eine ausreichend feine Abstufung zu erhalten, wurde die Zahl der Balken mit sieben festgelegt. In Abbildung 6.6 sind alle acht möglichen Zustände und die Relevanz-Bereiche, in denen sie auftreten, dargestellt.

Bei der Implementierung gab es noch zwei weitere *Anforderungen*. Als erstes sollten die Balken, die nicht gefüllt sind, trotzdem sichtbar sein, um für die gefüllten einen Referenzpunkt zu bilden. Durch die Verwendung eines hellen Grautons wurde dies erreicht. Als zweites sollte sichergestellt werden, dass ab einer gerundeten Relevanz von 51% vier der sieben Balken gefüllt sind. Damit soll eine klare und für den Benutzer sichtbare Unterscheidung zwischen den nicht klassifizierten oder nicht durch Bewertungen betroffenen Ergebnissen, die alle die


**Abbildung 6.7:** Anzeige eines Ergebnisses mit Anpassungen für Prospector

neutrale Relevanz 50% erhalten, und jenen, die bereits eine kleine positive Relevanz besitzen, getroffen werden. Dividiert man die gerundete Relevanz-Prozentzahl durch den experimentell ermittelten Wert 12,75, so ergibt sich die korrekte Zahl der gefüllten Balken.

Implementiert ist die Anzeige der Relevanz in `FancyDocumentSerializer`, jener Serialisierungskomponente, die für die Erzeugung des HTML-Codes zur Präsentation der Ergebnisliste verantwortlich ist. Für jedes Ergebnis wird zuerst aus dem `RawDocument`-Objekt mit `getScore()` die Relevanz extrahiert. Diese wird in eine ganzzahlige Prozentzahl umgewandelt und damit die Anzahl der gefüllten Balken berechnet. Die Balken werden als HTML-Bildverweise (`img`-Element) im Code eingebunden, wobei für jede Balkengröße je eine gefüllte und eine ungefüllte Version zur Verfügung steht. Über das `title`-Attribut der HTML-Elemente wird die Relevanz in Form einer Prozentzahl als Tooltip definiert und so beim Verweilen mit der Maus über den Balken angezeigt. Ein Beispiel dafür, wie die Relevanzanzeige schlussendlich in der Ergebnisliste dargestellt wird, ist in Abbildung 6.7 zu sehen.

### Themenklassifizierungen

Durch die Komponente zum Clustern (siehe 6.3.1) werden die Klassifizierungen der Ergebnisse in einer Hierarchie zusammengefasst und durch die Web-Anwendung von Carrot$^2$ automatisch links neben der Ergebnisliste als Baum angezeigt. Dennoch waren für Prospector einige Anpassungen in der Oberfläche nötig.

Befindet sich in Carrot$^2$ der Mauszeiger über dem Titel eines Ergebnisses, so werden in der Cluster-Hierarchie die Namen all jener Cluster gelb hinterlegt, in denen das Ergebnis eingeordnet ist. Durch Betätigen eines Symbols rechts neben dem Titel kann diese Hervorhebung dauerhaft gemacht werden. Sie bleibt dann für ein Ergebnis sichtbar, auch wenn der Mauszeiger nicht mehr über dessen Titel positioniert ist.

Auch für Prospector erschien diese Funktionalität nützlich, um dem Benutzer in der Menge der Klassifizierungen zu zeigen, wo ein Ergebnis eingeordnet ist. Allerdings bedurfte es einiger Änderungen und Erweiterungen. Schon bisher wurde beim Ergebnis der Pfad zu jenem Thema, in dem ein Ergebnis klassifiziert war, angezeigt. Diese Anzeige, die früher bei einigen Suchmaschinen ebenfalls in der Ergebnisliste zu finden war, wurde auch in der Carrot$^2$-Weboberfläche eingebaut. Die einzelnen Teile des Pfades sind durch „>“-Symbole getrennt, um die Ordnung der Themen im Pfad hervorzuheben (siehe Abbildung 6.7).

Anstatt wie bisher beim Titel werden die Themen, in die das jeweilige Ergebnis klassifiziert ist, in der Cluster-Anzeige gelb hinterlegt, wenn sich der Mauszeiger über dem Klassifizierungspfad befindet. Das Symbol neben dem Titel zum dauerhaften Hervorheben wurde entfernt. Stattdessen wird diese Funktion aktiviert bzw. deaktiviert, wenn man auf den Pfad klickt. Gleichzeitig wird dieser Pfad in der Klassifikationshierarchie vollständig geöffnet. Realisiert ist dies durch eine rekursive JavaScript-Funktion. Diese startet beim untersten Knoten des Pfades und arbeitet sich bis zur Wurzel hoch. Auf jeder Ebene „klappt“ sie den Zweig aus und macht so die Kindknoten sichtbar. Ein nochmaliges Klicken des Pfads bei einem Ergebnis bewirkt das Gegenteil und schließt die ausgeklappten Zweige des Baums.

Die Klassifizierungen im ODP sind oft sehr feingranular, und daher sind die Pfade meist sehr lang. Dabei kommt es vor, dass ab einer gewissen Ebene in der Klassifizierungshierarchie keine Verzweigungen mehr auftreten. Um dem Benutzer in diesem Fall zu ersparen, jede Ebene einzeln auszuklappen, wurde folgender Mechanismus implementiert: beim Ausklappen einer Ebene wird so lange weiter in die Tiefe der Hierarchie ausgeklappt, solange der Kindknoten die selbe Zahl an Dokumenten enthält. Für jedes Thema in der Hierarchie ist die kumulierte Zahl der Dokumente, die in ihm oder einem untergeordneten Thema klassifiziert sind, bekannt. Mit der zuvor genannten Heuristik lassen sich so Verzweigungen ebenso erkennen wie auch, wenn die Menge der verbleibenden Dokumente auf unterschiedlichen Ebenen klassifiziert ist.

#### Benutzer berücksichtigen

Die bisher beschriebenen Anpassungen und Erweiterungen berücksichtigten die Identität des Benutzers noch nicht. Um die Ergebnisse personalisieren zu können, muss diese jedoch in allen Phasen der Interaktion bekannt sein. Zu diesem Zweck wurde, wie von Gauch et al. [GSCM07] vorgeschlagen und auch in den vergangenen Versionen von Prospector verwendet, auf eine Kombination aus expliziter Anmeldung und Nachverfolgung durch Cookies gesetzt.

**Registrieren** Die Erstanmeldung erfolgt über einen Registrierungs-Dialog, in dem der Benutzer seinen Benutzernamen und ein Passwort wählen kann. Um den Benutzern *größtmögliche Anonymität* zu bieten, werden noch im Browser mit JavaScript die Message-Digest algorithm 5 (MD5)-Hashwerte dieser beiden Angaben gebildet und nur diese zum `RegisterServlet` übertragen. Dort wird der Benutzer mittels schon vorhandenem Code aus vergangenen Versionen von Prospector registriert, sofern er nicht schon existiert. Ein leeres Benutzermodell wird angelegt und seine Affinitäten zu den Gruppen werden mit 0 festgelegt. Wichtige Neuerung: durch das Hashen kann nicht einfach auf einen leeren Benutzernamen oder ein leeres Passwort überprüft werden. In beiden Fällen muss zur Validierung der Benutzereingaben mit `d41d8cd98f00b204e9800998ecf8427e`, dem MD5-Hashwert der leeren Zeichenkette verglichen werden.

**Einloggen** Das Anmelden läuft ähnlich ab, nur dass hier das `LoginServlet` die Korrektheit des angegeben Passworts prüft. In beiden Fällen wird anschließend ein *Cookie mit dem*

*Hashwert des Benutzernamens* als Identifikation zurückgegeben. Mit diesem kann sich der Benutzer bei der weiteren Interaktion automatisch authentifizieren. Selbstverständlich handelt es sich hierbei um keine hundertprozentig sichere Lösung, die vor dem Übernehmen eines fremden Benutzerprofils schützt. Durch das Erzeugen eines Cookies mit dem Hashwert des Benutzernamens der Zielperson könnte dies schnell bewerkstelligt werden. Sollte das Passwort (auch als Hashwert) nicht im Cookie gespeichert werden, so würde Abhilfe schaffen, einen eindeutigen Zufallswert als Authentifizierungsmerkmal am Server zu generieren und im Cookie zurückzugeben. Diese Information müsste für jeden Benutzer gespeichert und beim Übermitteln des Cookies geprüft werden. Da es sich bei Prospector um ein Forschungssystem handelt, hatte dies jedoch nicht Priorität, müsste aber vor einer Veröffentlichung entsprechend implementiert werden.

**Ausloggen** Beim Abmelden vom System wird vom `LogoutServlet` das Cookie vom Rechner des Benutzers entfernt und eine automatische Weiterleitung auf die Index-Seite von Prospector gemacht. Sowohl zum Registrieren und Anmelden als auch für die Bestätigungsrückfrage beim Abmelden werden modale, mit JavaScript erzeugte Dialoge verwendet. jQuery bietet hier entsprechende Bibliotheksfunktionen und erlaubt auch ein Verändern des Aussehens dieser Dialoge mit CSS.

**Dialoge** Der Einsatz von derartigen Dialogen wurde beschlossen, weil sie im Gegensatz zu HTML-Seiten keine neue Seite laden und beim Schließen den bisherigen Seiteninhalt ohne Verzögerung wieder sichtbar machen. Dies ist insbesondere dann interessant, wenn der Benutzer versehentlich den Dialog geöffnet hat, oder nach einer erfolglosen Anmeldung wieder zurück zu bereits angezeigten Ergebnissen will. Voraussetzung, dass dies alles funktioniert, ist eine asynchrone Kommunikation mit den Servlets, wie sie AJAX bietet. Sowohl beim Registrieren als auch beim Anmelden werden die Daten durch eine JavaScript-Funktion im Hintergrund übermittelt und erst bei der Rückmeldung des Servlets der Inhalt der Seite verändert. So ist es auch möglich, Fehlermeldungen direkt und ohne Verzögerungen im Dialog anzuzeigen.

**Personalisierung** Beim eigentlichen Suchvorgang müssen die in den Modellen gespeicherten Interessen des angemeldeten Benutzers nun berücksichtigt werden. Zu diesem Zweck wird bei einer Anfrage für die Ergebnisliste (und nicht bei einer für die Klassifikationshierarchie) im `QueryProcessorServlet` zu Beginn die Information im Login-Cookie abgefragt. Wurde ein entsprechender Benutzer gefunden, werden seine Daten (Modell, Affinitäten) geladen. Diese werden unter einem festgelegten Namen als einer jener Parameter gespeichert, die dem Controller der Verarbeitungskette übergeben werden. Diese Parameter können von jeder Komponente gelesen werden, und somit hat auch die `ProspectorLocalFilterComponent` die nötigen Informationen zum Benutzer.

**Ergebnisanzeige** Auch bei der Anzeige der Ergebnisse muss berücksichtigt werden, ob ein Benutzer angemeldet ist oder nicht. Die Relevanzen werden nur bei angemeldeten Benutzern

angezeigt, ebenso wie die Bedienelemente zum Bewerten. Gesteuert wird dies im `FancyDocumentSerializer`, der Serialisierungskomponente für die Ergebnisliste. Der Zugriff (Betrachten und Verändern) auf das Benutzermodell sowie die Einstellungen mit den Affinitäten zu den Gruppen ist auch nur für angemeldete Benutzer möglich und sinnvoll. Die Unterscheidung wird hier bei der Transformation des serialisierten Extensible Markup Language (XML)-Codes zum HTML-Code der gesamten Such- und Ergebnisseite von Carrot$^2$ getroffen. Abhängig davon, ob anhand der Authentifizierungsinformation (d. h. im Cookie) in der HTTP-Anfrage ein angemeldeter Benutzer festgestellt werden konnte, wird dessen Name in der XML-Definition dieser Seiten gespeichert. Ist ein solcher Name definiert, so wird bei der Transformation anderer Code (z. B. Link „Ausloggen“) erzeugt als bei einem anonymen Benutzer (z. B. Link „Einloggen“).

#### Bewerten

Bisher war das Bewerten eines Ergebnisses in Prospector nur während des Betrachtens der entsprechenden Seite im darüber angezeigten Bewertungsbereich möglich. Die Evaluierung in den Niederlanden hat jedoch gezeigt, dass diese Möglichkeit nur unzureichend genützt wird. Mehr Zuspruch fand das negative Bewerten durch Betätigen der „Unsuitable“-Schaltfläche neben dem jeweiligen Ergebnis (siehe 5.2.2). Aus diesem Grund wurde entschieden, Bewertungen nur mehr über die Ergebnisliste zu ermöglichen und dort auch positive Rückmeldungen zu registrieren.

Als Symbole für die Schaltflächen zum Bewerten wurden eine grüne, mit dem ausgestreckten Daumen nach oben zeigende geballte Hand für positive und eine rote Hand mit dem Daumen nach unten für negative Bewertungen des Ergebnisses gewählt (siehe Abbildung 6.7 auf Seite 93). Entsprechende Tooltips, die beim Verweilen mit dem Mauszeiger über den Symbolen erscheinen, erklären die Bedeutung näher: „Mehr Ergebnisse wie dieses“ für positive und „Passt nicht auf meine Suche“ für negative Bewertung. Ist das Ergebnis nicht klassifiziert, so kann es auch nicht bewertet werden. In diesem Fall sind die Symbole inaktiv, werden grau dargestellt und der Tooltip weist darauf hin, dass eine Bewertung nicht möglich ist, weil das Ergebnis in keiner Themenkategorie ist.

Anand und Mobasher [AM05] nennen ein Problem, das auch in Studien gezeigt wurde: ohne greifbaren Nutzen für die Benutzer betrachten diese mehr Ergebnisse als sie schlussendlich bewerten. Da die Adaption in Prospector von den Bewertungen lebt, sollte sichergestellt werden, dass die Benutzer auch wirklich bewerten. Aus diesem Grund „pulsieren“ die Daumen-Symbole leicht, wenn der Benutzer vom Betrachten eines Ergebnisses zur Liste zurückkehrt und das Ergebnis noch nicht bewertet hat. Dies geschieht auch beim Verwenden der Seiten-Vorschau. Diese Funktion war bereits in Carrot$^2$ eingebaut und wird durch das Lupen-Symbol rechts neben dem Titel aktiviert bzw. deaktiviert (siehe Abbildung 6.7 auf Seite 93). Die Seite des entsprechenden Ergebnisses wird dabei in einem IFRAME, das unterhalb des Snippets positioniert ist, geladen und erlaubt so eine Vorschau auf den Inhalt des Ergebnisses.

Realisiert wurde das Pulsieren der Daumen-Symbole mithilfe der Animations-Funktionen von jQuery. Vier mal werden diese von 100% Deckkraft auf 30% umgeblendet und wieder zurück. Dieser Effekt erschien hinreichend unaufdringlich, kann den Fokus aber trotzdem auf die Schaltflächen zum Bewerten lenken. Ausgelöst wird das Pulsieren nach dem Betrachten eines Ergebnisses. Dies kann auf vier verschiedene Arten geschehen:

- *Ergebnis in neuem Fenster geöffnet*: Carrot[2] verfügte bereits über ein Symbol mit einem Pfeil nach rechts oben, gleich neben dem Titel, welches beim Betätigen die Ergebnisseite in einem neuen Fenster öffnet (siehe Abbildung 6.7 auf Seite 93). In JavaScript wurde für diesen Klick eine Ereignisbehandlungsfunktion registriert. Sie registriert ihrerseits eine Funktion, die genau einmal ausgeführt wird, wenn das aktuelle Fenster wieder den Fokus erhält, und dann die Daumen-Symbole nach 500ms alle 400ms pulsieren lässt. Schließt der Benutzer nach dem Betrachten der Ergebnisseite deren Fenster, so wird dieses Fokus-Ereignis ausgelöst.

```
$(".lnw").click(function (e) {
   $(window).one("focus", function () {
      flashRatingBox(e.target, 500, 400);
      return false;
   });
});
```

- *Ergebnis im Hintergrund geöffnet*: viele Browser unterstützen das Öffnen eines Links im Hintergrund, beispielsweise in einem neuen Tab, ohne den aktuellen zu verlassen. Zur Erkennung einer derartigen Operation, die teilweise durch einen Klick mit dem Mausrad aktiviert wird, muss man am Link eine Ereignisbehandlungsfunktion für das Betätigen der Maustaste (`mousedown`) registrieren, da hier kein herkömmliches Klick-Ereignis erzeugt wird.

```
$(".lsw").mousedown(function (e) {
   $.cookie("clicked", getResultID(this));

   $(window).one("blur", function () {
      $(window.top).one("focus", function () {
         flashRatingBox(e.target, 100, 400);
         return false;
      });
      return false;
   });
   return true;
});
```

  Wichtig ist, dass erst beim Verlassen des aktuellen Fensters/Tabs (`blur`) ähnlicher Code wie im vorangegangenen Code-Beispiel ausgeführt wird. Wenn der Benutzer nämlich mehrere Links im Hintergrund öffnet, würden die Daumen-Symbole ansonsten schon

frühzeitig pulsieren. Tests haben gezeigt, dass das Pulsieren bei der Rückkehr von im Hintergrund geöffneten Ergebnissen nicht hundertprozentig funktioniert. Die Erkennung des Aktivierens des Fensters/Tabs mit der Ergebnisliste ist nicht zuverlässig und auch vom Browser abhängig. Da es sich aber um keine grobe Beeinträchtigung der Kernfunktionalität handelt, wurde diesem Mangel nicht viel Relevanz beigemessen.

- *Ergebnis im selben Fenster geöffnet*: betätigt der Benutzer den Link einfach mit der Maustaste, so wird die Ausführung von JavaScript gestoppt und alle Ereignisbehandlungsroutinen gehen verloren. Ein Ansatz wie in den vorangegangenen beiden Beispielen ist also nicht möglich. Die Lösung ist, ein Cookie mit der Identifikation des angewählten Ergebnisses zu speichern und dieses bei der Rückkehr zur Ergebnisliste wieder auszulesen.

  Das Speichern erfolgt im vorigen Code-Beispiel unter dem Cookie-Namen `clicked`. Kehrt der Benutzer zur Ergebnisliste zurück, so wird deren JavaScript-Code erneut ausgeführt. In dem Teil zur Initialisierung der Seite wird dabei das Cookie ausgelesen und überprüft, ob es Daten enthält (siehe nächstes Code-Beispiel). Ist dies der Fall, wird es zurückgesetzt, in der Ergebnisliste an die Position des betrachteten Ergebnisses gescrollt und das Pulsieren aktiviert.

```
var clickedResult = jQuery.cookie("clicked");
if (clickedResult) {
   jQuery.cookie("clicked", null);
   jQuery.scrollTo("#d" + clickedResult, {offset:-90, onAfter:
     function(){
       flashRatingBox($("#rating_box" + clickedResult), 500, 400);
     }
   });
}
```

- *Vorschau geöffnet/geschlossen*: sowohl beim Öffnen als auch beim Schließen der Seitnvorschau pulsieren die Daumen-Symbole, um den Benutzer darauf hinzuweisen, die soeben gesehene Seite zu bewerten.

Das eigentliche Bewerten beim Betätigen eines der Daumen-Symbole erfolgt asynchron. An das `RateServlet` werden dabei der Index des Ergebnisses in der Liste und die Bewertung (positiv oder negativ) übergeben. Anhand der in der Session gespeicherten Ergebnisliste kann dieses das ursprüngliche Ergebnisobjekt mit dem Index laden und samt der Bewertung an den Prospector-Algorithmus zum Aktualisieren der Modell übergeben. Anschließend werden die Daumen-Symbole deaktiviert, der gewählte Daumen verstärkt und der andere verblasst dargestellt und in einem Cookie gespeichert, wie dieses Ergebnis bewertet wurde. Dieses Cookie wird beim Aufbau der Ergebnisliste ausgelesen, und die entsprechenden Ergebnisse werden mit ihren Bewertungen markiert, wenn der Benutzer beispielsweise von einem Link zurückkehrt. Bei einer neuen Suche, wird das Cookie entfernt.

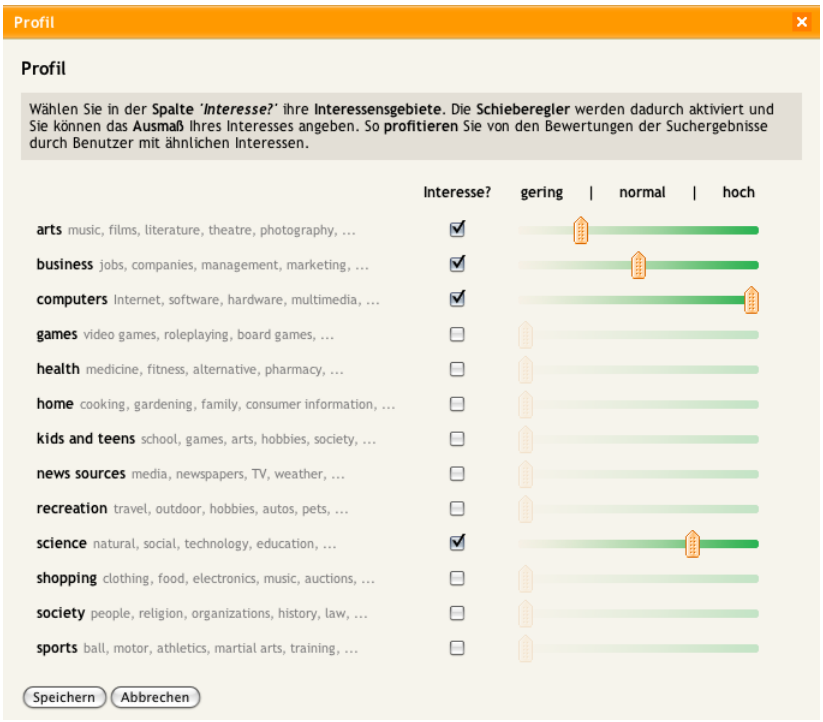


**Abbildung 6.8:** Profil zum Festlegen der Affinitäten zu Gruppen

### Profil und Modell

Schon die bisherigen Versionen von Prospector verfügten über eine Möglichkeit zum Festlegen der Affinitäten zu den einzelnen Gruppen im so genannten Profil und zum Betrachten und Bearbeiten der Inhalte des Benutzermodells. Diese Funktionalitäten wurden erneut implementiert und gleichzeitig verbessert. Grundlage dafür waren unter anderem die Erkenntnisse aus der Evaluierung in den Niederlanden (siehe 5.2.2).

Die Profil-Seite zeigte bisher die Namen der Gruppen mit jeweils sechs Optionsfeldern („Kein Interesse“ und die Zahlen von eins bis fünf) zur Auswahl der gewünschten Affinität (siehe Abbildung 5.8). Die erste Änderung war, das Feld für kein Interesse in ein Kontrollkästchen umzuwandeln und mit der positiven Bedeutung „Interesse“ zu versehen. Wählt man es an, kann man das Ausmaß des Interesses angeben. Dieses wird mit einem Schieberegler festgelegt, der an fünf Stellen einrastet und bei keinem Interesse deaktiviert und halb ausgeblendet wird.

Farblich unterstützt wird das Verständnis von der Stärke der Affinität durch einen grünen Farbverlauf, der von links nach rechts von ganz blass bis kräftig grün geht. In Abbildung 6.8 sieht man ein Beispiel für die neue Profil-Ansicht. Ebenfalls neu sind zusätzliche Informationen zu den Gruppennamen. Diese haben bisweilen aufgrund ihrer teils geringen Spezifizität für Verwirrung gesorgt. Durch das Anführen von einigen Unterthemen aus dem ODP-Verzeichnis soll hier Klarheit geschaffen werden.

Angezeigt wird das Profil nach Betätigen des entsprechenden Links „Profil“ im rechten oberen Bereich der Web-Oberfläche von Prospector (siehe Abbildung 6.12 auf Seite 105) in einem JavaScript-Dialog, wie er auch schon zum Registrieren oder Anmelden verwendet wurde. Die Daten dazu, welche Gruppen vorhanden sind, und wie stark die Affinität des Benutzers dazu jeweils ist, werden vom `ProfileManageServlet` mit AJAX nachgeladen. Die Übertragung der Daten erfolgt im JSON-Format. Dieses erlaubt die strukturierte Kodierung von Daten in Name-Wert-Paaren und geordneten Listen. Das Format eignet sich besonders für die Web-Entwicklung, da es für alle gängigen Programmiersprachen Bibliotheken zum Kodieren von Daten gibt, und JavaScript sehr leicht JSON-Code in JavaScript-Objekte und Arrays umwandeln kann [JSO08].

Ein Beispiel für die in JSON übertragenen Daten findet sich in Code 6.5. Es handelt sich hier um eine Liste (umschlossen von eckigen Klammern) von Objekten (innerhalb der geschwungenen Klammern) mit Attributen und Werten getrennt durch einen Doppelpunkt. In JavaScript kann nach dem Parsen auf die Daten in der Form `data[1].weight` zugegriffen werden. Mit diesen Informationen wird das Formular für das Profil aufgebaut, und die Kontrollkästchen und Schieberegler werden mit den entsprechenden Werten initialisiert. Beim Speichern werden die vom Benutzer vorgenommenen Änderungen gesammelt, in JSON umgewandelt und wieder an das `ProfileManageServlet` gesendet, wo diese sie dauerhaft speichern.

```
[
   {
      "group": "arts",
      "relationId": 1220516530773,
      "description": "music, films, literature, theatre, photography",
      "weight": 2
   },
   {
      "group": "business",
      "relationId": 1220516530774,
      "description": "jobs, companies, management, marketing",
      "weight": 3
   },
   ...
]
```

**Code 6.5:** JSON-Code mit Informationen zu Gruppen und Stärke der Affinität

Auch bei der Seite zum Betrachten und Verändern des Benutzermodells gibt es Neuerungen. Sie wird nun ebenfalls in einem Dialog angezeigt und der Inhalt wird asynchron geladen. Die Anzeige der Prozentzahlen, die dem Interesse an dem jeweiligen Thema entsprechen, wurde entfernt. Sie trägt nur wenig zum Verständnis bei und sollte dank des internen, festen Wertebereichs zwischen 0 und 100% und der Verwendung von Schiebereglern nicht mehr nötig sein. Als nichttextueller Hinweis auf die Stärke des Interesses bzw. Desinteresses wurde stattdessen der Farbverlauf von rot zu transparent und zu grün eingebaut. Neu ist auch die Schaltfläche zum Löschen eines Teilbaums. Sie blendet den Knoten und alle seine Unterknoten

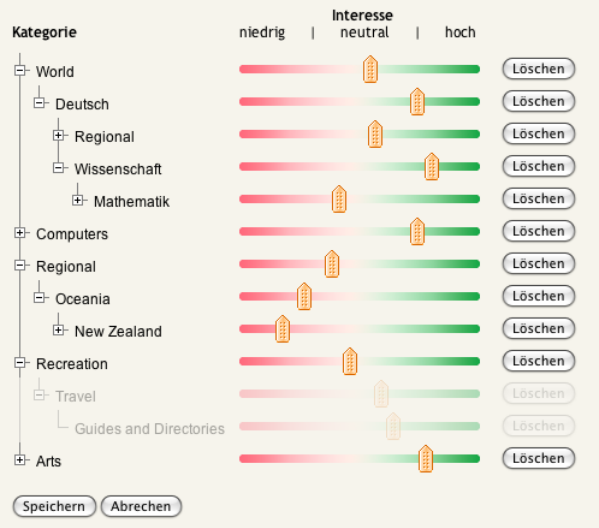


**Abbildung 6.9:** Benutzermodell zum Betrachten und Verändern der Suchinteressen

halb aus und lässt sie beim Speichern im Modell entfernen. Zu sehen sind all diese Änderungen in Abbildung 6.9.

Die Daten zum Modell erhält der Modell-Dialog wie das Profil als JSON-Code vom `ModelManageServlet`. Änderungen durch den Benutzer sendet es an dieses zurück, wo sie dauerhaft im Modell gespeichert werden. Auch diese Kommunikation geschieht über JSON.

### Dynamisches Umreihen

Ebenfalls Neuerungen hat es bei der Funktion zum Umreihen der Ergebnisse nach einem einzigen Gruppenmodell gegeben. Bisher war es nur möglich, die entsprechende Gruppe auszuwählen und eine Suchanfrage zu senden. Bei diesem Schritt wurde die gesamte Seite neu geladen. Bei der in Carrot[2] integrierten Version von Prospector wird dies vermieden, um Datenvolumen zum Herunterladen zu sparen und somit die Bedienung der Anwendung flüssiger zu gestalten. Es müssen also lediglich die angezeigten Ergebnisse umsortiert werden; die Klassifizierungshierarchie und der Rest der Seite bleiben davon unberührt.

Möglich wurde dies einmal mehr durch eine asynchrone Anfrage an das `RerankServlet`. Diese wird ausgelöst, wenn der Benutzer in der Auswahlliste rechts neben dem Suchformular eine andere Reihung wählt. Die zur Auswahl stehenden sieht man in Abbildung 6.10 auf der nächsten Seite, die gerade gewählte Reihung ist fett hervorgehoben. Ist der Benutzer nicht beim System angemeldet, so stehen alle Auswahlmöglichkeiten außer „Für Sie personalisiert“ zur Verfügung und erlauben auch anonymen Nutzern, Vorteile aus der Benutzung von Prospector zu ziehen.

Beim Wählen einer Reihung, wird diese Information im Hintergrund an das `RerankServlet` gesendet. Dieses lädt die Ergebnisliste der vorangegangenen Suche aus der Sitzung und lässt sie vom Prospector-Algorithmus entsprechend des gewählten Modells neu reihen. Als Ergebnis

liefert es eine JSON-Liste, wo für jedes Ergebnis (identifiziert über seinen ursprünglichen Index) der neue Index und die berechnete Relevanz angegeben sind.

Das eigentliche Umsortieren in der Webseite erfolgt mittels JavaScript. Anhand des ursprünglichen Index kann für jedes Ergebnis das entsprechende HTML-Element gefunden werden. Bei diesem werden der angezeigte Index und die Relevanz auf die neuen Werte geändert. Anschließend werden alle Ergebniselemente aus der Seite entfernt, in einer Liste nach den neuen Indizes sortiert und wieder eingefügt. Um den Benutzer zu informieren, dass die Operation im Gange ist, wird währenddessen neben der Auswahlliste zum Umreihen eine sich drehender Balken angezeigt.

Da der originale Index in der Element-ID der Ergebnis-Elemente verwendt wird, muss beim Umreihen „In originaler Reihenfolge“ gar keine Anfrage an das `RerankServlet` gestellt werden. Es wird einfach unter der Verwendung dieses originalen Index wie oben beschrieben umsortiert. Anschließend werden die angezeigten Relevanzen ausgeblendet, da sie in der originalen Reihenfolge keinen Sinn ergeben. Eine weitere Optimierung betrifft den Wechsel von der originalen Reihenfolge zur zuvor gewählten Reihenfolge. Auch hier ist die Index-Information noch bei den Elementen gespeichert, es muss also nur nach ihr umsortiert und die Relevanzen wieder angezeigt werden. Gespeichert wird die Information, welche Reihung zuvor gewählt war, in einem Cookie.

### OpenSearch Suchmaschinen-Definition

Die meisten modernen Browser verfügen mittlerweile über Eingabefelder, mithilfe derer direkt Anfragen an eine bestimmte Suchmaschine gestellt werden können. Internet Explorer ab Version 7 und Mozilla Firefox ab Version 2 unterstützen darüber hinaus die OpenSearch Beschreibung für Suchmaschinen [Ope08]. Mit dieser können in einem XML-Format Suchmaschinen mit deren Such-URLs, Kodierrichtlinien, Bestimmungen zu Weiterverwendung der Ergebnisse und noch vielem mehr beschrieben werden. Die genannten Browser können derartige Definition lesen und auf die jeweiligen Suchfunktionalitäten direkt zugreifen.

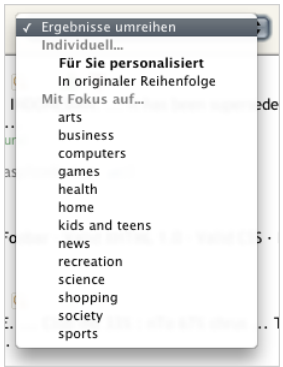


**Abbildung 6.10:** Auswahl der Reihung für das dynamische Umreihen

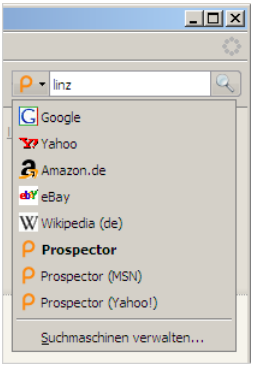


**Abbildung 6.11:** Auswahl der zu verwendenden Suchmaschine im Browser

Für Prospector wurden OpenSearch-Definitionen erstellt – eine für jede Suchmaschine, die im Hintergrund die Ergebnisse liefert. Damit sollte sichergestellt werden, dass Prospector ebenso bequem zu benutzen ist, wie bestehende Suchmaschinen. Die einfache Auswahl und Verwendung ist in Abbildung 6.11 zu sehen. Eine der erstellten OpenSearch-Definitionen findet sich als Beispiel in Code 6.6.

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">

   <ShortName>Prospector</ShortName>
   <LongName>Prospector Personalized Web Search</LongName>
   <Description>Prospector lets you personalize your web searches.
   Each time you rate a result Prospector learns what you like/dislike
   and will adjust its results in future searches.</Description>
   <Tags>prospector personalized adaptive</Tags>
   <Image height="16" width="16" type="image/vnd.microsoft.icon">
     http://prospector.fim.uni-linz.ac.at/prospector/favicon.ico
   </Image>
   <Query role="example" searchTerms="linz" />
   <Contact>koenig@fim.uni-linz.ac.at</Contact>
   <Developer>Florian Koenig and Alexandros Paramythis</Developer>
   <Attribution>
     Prospector (c) 2004-2008, Florian Koenig, Alexandros Paramythis
     Carrot2 Engine (c) 2002-2008 Stanislaw Osinski, Dawid Weiss
     Open Directory Project data kindly provided by http://www.dmoz.org
     Search Results Copyright of the respective search engine
   </Attribution>
   <Language>*</Language>
   <OutputEncoding>UTF-8</OutputEncoding>
   <InputEncoding>UTF-8</InputEncoding>
   <Url type="text/html" template="http://prospector.fim.uni-linz.ac.at
      /search?in=Web&amp;q={searchTerms}&amp;s=50&amp;alg=Prospector"/>
</OpenSearchDescription>
```

**Code 6.6:** OpenSearch-Definition für die Suche mit Prospector

Aktiviert wird die Suchmaschine im Browser durch Code 6.7. Dieser HTML-Code stellt sich in der Seite, in die er eingebunden ist, als Schaltfläche dar. Wird diese betätigt, so liest der Browser die entsprechende OpenSearch-Definition, fragt den Benutzer, ob er diese Suchmaschine wirklich hinzufügen möchte, und führt dies bei einer Bestätigung aus.

```
<button onclick="javascript:window.external.AddSearchProvider('prosp-
    opensearch.xml')">Prospector als Suchmaschine</button>
```

**Code 6.7:** OpenSearch-Definition zum Browser hinzufügen

### Sonstige Anpassungen

Um die Carrot$^2$-Weboberfläche optimal an die Verwendung mit dem Prospector-Algorithmus anzupassen, wurden weitere Änderungen und Erweiterungen vorgenommen. Dabei wurden auch, insbesondere im Hinblick auf die Evaluierung, Teile entfernt, die nicht direkt mit der Funktionalität von Prospector im Zusammenhang standen. Umgekehrt wurde die Web-Anwendung erweitert, wo dies sinnvoll erschien und einen Mehrwert schuf.

Entfernt wurde zuerst die Möglichkeit zum Wählen des Clustering-Algorithmus; Prospector wurde als Standard-Algorithmus festgelegt. Da für Prospector einige größere Änderungen an der Web-Oberfläche vorgenommen wurden und diese nur als Basis für die Weiterentwicklung diente, erschien es plausibel, ein dezidiertes Produkt zu schaffen. Die Anwendung ist mehr als nur ein einfacher Clustering-Algorithmus und auch für die Benutzer wäre eine Auswahl alternativer Algorithmen mehr verwirrend als hilfreich.

Wie in Abbildung 6.2 auf Seite 81 gezeigt, kann der Benutzer durch Anwahl stilisierter Registerblätter die zu verwendende Suchmaschine für die originalen Ergebnisse festlegen. Anders als bei ansonsten in Anwendungen verwendeten Registerblättern gab es außer bei den Registerlaschen selbst in Carrot$^2$ keine visuelle Rückmeldung zu einer derartigen Anwahl. Wurden bereits Ergebnisse angezeigt, so blieben diese unverändert weiter sichtbar. Dies wurde geändert, sodass beim Wechsel der Suchmaschine nicht mehr die Ergebnisliste angezeigt wurde. Beim Zurückwechseln wurde diese allerdings wieder sichtbar gemacht.

Um eine direkte Vergleichbarkeit und bei unpassenden Ergebnislisten gleichsam einen „Ausweg“ zu schaffen, wurde auf der Ergebnisseite der Link „Diese Suche in Google“ eingebaut. Dieser öffnet ein neues Fenster und lädt dort die Ergebnisseite der Google Web-Suche mit jener Suchanfrage, die gerade in Prospector verwendet wurde. Mit dieser Maßnahme sollte der „Umstieg“ von der dominanten Suchmaschine Google auf Prospector erleichtert werden.

Das Open Directory Project schreibt in seiner Lizenz[6] für die Verwendung der Daten vor, dass ein Hinweis auf deren Ursprung in einer bestimmten Form gemacht wird. Dieser Hinweis wurde unterhalb des IFRAME mit der Themenhierarchie eingefügt. Da an Stelle des Carrot$^2$-Logos nun ein eigenes Prospector-Logo zur Verwendung kam, wurde daneben auch ein Hinweis

[6] `http://www.dmoz.org/license.html`

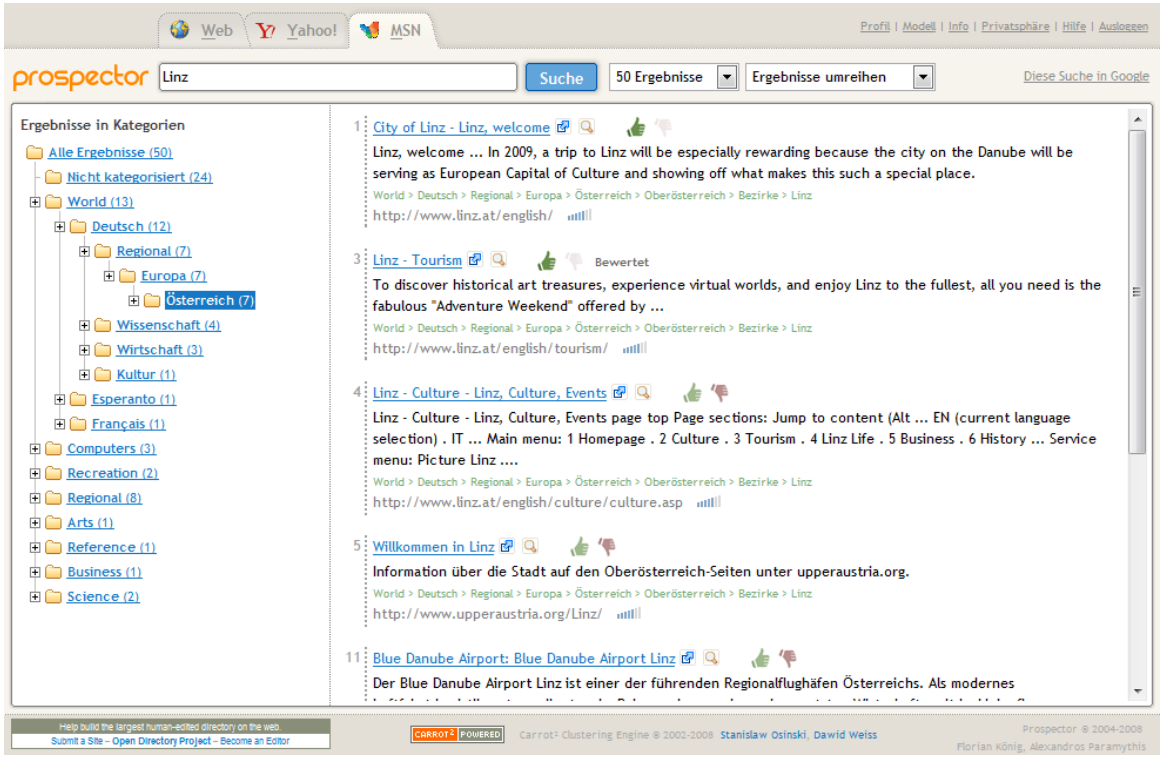


**Abbildung 6.12:** Web-Oberfläche von Prospector nach der Weiterentwicklung

auf die Urheberschaft von Carrot$^2$ gemacht. Auf der ganz rechten Seite blieb schlussendlich noch Platz für einen Urheberrechtsverweis zu Prospector.

Einen Eindruck vom finalen Erscheinungsbild der Web-Oberfläche zu Prospector erhält man in Abbildung 6.12. Zu sehen ist eine Suche nach dem Begriff „Linz“ mit MS Live Search als Quellsuchmaschine. Die Anzeige der Ergebnisse wurde auf eine links ausgewählte, thematische Kategorie eingeschränkt. Bei den Ergebnissen selbst wurden zwei bereits bewertet, das zweite gerade eben, erkennbar an dem kurze Zeit nach der Bewertung wieder ausgeblendeten Hinweis „Bewertet“.

## 6.4 Erweiterungen des Algorithmus

Im vorangegangenen Abschnitt wurde beschrieben, wie Prospector in das Carrot$^2$-Framework und dessen Web-Oberfläche eingebaut wurde. Daneben gab es noch einige Erweiterungen und Verbesserungen innerhalb von Prospector selbst. Diese sollten das System präziser, flexibler und schneller machen. Im Folgenden werden die wichtigsten Neuerungen im Bereich Klassifizierung von Suchergebnissen und bei deren Umreihung beschrieben.

### 6.4.1 Umfangreichere Klassifizierung

Eine Webseite kann im Open Directory Project mehr als nur einem Thema zugeordnet sein. Bisher wurde für jedes Suchergebnis aber nur eine Klassifizierung in ein Themengebiet vorgenommen, nämlich in jenes mit den meisten Verweisen auf Web-Inhalte. Bei der Evaluierung von Prospector in den Niederlanden hatte sich gezeigt, dass diese Heuristik nicht immer wie gewünscht funktioniert und unpassende Klassifizierungen liefert.

Aus diesem Grund wurde die Klassifizierung derart erweitert, dass nun die *Pfade zu allen Themengebieten*, denen ein Ergebnis zugeordnet ist, aus den Daten des ODP ermittelt und in der weiteren Folge berücksichtigt werden. Anstelle einer einfachen Liste von Themengebieten, die zusammen den von der Wurzel ausgehenden Pfad bilden, wird nun eine Liste von derartigen Pfad-Listen für jedes Ergebnis bestimmt. Auswirkungen hatte dies in folgenden Bereichen von Prospector:

- *Ergebnisanzeige*: sowohl bei der Anzeige der Klassifikation(en) in der Ergebnisliste als auch in der Themenhierarchie sind nun für jedes Ergebnis möglicherweise mehr als ein Pfad zu berücksichtigen. Dank seiner Ausrichtung auf Clustering unterstützte das Carrot$^2$-Framework bereits, dass ein Ergebnis mehreren Themengebieten zugeordnet ist und bei deren Auswahl in der Hierarchie auch entsprechend angezeigt wird.
- *Relevanzberechnung*: für jeden Klassifikationspfad muss anhand der Benutzer- und Gruppenmodelle eine Relevanz berechnet werden, und diese Einzel-Relevanzen müssen zu einer einzigen kombiniert werden. Die Kombination erfolgt hierbei über ein gewichtetes Mittel, wobei die Gewichte den Affinitäten des Benutzers zu den Gruppenmodellen mit den gleichen Namen wie die ersten Teile der Pfade entsprechen. Bei Themen oberster Ebene, für die keine gleichnamige Gruppe existiert, wird eine mittlere Affinität (in diesem Fall der Wert 3) angenommen.
- *Bewerten*: eine Veränderung der Gewichtungen in den Modellen muss für jeden Klassifikationspfad vorgenommen werden.

Eine weitere Neuerung betrifft die *Granularität der Klassifizierung* im Bezug auf die URL der Ergebnisse. Bisher wurden nur das Schema (z. B. „http“) und der volle Rechnername samt Domainname dazu verwendet. Dies führte dazu, dass Ergebnisse auf Sites mit vielen in den ODP-Daten verzeichneten Unterseiten immer in alle Themen klassifiziert wurden. Nun wird zuerst versucht, die vollständige URL für die Klassifikation zu verwenden. Erst wenn dies kein Ergebnis liefert, wird nach der alten Methode klassifiziert.

### 6.4.2 Verbesserte Reihung

Die zweite größere Änderung im Prospector-Algorithmus betrifft die Reihung. Mit ihr soll für die Berechnung der endgültigen Reihenfolge der Ergebnisse die *ursprüngliche Reihung stärker mit einbezogen* werden. Auch in der Literatur wird empfohlen, die ursprüngliche Reihung, die sich nur an der Relevanz im Bezug auf die Suchanfrage orientiert, einzubeziehen, und

das nicht nur, wie bisher, wenn zwei oder mehrere Ergebnisse eine gleiche Relevanz besitzen [KL05]. Es hat sich auch gezeigt, dass das Verhalten von Prospector bisweilen teils recht „sprunghaft“ und für die Benutzer schwieriger zu durchschauen war. Zwei Beispiele sollen dies veranschaulichen:

- *Schwach ausgeprägte/differenzierte Relevanzen*: dieser Effekt tritt auf, wenn Ergebnisse Relevanzen knapp über- oder unterhalb von 0,5 haben oder sich in den Relevanzen nicht stark unterscheiden. Selbst kleinste Unterschiede in den berechneten Relevanzen führen zu deren mathematischer Ungleichheit. Die Folge war daher bisher, dass die originale Reihung nicht berücksichtigt wurde. So konnte es geschehen, dass Ergebnisse mit minimal positiver(er) bzw. negativer(er) Relevanz vor bzw. nach einem anderen Ergebnis gereiht wurden, selbst wenn dieses in der ursprünglichen Reihung weit hinter bzw. vor diesem lag.
- *Anhäufung von Ergebnissen gleichen Typs*: es findet eine klare Trennung aller positiv relevanten, aller neutralen und aller negativ relevanten Ergebnisse statt. Die nicht klassifizierten Ergebnisse bilden einen Block mit Relevanz 0,5 in der Mitte der Ergebnisliste. Positiv wie auch negativ bewertete Ergebnisse machen bei der Reihung „große Sprünge um diesen Block herum“, unverhältnismäßig große in Bezug auf ihre ursprüngliche Reihung.

Zur Lösung dieser Probleme, insbesondere des ersten, böte sich mit dem Einführen eines Schwellwerts ein einfacher Ansatz an.

#### Schwellwert

Der erste Ansatz ergibt sich direkt aus dem oben beschriebenen Problem mit den schwach differenzierten Relevanzen. Der Grundgedanke dabei ist, die Relevanzen zweier Ergebnisse als gleich zu betrachten, wenn deren Differenz unter einem gewissen Schwellwert liegt. Die Folge wäre, dass beim Umreihen für die Ermittlung der relativen Position dieser Ergebnisse zueinander deren originale Reihenfolge verwendet wird.

Die Herausforderung ist nun die Wahl eines geeigneten Schwellwerts. Es muss sichergestellt werden, dass der ganzzahlige Prozentwert, der für die Anzeige der Relevanz in der Web-Anwendung verwendet wird, innerhalb der Liste monoton absteigend ist und keine „Sprünge“ aufweist. Eine erste Idee ist, den Schwellwert mit 0,005 (entspricht einem halben Prozent) zu wählen. Er soll sicherstellen, dass nur sehr ähnliche Relevanzwerte als gleich behandelt werden und die Prozentwerte zweier derartiger Suchergebnisse auch nach dem Runden absteigend geordnet sind.

Das Problem ist aber, dass dieser Schwellwert noch so klein gewählt werden kann, es gibt immer Fälle (z. B. mit den Werten 0,5049 und 0,5051), wo er unterschritten wird, aber ein Runden der Prozentwerte trotzdem unterschiedliche Ergebnisse liefert. Es wären also über diesen einfachen Ansatz hinausgehende, umfangreichere Anpassungen nötig, um ein korrektes Funktionieren zu garantieren. Mit dem als nächstes beschriebenen Ansatz treten diese

Probleme hingegen nicht auf, und er löst auch die zweite oben genannte Unregelmäßigkeit bei der Reihung. Aus diesem Grund wurde die Schwellwert-Methode nicht weiter verfolgt.

### Data fusion

In der Literatur findet sich mit „data fusion“ ein allgemeiner Begriff zu dem oben genannten Problem, zusätzlich zur Reihung nach dem Prospector-Algorithmus auch die originale Reihung mit einzubeziehen. Laut Lillis et al. [LTM+06] konnte gezeigt werden, dass im Bereich Information Retrieval für bestimmte Aufgaben jeweils einige Algorithmen bessere Ergebnisse liefern als andere. Die Kombination der Suchergebnisse unterschiedlicher Algorithmen führte in der Folge zu einer besseren Suchleistung. Lillis et al. definieren die Ausprägungen dieses Kombinationsschritts näher:

> *„The term 'data fusion' specifically refers to the use of a number of different IR systems to retrieve documents from the same document collection and the combination of their result sets using any information that is available in order to achieve superior results. This is in contrast to related tasks such as 'collection fusion', where the document collections being searched are distinct (Voorhees et al. 1994) and situations where there is only partial overlap between the document collections.“*

Besondere Forschung auf diesem Gebiet gab es im Bereich der Meta-Suchmaschinen, wie auch Prospector eine ist [LTM+06]. Renda und Straccia [RS03] unterscheiden hier die in der Literatur vorgeschlagenen Methoden zur Kombination der einzelnen Ergebnislisten, die jeweils noch weiter dahingehend unterschieden werden, ob sie trainiert werden müssen oder nicht:

- *Rang-basiert*: die Ergebnisse werden anhand ihrer Reihenfolge in den einzelnen Listen zu einer kombinierten Ergebnisliste zusammengefügt.
- *Relevanz-basiert*: die Methoden bedienen sich der von den Suchmaschinen/-algorithmen berechneten Relevanzen eines jeden Ergebnisses. Möglich ist beispielsweise eine lineare Kombination, bei der die unterschiedlichen Relevanzen eines Ergebnisses gewichtet summiert werden [LTM+06].

Experimente zeigten, dass Relevanz-basierte Methoden den Rang-basierten überlegen sind. Ebenso lieferten Methoden, die auf Trainingsdaten aufbauen, bessere Ergebnisse als jene, die ohne Training arbeiteten [RS03]. Ein Problem mit den Relevanz-basierten Methoden ist jedoch, dass die Relevanzen oft nicht bekannt sind. Um sie außerdem vergleichbar zu machen, müssen sie zuerst *normalisiert* werden [LTM+06]. Renda beschreibt dazu unter anderem folgende zwei Formeln. Beide beziehen sich auf eine Liste von Ergebnissen $\tau$ und berechnen eine normalisierte Relevanz $w^\tau(i)$ für ein Ergebnis auf Rang $i$, wobei $i \in \{1, 2, \ldots, |\tau|\}$.

$$w^\tau(i) = \frac{s^\tau(i) - \min_{j \in \tau} s^\tau(j)}{\max_{j \in \tau} s^\tau(j) - \min_{j \in \tau} s^\tau(j)} \tag{6.1}$$

Formel 6.1 auf der vorherigen Seite normiert dazu die Relevanz $s^\tau(i)$ mithilfe der minimalen und maximalen Relevanz der Ergebnisliste. Das Ergebnis mit der zuvor höchsten Relevanz hat anschließend Relevanz 1, das mit der niedrigsten Relevanz 0. Dazwischen liegen die anderen Ergebnisse entsprechend der Verteilung ihrer ursprünglichen Relevanzen [RS03].

$$w^\tau(i) = 1 - \frac{\tau(i) - 1}{|\tau|} \tag{6.2}$$

Formel 6.2 normiert den Rang $\tau(i)$ über die Kardinalität der Liste. Die Subtraktion von 1 ist nötig, da der erste Rang mit 1 anstelle von 0 definiert wurde. Die Normalisierung ergibt eine Gleichverteilung der Ränge zwischen 1 für das erstgereihte und 1 dividiert durch die Länge der Liste für das letztgereihte Ergebnis [RS03]. Die derart transformierten Ränge können nach diesem Schritt wie Relevanzen für weitere Berechnungen verwendet werden.

Eine Standardtechnik der „data fusion“ ist die von Fox und Shaw [FS94] entwickelte Methode *CombMNZ*. Dabei werden die normalisierten Relevanzen, die einem Ergebnis von den einzelnen Suchmaschinen/-algorithmen zugeordnet wurden, aufsummiert und mit der Anzahl der nicht-negativen Relevanzen multipliziert. Dies kombiniert nicht nur alle Bewertungen zu einer neuen, sondern begünstigt auch jene Ergebnisse, die von mehr Suchmaschinen/-algorithmen als relevant erkannt wurden bzw. überhaupt gefunden wurden [LTM+06]. Es ist auch möglich, die einzelnen Relevanzen unterschiedlich stark zu gewichten [RS03].

Beim Prospector-System geht es nun um die *Kombination der Ergebnisse* zweier verschiedener Suchsystemen, die auf die selbe Menge an Dokumenten zugreifen. Das eine Suchsystem ist die Suchmaschine, welche die originalen Ergebnisse liefert. Sie liefert üblicherweise nur Information über deren Ränge, nicht aber über die einzelnen Relevanzen. Das zweite Suchsystem entspricht dem Prospector-Algorithmus selbst, welcher anhand der Modelle Relevanzen berechnet. Implementiert wurde die Kombination dieser zwei Bewertungen eines jeden Ergebnisses durch folgenden *Algorithmus*:

1. Ränge der originalen Reihung mit Formel 6.2 normalisieren.
2. Berechnete Relevanzen gemäß Formel 6.1 auf der vorherigen Seite normalisieren. Diese werden anstelle der Ränge verwendet, da sie die Benutzerinterinteressen differenzierter widerspiegeln.
3. Die beiden Ergebnisse gewichtet zu einer neuen Relevanz summieren. Die Multiplikation mit der Anzahl der nicht-negativen Relevanzen wie im originalen CombMNZ kann entfallen, da es nicht vorkommen kann, dass eines der Suchsysteme ein Ergebnis nicht liefert. Diese vereinfachte Version der Kombination nennt sich *CombSUM* [FS94].
4. Kombinierte Relevanzen erneut mit Formel 6.1 auf der vorherigen Seite normalisieren. Dies ist nötig, weil das Prospector-System von Relevanzen im Bereich [0, 1] ausgeht.

Implementiert wurde dieser Algorithmus wie in Code 6.8 auf der nächsten Seite und Code 6.9 auf der nächsten Seite zu sehen ist. In der Komponente `DataFuserLocalFilterComponent`, die nach der Komponente zum Berechnen der Relevanzen eingebunden ist, werden

die eingehenden Ergebnisse über die Methode `processItem(SearchResultItem item)` an eine Instanz des `NormalisedCombSumAlgorithm` weitergegeben. Dort werden sie in einer Liste gespeichert und die minimale bzw. maximale Relevanz berechnet.

```
public class NormalisedCombSumAlgorithm implements FusionAlgorithm {

   private List items;
   private double minScore, maxScore;
   private static final double LAMBDA = 0.75;

    public NormalisedCombSumAlgorithm() {
      items = new ArrayList();
      minScore = Double.MAX_VALUE;
      maxScore = Double.MIN_VALUE;
    }

   public void processItem(SearchResultItem item)
                              throws FusionException {
      if (item != null) {
         double score = item.getScore();
         if (score < minScore) minScore = score;
         if (score > maxScore) maxScore = score;
         items.add(item);
      } else {
         throw new FusionException("Item to be processed is null");
      }
   }

   public List getFusedResult() throws FusionException {
      ...
   }
}
```

**Code 6.8:** `NormalisedCombSumAlgorithm`: Initialisierung und Sammeln der Ergebnisse

Nachdem durch die `DataFuserLocalFilterComponent` alle Ergebnisse an den Algorithmus geleitet wurden, werden sie durch den Aufruf der Methode `getFusedResult()` verarbeitet, zurückgegeben und an die nächste Komponente weitergegeben. Die Verarbeitung erfolgt in zwei Schritten: Im ersten werden die Ränge und Relevanzen normalisiert und kombiniert, im zweiten werden die neuen Relevanzen normalisiert. Diese Aufteilung ist nötig, da für den zweiten Normalisierungsschritt zuerst die maximale und minimale kombinierte Relevanz bekannt sein muss.

```
public List getFusedResult() throws FusionException {

   if (items == null || items.size() == 0)
      throw new FusionException("No items processed yet");
```

```
    // If all scores are equal no calculations are needed
    if (maxScore != minScore) {

       double minCombSum = Double.MAX_VALUE;
       double maxCombSum = Double.MIN_VALUE;
       double score, normRank, normScore, combSum;

       Iterator itemIt = items.iterator();
       while (itemIt.hasNext()) {
          SearchResultItem item = (SearchResultItem) itemIt.next();
          score = item.getScore();

          normRank = 1 - (((double) item.getOrigIdx()) / items.size());
          normScore = (score - minScore) / (maxScore - minScore);

          combSum = (1 - LAMBDA) * normRank + LAMBDA * normScore;
          item.setScore(combSum);

          if (combSum < minCombSum) minCombSum = combSum;
          if (combSum > maxCombSum) maxCombSum = combSum;
       }

       // Normalize the sum of score and rank
       itemIt = items.iterator();
       while (itemIt.hasNext()) {
          SearchResultItem item = (SearchResultItem) itemIt.next();
          item.setScore((item.getScore() - minCombSum) /
                        (maxCombSum - minCombSum));
       }
    }
    return items;
}
```

**Code 6.9:** `NormalisedCombSumAlgorithm`: Berechnung der kombinierten Relevanzen

Die so neu berechneten Relevanzen können wie zuvor als Wahrscheinlichkeiten des Interesses an dem jeweiligen Ergebnis interpretiert werden. Das erstgereihte Ergebnis hat durch den abschließenden Normalisierungsschritt eine Relevanz von 1, das letztgereihte die Relevanz 0. Für ein und das selbe Ergebnis kann aufgrund der Normalisierung bei unterschiedlichen Suchen je nach originaler Reihung der Relevanzwert schwanken. Aus diesem Grund wird in der Benutzeroberfläche weiterhin die von Prospector berechnete Relevanz angezeigt. Es kann dadurch zwar vorkommen, dass diese im Laufe der Ergebnisliste nicht monoton ansteigt, doch sie spiegelt die wahren Interessen des Benutzers auf Basis der Modelle besser wider.

## 6.5 Systemüberwachung

Bereits in Version 2 verfügte Prospector über einige Funktionen, die das System in seinem Lauf beobachten und relevante Daten mitprotokollieren. Diese Daten zur Benutzung des Sys-

tems durch die Benutzer, der Suchen und Bewertungen, die diese ausgeführt haben und der Änderungen der Modelle wurden auch teilweise in der Evaluierung verwendet. Für die neue Version sollte die Systemüberwachung erweitert und verbessert werden.

Bisher wurden die protokollierten Daten in einfache Textdateien geschrieben. Um diese nach der Evaluierung in den Niederlanden auszuwerten, wurden sie von einem Parser eingelesen und in eine Datenbank geschrieben. Dank der Möglichkeiten von SQL-Abfragen konnten sie so gefiltert, aggregiert und für weitere Analysen aufbereitet werden. Für die Weiterentwicklung der Systemüberwachung wurde daher die Entscheidung getroffen, gleich in eine Datenbank zu protokollieren. Im Folgenden wird zuerst das Objektmodell der zu protokollierenden Ereignisse und ihrer Daten vorgestellt. Anschließend wird beschrieben, wie diese Ereignisse in die Datenbank persistiert werden. Zum Schluss werden die Mechanismen zum Erzeugen, Sammeln und Abspeichern der Ereignisse erläutert.

### 6.5.1 Objektmodell

Die Basis des Objektmodells bildet eine Vererbungshierarchie von Ereignistypen. Jeder Typ verwaltet gewisse Daten und tritt bei bestimmten Aktionen auf. Darüber hinaus gibt es noch Hilfsklassen zum Speichern der Ergebnislisten. Alle Klassen und ihre Beziehungen sind in Abbildung 6.13 auf der nächsten Seite dargestellt. Die grauen Notizen geben für ein Ereignis jeweils die möglichen Aktionen an, bei denen dieses auftreten kann. Die weißen Notizen zeigen mögliche Werte für bestimmte Datenfelder eines Ereignisses.

Folgende Ereignisse werden von Prospector unterstützt:

- `AuditEvent`: ist die abstrakte Basisklasse und definiert eine automatisch generierte ID `id`, einen Zeitstempel `timestamp`, den genauen Typ `type` (entspricht der Unterklasse und wird für die Persistierung benötigt), eine Identifikation des Benutzers `userId` und, welche Aktion `action` (z. B. `login`, `search`, `rate`, ...) ausgeführt wurde.
- `UserAuditEvent`: die Oberklasse all jener Ereignisse, die direkt vom Benutzer ausgelöst werden. Die Klasse definiert keine weiteren Datenfelder, und das Ereignis tritt beim Einloggen (`login`), Ausloggen (`logout`) und Registrieren (`register`) auf.
- `ResultAuditEvent`: wird verwendet, wenn das Ereignis ein Suchergebnis in seiner Gesamtheit wie beim Suchen (`search`) oder Umreihen (`rerank`) betrifft. Es besitzt einen Verweis auf der entsprechende Ergebnis `result` und kennt dessen Reihung `order`.
- `ResultItemAuditEvent`: tritt bei Ereignissen auf, die ein einzelnes Suchergebnis betreffen. Dies ist das Betrachten des Ergebnisses (`follow`), das Öffnen in einem neuen Fenster/Tab (`open`), das Öffnen der Voransicht (`preview`) und das „Zurückkehren“ von einer der zuvor genannten Ansichten zum Ergebnis (`return`). Der Rang `index` des jeweiligen Ergebnisses in der Liste wird gespeichert.
- `RateAuditEvent`: wird durch das Bewerten (`rate`) eines Ergebnisses ausgelöst. Gespeichert wird die vom Benutzer gewählte Bewertung `bias`.

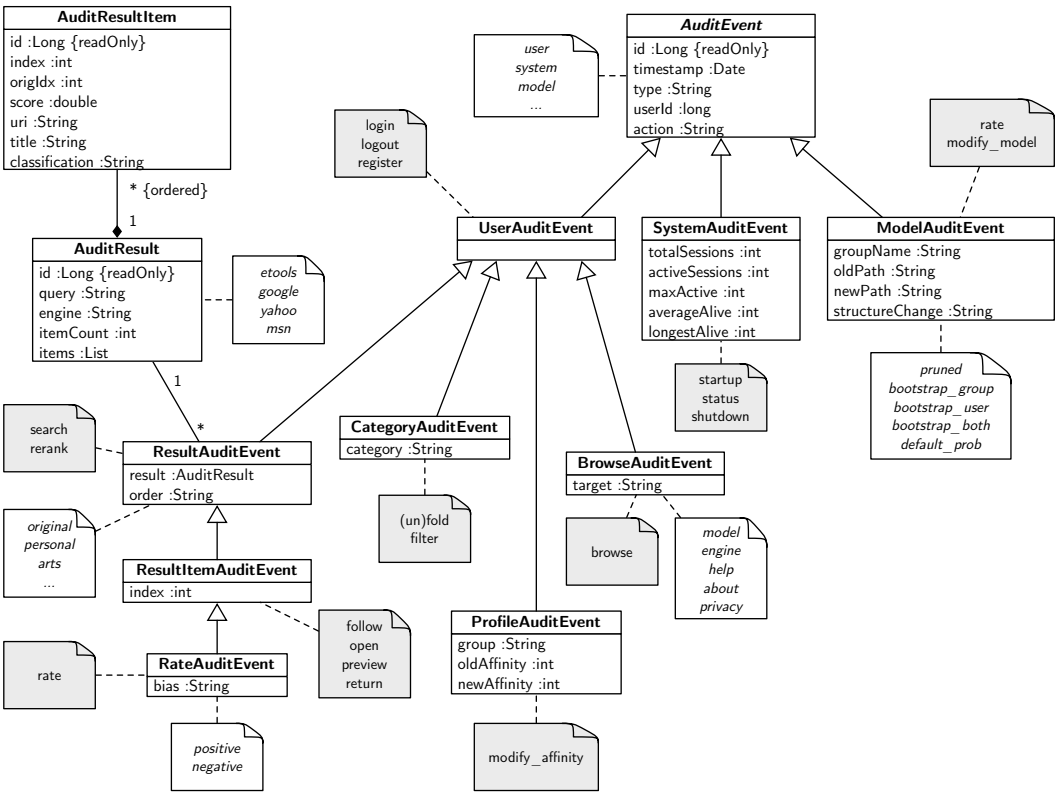


**Abbildung 6.13:** Klassenmodell der Ereignisse zur Systemüberwachung

- `CategoryAuditEvent`: wird für das Protokollieren von Benutzeraktionen in der Themenhierarchie verwendet. Wenn dieser ein Thema/eine Kategorie ausklappt (`unfold`), einklappt (`fold`) oder anwählt (`filter`), so wird der Pfad in `category` gespeichert.

- `BrowseAuditEvent`: dieses Ereignis tritt bei der Navigation in der Benutzeroberfläche (`browse`) auf und protokolliert das Ziel `target` der Benutzeraktion.

- `ProfileAuditEvent`: tritt auf, wenn der Benutzer seine Affinität zu einer Gruppe ändert (`modify_affinity`). Gespeichert werden der Name der Gruppe `group` und die alte und neue Affinität `oldAffinity` und `newAffinity`.

- `ModelAuditEvent`: bei Veränderungen in Modellen, die explizit (`modify_model`) oder durch eine Bewertung (`rate`) hervorgerufen wurden. Protokolliert werden der Gruppenname `groupName` (bei einem Gruppenmodell), der alte Pfad mit den Gewichten `oldPath`, der neue Pfad `newPath` und, ob sich eine Änderung der Struktur ergeben hat. In `structureChange` steht, ob ein Teilbaum entfernt wurde (`pruned`), welche Modelle für die Ableitung eines Gewichts konsultiert wurden (`bootstrap_(user|group|both)`), oder ob der Standardwert verwendet wurde (`default_prob`).

- `SystemAuditEvent`: wird bei Systemereignissen wie Start (`startup`) oder Herunterfahren (`shutdown`) erzeugt. Vorgesehen sind auch periodische Statusmeldungen (`status`) mit Information zur aktuellen Systembenutzung in den Datenfeldern.

### 6.5.2 Persistenz

Für die Persistierung der Ereignisobjekte wurde der *objekt-relationale Persistenz- und Abfrage-Dienst Hibernate*[7] verwendet. Dieser ermöglicht die Abildung von Objekten und ihren Beziehungen (Vererbung, Objektreferenzen, ...) auf relationale Datenbankstrukturen. Ohne datenbankspezifische Structured Query Language (SQL)-Befehle schreiben zu müssen, können so die Objekte mit einfachen Java-Befehlen in die Datenbank gespeichert und aus ihr gelesen werden. Hibernate kapselt den Zugriff zu den unterschiedlichsten Datenbankmanagement-Systemen, kümmert sich um das Anlegen des Datenbankschemas unter Verwendung der definierten Objekte und Beziehungen und bietet transaktionsorientierten Datenzugriff.

Zum *Einrichten von Hibernate* müssen neben einer allgemeinen Konfigurationsdatei mit Verbindungseinstellungen (Datenbank, Benutzer, Passwort, Cache-Optionen, ...) auch die Abbildungen der Objekte und Beziehungen auf Tabellen, Schlüssel und Constraints angelegt werden. Code 6.10 zeigt diese Konfiguration für die Klasse `AuditEvent`. Es wird festgelegt, dass die Daten in der Tabelle `event` mit einer automatisch generierten ID gespeichert werden sollen. Die Datenfelder werden als Spalten mit ihrem Typ (so er nicht automatisch erkannt wird), Längen und Einschränkungen (`NOT NULL`) definiert. Das Feld `type` wird außerdem als jenes bestimmt, welches in der Datenbank die Unterscheidung zwischen den konkreten Unterklassen von `AuditEvent` ermöglicht.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="at.jku.fim.prospector.auditing.event">
 <class name="AuditEvent" table="event">
  <id name="id" column="id">
   <generator class="native" />
  </id>
  <discriminator column="type" type="string" length="20" />
  <property name="timestamp" column="time_stamp" not-null="true" />
  <property name="userId" column="user_id" />
  <property name="action" type="string" length="20" not-null="true" />
 </class>
</hibernate-mapping>
```

**Code 6.10:** Hibernate Mapping für `AuditEvent`

Klassen in der Hierarchie der Ereignisklassen werden wie in Code 6.11 auf der nächsten Seite definiert. Die Felder, in denen diese Klasse ihre Elternklasse erweitert (hier `order`), sind das einzige, was in der jeweiligen Tabelle (`event_result`) gespeichert wird. Um ein Objekt wieder vollständig einzulesen, muss diese Tabelle mit denen ihrer übergeordneten Klassen mit einem Join über die Schlüsselspalte (`id`) verbunden werden. Über den Wert zur Unterscheidung

[7] `http://www.hibernate.org/`

der Typen (`discriminator_value` mit Wert `user_result`), der in der obersten Tabelle der Hierarchie `event` in der Spalte `type` gespeichert wird, kann jederzeit auf die Klasse für einen Datensatz geschlossen werden.

In Code 6.11 sieht man auch, wie eine *Objektreferenz* abgebildet wird. In dem Feld `result` wird auf ein Objekt vom Typ `AuditResult` verwiesen. Da ein Ergebnisobjekt in mehreren Ereignisobjekten referenziert werden kann, wird hier eine n:1-Beziehung (`many-to-one`) definiert. Beim Speichern dieses Ereignisses wird dessen Ergebnis-Objekt automatisch in seine an anderer Stelle definierte Tabelle gespeichert und über die Spalte `result_id` referenziert.

```
<hibernate-mapping package="at.jku.fim.prospector.auditing.event">
  <subclass name="ResultAuditEvent" extends="UserAuditEvent"
      discriminator-value="user_result">
    <join table="event_result">
      <key column="id" />
      <many-to-one name="result" class="at.jku.fim.prospector.auditing.
          result.AuditResult" column="result_id" not-null="true"
           cascade="save-update,persist" />
      <property name="order" type="string" length="20"
           column="rank_order" not-null="true" />
    </join>
  </subclass>
</hibernate-mapping>
```

**Code 6.11:** Hibernate Mapping für `ResultAuditEvent`

### 6.5.3 Verarbeitungslogik

Bereits in Version 2 von Prospector gab es einige Funktionen zur Systemüberwachung. Diese wurden als Basis verwendet und in vielen Aspekten weiterentwickelt. Neu war neben der oben beschriebenen Ereignishierarchie vor allem die Möglichkeit zum Protokollieren von Aktionen in der Benutzeroberfläche. Bisher wurde nur protokolliert, was ohnehin im Kern von Prospector vorging (z. B. Suchen und Bewertungen). Durch das *Mehr an Interaktivität* in der Web-Oberfläche konnten nun *zusätzliche Informationen zur Systembenutzung* gesammelt werden.

Zum anderen ergaben sich neue Herausforderungen durch das *Persistieren in eine Datenbank*. Ziel hierbei war, einen möglichst großen Durchsatz an Schreiboperationen bei gleichzeitiger niedriger Latenz sicherzustellen. Dies war nötig, da es in keinem Fall zu Verzögerungen bei den eigentlichen Systemfunktionen kommen sollte. Lösungen für beide Anforderungen werden im Folgenden präsentiert.

### Web-Oberfläche

Als Beispiel für das Protokollieren von Aktionen in der Web-Oberfläche soll das Öffnen eines Links in einem neuen Fenster/Tab dienen. Die grundlegende Ereignisbehandlungsroutine, die mit JavaScript an die entsprechenden Symbole in der Ergebnisliste gebunden wird, wurde bereits in Code 6.3.2 auf Seite 97 angeführt. Diese wurde nun, wie in Code 6.12 gezeigt, um Kommandos zum asynchronen Senden von Protokoll-Informationen mittels AJAX an ein Servlet erweitert. Dieses ist über den Namen `audit` erreichbar.

```
$(".lnw").click(function (e) {
   var link = this;
   $(window).one("focus", function () {
      flashRatingBox(e.target, 500, 400);
      auditReturn(getResultIndex(link));
      return false;
   });
   auditOpen(getResultIndex(link));
});

function auditOpen(itemIndex) {
   jQuery.post("audit", { action: "open", index: itemIndex });
}

function auditReturn(itemIndex) {
   jQuery.post("audit", { action: "return", index: itemIndex });
}
```

**Code 6.12:** Protokollieren des Öffnens eines Links in einem neuen Fenster oder Tab und der anschließenden Rückkehr zum Ergebnis

Dieses `AuditServlet` verarbeitet derartige Protokollierungen in den Bereichen Navigation (`browse`), Themenhierarchie (`unfold`, `fold`, `filter`), Umreihen (`rerank`) und Ergebnisse (`follow`, `open`, `preview`, `return`). Für diese lässt es die entsprechenden Ereignisobjekte mit den übergebenen Daten erstellen und persistieren. Alle anderen Ereignisse werden in den anderen Teilen von Prospector (Servlets, Bewertungsalgorithmus, ...) protokolliert.

### Persistierung

Die erste Herausforderung beim eigentlichen *Persistieren der Ereignis-Objekte* war, die Laufzeit für den entsprechenden Methodenaufruf so gering wie möglich zu halten. Anders als bei dem zuvor genannten Beispiel, wo durch einen asynchronen Aufruf die Interaktion des Benutzers mit dem Browser nicht beeinträchtigt wird, gibt es Situationen, wo diese Nebenläufigkeit durch andere Mittel geschaffen werden muss. In diesem Fall werden die weiteren Persistierungsschritte an einen Thread delegiert und die Methode kehrt somit schnellstmöglich zum Aufrufer zurück.

Zu sehen ist die Implementierung in Code 6.13. Um die Zeit zum Erstellen eines neuen Threads zu sparen, wird ein Thread Pool verwendet. Dieser erzeugt beim Start Threads, denen zur eigentlichen Ausführung nur mehr Objekte vom Typ `Runnable` übergeben werden. Deren Methode `run()` enthält den auszuführenden Code und wird dann aufgerufen. In diesem Fall wird dabei das Ereignis-Objekt an eine zuvor gesperrte Warteschlange übergeben und anschließend die Persistierung angestoßen.

```
public void queueEvent(final AuditEvent event) {
   if (!queueThreadPool.isShutdown()) {
      queueThreadPool.execute(new Runnable() {
         public void run() {
            try {
               queueLock.lockInterruptibly();
               eventQueue.offer(event);
               attemptPersisting(BATCH_SIZE, BATCH_SIZE, true);
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
      });
   }
}
```

**Code 6.13:** Übergabe eines Ereignis-Objekts zum Persistieren

In diesem letzten Schritt steckt die Lösung für die zweite Herausforderung, nämlich einen *größtmöglichen Durchsatz* beim Persistieren der Ereignis-Objekte zu schaffen. Um dieses Ziel zu erreichen, darf nicht jedes Objekt einzeln in der Datenbank gespeichert werden, sondern dies muss gebündelt in *batches* geschehen. Hibernate bietet die Möglichkeit, Speicheroperationen bis zu einer gewissen Größe vorzuhalten und zu einem bestimmten Zeitpunkt gemeinsam abzuarbeiten. Dies spart den Overhead eines etwaigen Verbindungsaufbaus, der Vorbereitung der entsprechenden SQL-Operation sowie der Transaktionsverwaltung.

Wie in Code 6.14 auf der nächsten Seite zu sehen ist, werden in der Methode `attemptPersisting()`, sofern mindestens `min` Elemente in der Warteschlange sind, `max` davon in eine temporäre Liste verschoben und dann persistiert. Entsprechen `min` und `max` beide der Größe der in Hibernate eingestellten *batches*, so wird diese Anzahl an Ereignis-Objekten auf einen Schlag gespeichert.

Sind nicht so viele Ereignisse in der Warteschlange, so tritt ein anderer Mechanismus in Kraft: ein Thread (`cleanupThread`) wird beauftragt, nach einer gewissen Zeit (`CLEANUP_DELAY`) erneut einen Versuch zum Persistieren zu starten, diesmal aber mit der minimalen Zahl an Objekten auf 0 und ohne erneut diesen Thread zu starten. Dadurch wird sichergestellt, dass selbst wenn nicht genügend Ereignisse eintreffen, um ein komplettes Bündel zu persistieren, die vorhandenen Objekte dennoch nach einer gewissen Zeit gespeichert werden. Gestartet

wird der Thread aber nur, wenn nicht schon ein derartiger läuft, da dieser ja ohnehin alle Ereignisse persistiert.

```
private void attemptPersisting(int min, int max, boolean reschedule) {

   Collection persistable = null;
   try {
      if (eventQueue.size() >= min) {
         persistable = new ArrayList<AuditEvent>();
         eventQueue.drainTo(persistable, max);
      } else if (reschedule && (cleanupThread == null ||
            cleanupThread.isDone() || cleanupThread.isCancelled())) {

         cleanupThread = cleanupScheduler.schedule(new Runnable() {
                        public void run() {
                           try {
                              queueLock.lockInterruptibly();
                              attemptPersisting(0, BATCH_SIZE, false);
                           } catch (InterruptedException e) {
                              e.printStackTrace();
                           }
                        }
                     }, CLEANUP_DELAY, TimeUnit.SECONDS);
      }

      if (persistable != null) {
         AuditHibernateUtil.persistEvents(persistable);
      }

   } catch (Exception e){
      e.printStackTrace();
   } finally {
      queueLock.unlock();
   }
}
```

**Code 6.14:** Persistieren der Ereignis-Objekte in der Warteschlange

## 6.6 Profiling und Optimierung

Bereits in Version 2 aber auch nach der Weiterentwicklung von Prospector benötigte das System rund 30 bis 40 Sekunden, um eine Suchanfrage abzuarbeiten. Verglichen mit herkömmlichen Suchmaschinen, die ihre Ergebnisse teils sogar in Sekundenbruchteilen liefern, ist dies entschieden zu lange und hat auch für Probleme bei der Evaluierung in den Niederlanden gesorgt. Wie bereits in Abschnitt 3.4 erwähnt, wird auch in der Literatur eine gute Verarbeitungsgeschwindigkeit als die größte Herausforderungen für personalisierte Suchmaschinen angesehen. Um die Leistung von Prospector zu steigern sollte daher zuerst herausgefunden

werden, wo die Flaschenhälse liegen, um das System an diesen Stellen anschließend zu optimieren.

Auf der Suche nach Teilen von Prospector, in denen es bei Suchanfragen zu Geschwindigkeitsproblemen kommen kann, wurden vier Teilbereiche identifiziert, die mit größeren Mengen an Daten arbeiten:

- *Suchanfrage*: unglücklicherweise kann die Geschwindigkeit der Suchmaschine, welche die originalen Ergebnisse liefert, nicht beeinflusst werden. Einzig die Optimierung der Suchroutinen selbst könnte hier helfen.

- *Klassifizierung*: für jedes Ergebnis muss bei der Klassifizierung der gesamte Datenbestand des ODP durchsucht werden. Hier wären Optimierungen sowohl in der Datenbank als auch beim Abfrage-Code möglich.

- *Relevanz-Berechnung*: für jede Klassifizierung müssen die Modelle durchsucht und entsprechende Relevanzen berechnet werden. Da Prospector bisher noch nicht optimiert wurde, kann neben der Datenbank zum Speichern der Modelle auch dieser Code beschleunigt werden.

- *Darstellung*: die stark interaktive Web-Oberfläche verwendet viel JavaScript-Code (z. B. zum Umreihen), der bislang nur nach funktionalen Gesichtspunkten entwickelt wurde. Da häufig verwendete Funktionen aber keine größeren Probleme bei der Geschwindigkeit zeigten, wurde dieser Bereich als nachrangig bewertet.

Die Bereiche Klassifizierung und Relevanz-Berechnung erschienen als am ehesten erfolgversprechend, Prospector bei Suchanfragen zu beschleunigen. Um festzustellen, wo der größte Handlungsbedarf besteht, wurde versucht, auf einfache Weise festzustellen, ob die Hauptlast bei der Datenbank oder im Code liegt. Zum Einsatz kamen zwei Werkzeuge:

- *UNIX-Kommando* `top`: ermöglicht das periodische Ausgeben der CPU-Last für einzelne Prozesse. Mit dem Befehl `top -l 200 -F -o cpu -n 5 -R > profile.txt` werden für 200 Sekunden jede Sekunde Informationen zu den Top-5 Prozessen gereiht nach CPU-Last in die Datei `profile.txt` geschrieben. Leider lässt sich das Intervall zwischen den Messungen nicht verkleinern. Das Ergebnis zeigte nicht klar, ob die Datenbank oder Tomcat die CPU stärker und länger belastete.

- *Mac-Programm Instruments*: mit diesem Programm ist es unter anderem möglich, diverse Kennzahlen von Prozessen auch im Lauf zu überwachen und graphisch darzustellen. Ein Beispiel für eine Protokoll von Suchabfragen in Prospector und den entsprechenden Systembelastungen ist in Abbildung 6.14 auf der nächsten Seite zu sehen. Eine klarer Unterschied zwischen MySQL und Tomcat ist auch hier nicht zu sehen. Erkennbar ist lediglich, dass bei den Datenbankabfragen auch die Lesezugriffe auf die Festplatte stark ansteigen, was einen Hinweis auf Optimierungspotenziale gibt.

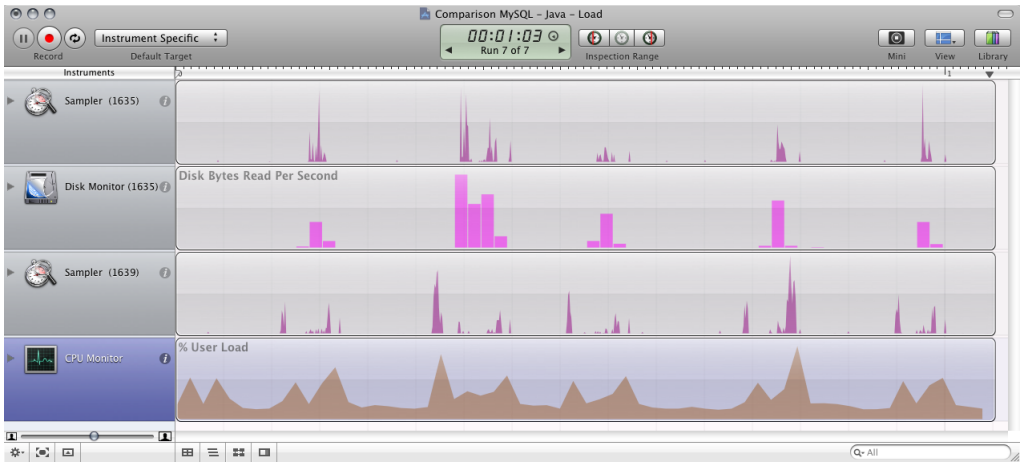


**Abbildung 6.14:** Profiling mit Apple Instruments: MySQL (CPU und gelesene Daten), Tomcat (CPU) und systemweit (CPU) bei fünf Suchanfragen an Prospector

Aufgrund dieser geringen Unterschiede wurde entschieden, sowohl bei der Datenbank als auch beim Code von Prospector nach Optimierungsmöglichkeiten zu suchen. Auch in der Web-Anwendung bzw. der von ihr genützten Carrot$^2$-Verarbeitungskette gab es sehr einfache Möglichkeiten zur Optimierung, die zuvor noch in Angriff genommen wurden.

### 6.6.1 Web-Anwendung

Wie bereits weiter oben beschrieben wird für die Anzeige der Ergebnisseite einer Suchanfrage die *Verarbeitungskette zweimal durchlaufen*, einmal bei der Anfrage für das Dokument-IFRAME und einmal für das Cluster-IFRAME. Um nicht bei beiden Malen die Suchmaschine befragen zu müssen, werden die Ergebnisse bereits in einem Cache zwischengespeichert. Eine Optimierungsmöglichkeit ergibt sich aber durch das Überspringen von Komponenten in der Verarbeitungskette:

- *Dokument-Anfrage*: die Clustering-Komponente `ODPClustererLocalFilterComponent` am Ende der Kette wird nicht benötigt.
- *Cluster-Anfrage*: weggelassen werden können der Prospector-Algorithmus (`ProspectorLocalFilterComponent`), die Komponente zur „data fusion“ (`DataFuserLocalFilterComponent`) und jene zum Sortieren (`ScoreSorterLocalFilterComponent`).

```
final HashMap props = new HashMap();
props.put(LocalFilterComponent.PARAM_REQUEST_TYPE,
          LocalFilterComponent.DOCUMENT_REQUEST);
```

**Code 6.15:** Festlegung des Anfragetyps durch das `QueryProcessorServlet`

Zu diesem Zweck müssen nicht zwei unabhängige Verarbeitungsketten geschaffen werden, denn man kann die Ausführung von Komponenten parametrisieren. Im `QueryProcessorServlet` wird zu diesem Zweck, wie in Code 6.15 auf der vorherigen Seite dargestellt, der Typ der Anfrage in den Properties gesetzt. Mit dieser Information können in den jeweiligen Komponenten Teile ähnlich wie in Code 6.16 nur dann ausgeführt werden, wenn der Typ mit einem vorgegebenen übereinstimmt.

```
if (LocalFilterComponent.DOCUMENT_REQUEST.equals(
      requestParams.get(LocalFilterComponent.PARAM_REQUEST_TYPE))) {
   ...
}
```

**Code 6.16:** Abfragen des Anfragetyps in einer Komponente der Verarbeitungskette

### jMeter

Um die Auswirkungen dieser und weiterer Optimierungen überprüfen zu können, wurde das Open Source-Programm Apache jMeter[8] eingesetzt. Dieses ist speziell dafür ausgerichtet, *Belastungstests und Leistungsmessungen* insbesondere an Web-Anwendungen aber auch an Web-Services, Datenbanken, Server-Diensten und beliebigen sonstigen Programmkomponenten (vorzugsweise in Java) vorzunehmen. jMeter verfügt über eine graphische Oberfläche zum Zusammenstellen der Tests, Überwachen von Testläufen und graphischen Auswerten der Ergebnisse. Es ist in Java programmiert und kann durch Plugins und selbst geschriebene Testtreiber erweitert werden.

Das grundlegende *Funktionsprinzip* von jMeter ist folgendes: man erstellt einen Testplan, der angibt, wie viele Threads mit welchen Parametern auf welche Teile des Systems zugreifen. Die Dauer der einzelnen Zugriffe wird gemessen und kann anschließend in Auswertungsmodulen nach bestimmten Gesichtspunkten analysiert werden. Der Testplan für Prospector umfasste folgende Module:

- *HTTP Cache Manager*: dieser simuliert den Cache, dank dem ein Browser Dateien, die sich nicht geändert haben, nicht bei jeder Anfrage vom Server laden muss.
- *HTTP Cookie Manager*: speichert wie in einem Browser Cookies und gibt diese bei jeder nachfolgenden Testanfrage an die Web-Anwendung zurück. Hier wurde ein Cookie voreingestellt, sodass ein eingeloggter Benutzer simuliert werden konnte.
- *CSV-Quelle*: ermöglicht es, aus einer Comma Separated Values (CSV)-Datei Werte auszulesen und in Variablen zu verwenden. Im Falle von Prospector wurden Suchbegriffe aus einer Datei geladen, in einer Variable gespeichert und zum Konstruieren der zu ladenden URL in den HTTP Request-Modulen verwendet. Die Suchbegriffe stammten aus einer Liste einfacher Englischer Wörter[9].

[8] `http://jakarta.apache.org/jmeter/`

[9] `http://www.members.optusnet.com.au/~charles57/Creative/Techniques/random_words.htm`

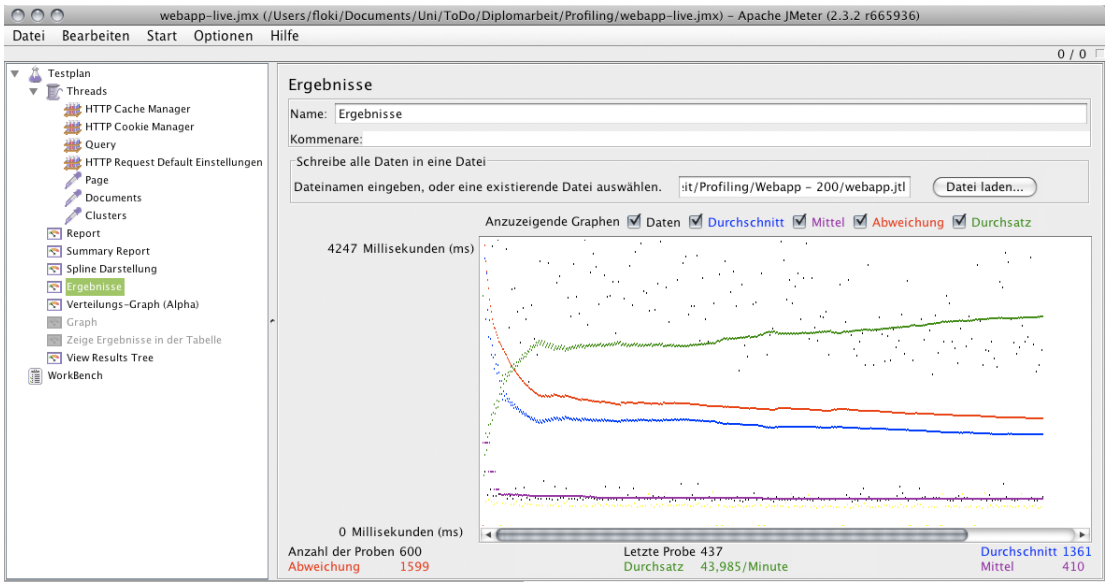


**Abbildung 6.15:** Beispiel für die Analyse eines Testlaufs mit jMeter

- *HTTP Request Standardeinstellungen*: hier können Standardwerte angegeben werden, die für alle HTTP Request-Module gelten. Verwendet wurde dies, um Rechnernamen, Port, Pfad und einige Parameter (Suchmaschine, Algorithmus, Anzahl Suchergebnisse, Suchanfrage) festzulegen, die für die URLs zum Laden der Ergebnisseite und der beiden darin enthaltenen IFRAMEs (Dokumente und Cluster) ident sind.

- *HTTP Request*: dieses Modul konstruiert anhand der Standardeinstellungen und eigenen Parametern die URL und führt die Anfrage durch. Einstellbar sind unter anderem auch die Art der Anfrage (GET, POST, PUT, ...), ob Weiterleitungen aufgelöst, und ob die im Ergebnis refrenzierten Bilder und Java Applets ebenfalls heruntergeladen werden sollen. Für Prospector wurden drei derartige Module konfiguriert, je eines für die Ergebnisseite, das Dokument-IFRAME und das Cluster-IFRAME.

- *Thread-Gruppe*: fasst die oben genannten Module zusammen und legt fest, wie viele Threads in welchen Abständen gestartet werden, um mit den gegebenen Einstellungen entsprechend URLs zu konstruieren und auf diese zuzugreifen, und wie oft sie Anfragen stellen sollen.

Im Anschluss an einen Testlauf können die Leistungsdaten nach diversen Gesichtspunkten analysiert werden. Interessant waren hierbei vor allem die durchschnittliche und die maximale Antwortzeit, der sich daraus ergebende Durchsatz und die Verteilung der Antwortzeiten. Abbildung 6.15 zeigt den Verlauf einiger dieser Kennzahlen während eines Teslaufs. Zu erkennen ist, dass sich die Werte nach einer anfänglichen Einlaufphase allmählich stabilisieren. Hier wäre auch erkennbar, wenn ein System ab einer gewissen Zahl gleichzeitiger Anfragen überlastet ist (*thrashing*) und der Durchsatz stark absinkt.

### 6.6.2 Code-Ebene

Nachdem, wie zu Beginn dieses Abschnitts bereits erwähnt, sowohl bei der Datenbank als auch beim darauf zugreifenden Code Optimierungspotenziale bestehen, wurde nach den Parametrisierungen in der Verarbeitungskette (siehe Unterabschnitt 6.6.1) ein *Quellcode-Profiling* durchgeführt. Da die bisherige Entwicklung in der Eclipse-Umgebung gemacht wurde, fiel die Wahl zuerst auf dessen Test & Performance Tools Platform Project (TPTP). Ein Hauptbestandteil dieses Plugins für Eclipse war jedoch für Mac OS X nicht implementiert. Die Suche nach einem alternativen Profiler förderte etliche kostenpflichtige Lösungen und viele unausgereifte freie Lösungen zu Tage. Eine mächtiger und integrierter Profiler fand sich schließlich in der kostenlosen *Netbeans*[10] Entwicklungsoberfläche.

Mit diesem Profiler ist es möglich, sowohl die *CPU-Belastung* einzelner Teile (z. B. Methoden) als auch den *Speicherbedarf* einer Anwendung während der Laufzeit zu überwachen und analysieren. Die Laufzeitanalyse kann sowohl für die gesamte Applikation als auch feingranular für bestimmte Teile aktiviert werden. Es wird dann für jede Methode die Zeit, die sie, abzüglich der Zeit der von ihre aufgerufenen Methoden, gelaufen ist, angezeigt und wie oft sie insgesamt aufgerufen wurde. So lassen sich jene Teile einer Anwendung finden, die besonders aufwändig sind, und wo man am besten zu optimieren beginnt.

Abbildung 6.16 auf der nächsten Seite zeigt beispielhaft eine *Laufzeitanalyse* mit einigen kritischen Methoden in Prospector. Je nach Umfang der gespeicherten Modelle, Anzahl der Suchergebnisse und Status von Caches gingen in den Tests bis zu 60% der Laufzeit auf das Konto der Methode `EntityManager.queryTopicsForURL(String)`. Dies verwunderte nicht, wird doch mit ihr versucht, für die URL eines jeden Ergebnisses in den rund 4,5 Millionen im ODP-Datenbestand verzeichneten Links eine Übereinstimmung und somit eine Klassifizierung in einem oder mehreren Themen zu finden. Das Ziel musste daher sein, unabhängig von Optimierungen in der Datenbank die Anzahl dieser Anfragen zu minimieren.

```
Query topicQuery = session.createQuery("select topic "
                     + "from Topic as topic, Link as link "
                     + "where link.URL = :url "
                     + "and link.topicCategoryId = topic.categoryId")
                     .setCacheable(true);
```

**Code 6.17:** Abfrage zum Klassifizieren einer URL in Themen des ODP

Die Persistenzschicht Hibernate, die neben der Verwaltung der Modelle auch für den Zugriff auf die ODP-Daten verwendet wird, bietet einige Möglichkeiten zur Geschwindigkeitssteigerung. Da die Abfrage selbst (siehe Code 6.17) nur aus einem einfach Join besteht und nicht weiter optimiert werden kann, und auch sonst keine Verbesserungsmöglichkeiten im Code ersichtlich waren, bot sich die Verwendung der *Caching-Funktionen in Hibernate* an. Hierbei wird unterschieden zwischen dem *second-level cache* und dem *query cache*. Ersterer speichert

[10] `http://www.netbeans.org/`

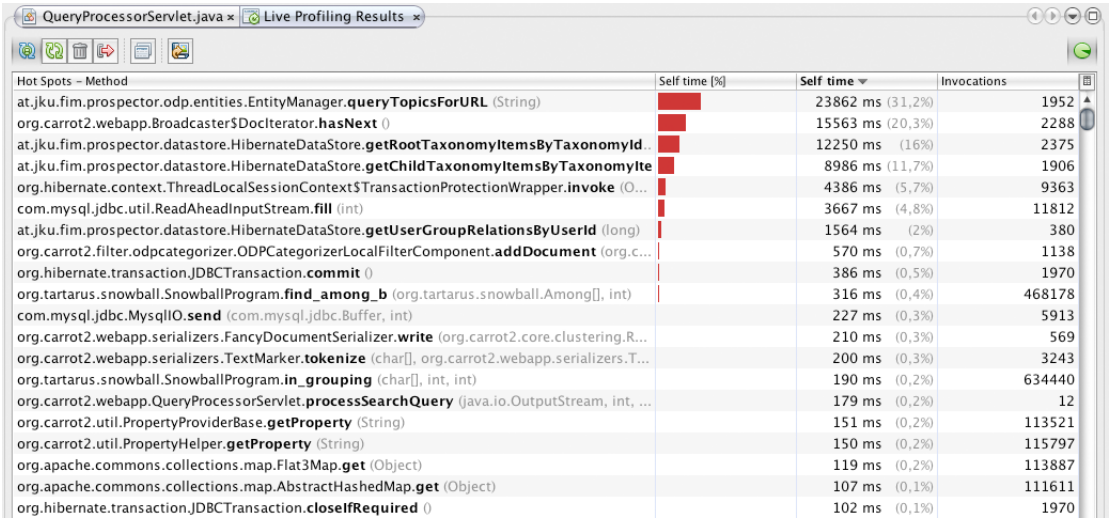


**Abbildung 6.16:** Laufzeitanalyse mit dem Profiler in Netbeans

die Daten von Objekten und kann über deren ID daraus ohne eine Datenbankabfrage ein entsprechendes Objekt erzeugen. Zweiterer speichert für konkrete Abfragen (mit vollständigen Parametern) die IDs der Objekte im Ergebnis.

Um die zwei Caches zu aktivieren, bedurfte es einiger Anpassungen. Zuerst musste die Abfrage bei ihrer Definition mit `setCacheable(true)` dafür freigegeben werden (siehe Code 6.17 auf der vorherigen Seite). Weitere Einstellungen wurden in der der Hibernate-Konfigurationsdatei `hibernate.cfg.xml` vorgenommen (siehe Code 6.18): EhCache[11] als Cache-Implementierung, die Position deren Konfigurationsdatei, dass der *query cache* aktiviert wird und Statistiken zur Cache-Nutzung gesammelt werden. Da bei den Abfragen der ODP-Daten ohnehin nur gelesen wird, wurde für die beiden Klassen `Link` und `Topic` der Nur-Lese-Modus aktiviert.

```
<property name="cache.provider_class">org.hibernate.cache.
    EhCacheProvider</property>
<property name="cache.provider_configuration_file_resource_path">at/jku
    /fim/prospector/ontology/odp/ehcache.xml</property>
<property name="cache.use_query_cache">true</property>
<property name="generate_statistics">true</property>

<class-cache class="at.jku.fim.prospector.ontology.odp.dataobject.Link"
     usage="read-only" />
<class-cache class="at.jku.fim.prospector.ontology.odp.dataobject.Topic
    " usage="read-only" />
```

**Code 6.18:** Cache-Einstellungen in der Hibernate-Konfigurationsdatei `hibernate.cfg.xml`

Für *EhCache* waren für jede Klasse, deren Objekte im *second-level cache* gespeichert werden sollten, noch weitere konkrete Einstellungen nötig. Diese wurden in der zuvor referenzierten

[11] `http://ehcache.sourceforge.net/`

Datei `ehcache.xml` (siehe Code 6.19) gesetzt: wie viele Objekte maximal im Speicher gehalten werden, ob diese nie durch einen Timeout enfernt werden, ob „überschüssige“ Objekte auf die Festplatte geschrieben werden, von dort auch nach einem Neustart eingelesen werden, wie lange (in Sekunden) ein Objekt insgesamt im Cache verbleibt, wie lange ohne Zugriff, und nach welcher Methode bei vollem Cache alte Objekte entfernt werden.

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="ehcache.xsd">

   <cache name="at.jku.fim.prospector.ontology.odp.dataobject.Topic"
      maxElementsInMemory="1000"
      eternal="false"
      overflowToDisk="false"
      diskPersistent="false"
      timeToLiveSeconds="1800"
      timeToIdleSeconds="1800"
      memoryStoreEvictionPolicy="LRU" />

      ...
</ehcache>
```

**Code 6.19:** Feingranulare Cache-Einstellungen für EhCache in `ehcache.xml`

Um die Nutzung des Caches zu überwachen, ermögliche Hibernate das *Abfragen von Statistiken*. In Code 6.20 sind beispielhaft Abfragen für den *query cache* angeführt. Für den *second-level cache* existieren äquivalente Methoden.

```
org.hibernate.stat.Statistics stats = ProspectorHibernateUtil.
   getSessionFactory().getStatistics();

putCount  = stats.getQueryCachePutCount();
hitCount  = stats.getQueryCacheHitCount();
missCount = stats.getQueryCacheMissCount();
hitRatio = hitCount / (hitCount + missCount);
```

**Code 6.20:** Abfrage von Statistiken zur Nutzung des *query cache* in Hibernate

Die soeben beschriebenen Einstellungen beziehen sich auf die Abfrage in der vom Profiler als *hot spot* identifizierten Methode `EntityManager.queryTopicsForURL(String)`. Auch für die Objekte und Abfragen zweier anderer Methoden, die einen beträchtlichen Teil der Laufzeit in Anspruch nahmen, wurde das Caching aktiviert: `DataStore.getRootTaxonomyItemsByTaxonomyId()`, welche die obersten Knoten eines Modells ermittelt und bei einer Suchanfrage mehrere tausende mal aufgerufen wird, obwohl bei 13 Gruppen-, einem Benutzermodell und 17 Themen oberster Ebene maximal 238 derartige Knoten existieren können, und `DataStore.getChildTaxonomyItemsByTaxonomyItem()`, die für einen Knoten im Modell dessen Kinder abfragt. Für letztere Methode gibt es bei der Anzahl der Abfragen keine wirkliche obere Grenze, aber da sich diese Daten im Verhältnis zur Häufigkeit ihrer Verwendung nur selten

| Bereich | Cache-Typ | Eingänge | Treffer | Fehlschläge | Trefferrate |
|---|---|---|---|---|---|
| Modelle | query | 10.293 | 116.965 | 10.293 | 91,91% |
| | second level | 145 | 297.182 | 14 | 99,99% |
| ODP | query | 24.238 | 44.011 | 24.238 | 64,49% |
| | second level | 8.924 | 13.222 | 0 | 100% |

**Tabelle 6.1:** Cache-Statistiken nach 200 Suchanfragen

ändern (z. B. beim Bewerten) erschien der Einsatz eines Caches hier optimal. Wichtig ist in diesen beiden Fällen, dass der Cache im Lese-Schreib-Modus betrieben wird, um Änderungen der Daten, beispielsweise durch Bewertungen des Benutzers, erkennen zu können.

Mit den in Code 6.20 auf der vorherigen Seite ermittelten Werten konnte die *Wirkung der Maßnahmen* klar nachvollzogen werden. Bei einem Testlauf mit insgesamt 200 Suchanfragen ergaben sich die Zahlen in Tabelle 6.1. Da die Modelle im Verhältnis zur Umfang des Caches nicht sehr groß waren und sich auch nicht oder nur minimal änderten, ergaben sich diese guten Trefferraten. Die Trefferrate beim *query cache* für ODP-Anfragen lässt sich folgendermaßen erklären: da die Verarbeitungskette zwei mal pro Suchanfrage durchlaufen wird, ergeben sich zumindest 50%. Die restlichen 14,49% lassen sich durch Ergebnisse erklären, die bei mehr als einer Suchanfrage vorkamen. Der *second level cache* wird mit den IDs jener Objekte abgefragt, die als Ergebnis von Anfragen bekannt sind. In diesem Fall war dieser immer schon nach dem ersten Durchlauf der Verarbeitungskette gefüllt, und es kam nie zu Fehlschlägen.

Insgesamt gesehen sorgte die Aktivierung der Caches für einen *merklichen Geschwindigkeitsgewinn*. Bedenkt man die relativ unkomplizierte Konfiguration, so zeigt sich einer der großen Vorteile des Einsatzes einer so flexiblen und mächtigen Persistenzschicht wie Hibernate. Den Bedenken, dass mehr Code und eine generischere Ausrichtung einen unnötigen Overhead bedeuten, konnte zumindest in diesem Fall mit positiven Effekten begegnet werden.

### 6.6.3 ODP-Klassifikation

Durch Caching konnten viele Datenbankzugriffe bei der Verwaltung der Modelle gespart werden, und auch die Abfragen der ODP-Daten profitierten vom Cache. Relativ gesehen benötigten letztere aber noch immer 70 bis 80% der Zeit bei einer Suchanfrage. Aus diesem Grund wurde auch die Datenbank selbst untersucht, insbesondere in den Bereichen:

- *Datenspeicherung*: welche Speicherstrukturen sind optimal?
- *Indizierung*: wie werden die Daten am besten indiziert?
- *Abfrage*: kann die Abfrage noch besser formuliert werden?
- *Caching*: wie können schon in der Datenbank Caches wirken?
- *Vorladen*: wie bekommt man beim Start möglichst viele Daten in den Hauptspeicher?

Bei dem verwendeten Datenbankmanagementsystem MySQL gibt es die Möglichkeit, zwischen mehrere *Arten physischer Datenspeicherung* zu wählen. Zwei der dafür am häufigsten verwendeten *engines* sind MyISAM und InnoDB. Erstere unterstützt keine Transaktionen, verspricht hohe Leistung und wird standardmäßig verwendet. InnoDB ist transaktionsbasiert und soll in bestimmten Situationen schneller sein. Um zu testen, welche Methode in diesem Fall vorteilhafter ist, wurden unter identen Bedingungen jeweils 2361 URLs durch ein Testprogramm klassifiziert. Mit MyISAM lief der Test um 36% schneller.

Ebenso getestet wurde, mit welchem *Datentyp* die Spalte, welche die klassifizierten URLs in den ODP-Daten enthält, gespeichert werden soll. Zur Auswahl standen die Typen VARCHAR und TEXT, welche beide eine Feldlänge von maximal 65.535 Zeichen erlauben. Der schnellere Typ CHAR erlaubt nur 255 Zeichen und ist daher für URLs ungeeignet. TEXT wurde von Hibernate automatisch gewählt, benötigt aber mehr Speicherplatz und Zugriffe. Dennoch stellte sich im Test (siehe oben) heraus, dass bei der Tabelle mit der Spalte im Typ TEXT die Anfragen doppelt so schnell abgearbeitet wurden.

Als eine der wirkungsvollsten Methoden zur Beschleunigung der Abfragen erwies sich die *Indizierung von Spalten*. Die ID-Spalten, über die die Tabelle mit den URLs mit der Tabelle der Themen verknüpft sind (siehe Code 6.17 auf Seite 123), verfügten bereits über einen Index, der automatisch von Hibernate beim Anlegen der Tabelle erzeugt wurde. Da bei der Abfrage nach Datensätzen für eine gegebene URL gesucht wird, benötigt diese Spalte ebenfalls einen Index.

Damit dieser Index die Daten zwar gut aufteilt, aber nicht über die gesamte Länge der Spalte geht und dadurch unnötig groß wird, musste zuerst herausgefunden werden, wie stark sich die Inhalte in dieser Tabelle unterscheiden. Wäre der Index nur elf Zeichen lang, so könnte er nur schlecht zwischen den Millionen von URLs, von denen die meisten mit `http://www.` beginnen unterscheiden und wäre nutzlos. Mit der Abfrage in Code 6.21 konnte durch Variieren der Zahl $n$, die nun auf 30 steht, ermittelt werden, wie viele URLs in den ersten $n$ Zeichen unterschiedlich sind.

```
SELECT COUNT(DISTINCT url) AS distinct_rows,
       COUNT(DISTINCT(LEFT(url, 30))) AS prefix_distinct
FROM (SELECT * FROM links LIMIT 0, 1000000) l
```

**Code 6.21:** Bestimmung der Unterscheidungsstärke der ersten 30 Zeichen in der `url`-Spalte

Abbildung 6.17 auf der nächsten Seite zeigt, wie die Unterscheidungskraft des Index mit seiner Größe variiert. Die Entscheidung für seine Länge fiel auf 30 Zeichen. Bei dieser Länge ist der Index vom Speicherverbrauch her so groß wie die Daten selbst (ca. 230 MB), erlaubt aber trotzdem die Unterscheidung von 80% der URLs.

Zur Überprüfung, ob die Indizes auch verwendet werden, und ob es noch Optimierungspotenzial in der Abfrage gibt, wurde untersucht, welchen *Ausführungsplan* MySQL konkret verwendet. Durch Einfügen des Schlüsselworts `EXPLAIN` vor der SQL-Abfrage kann man sich

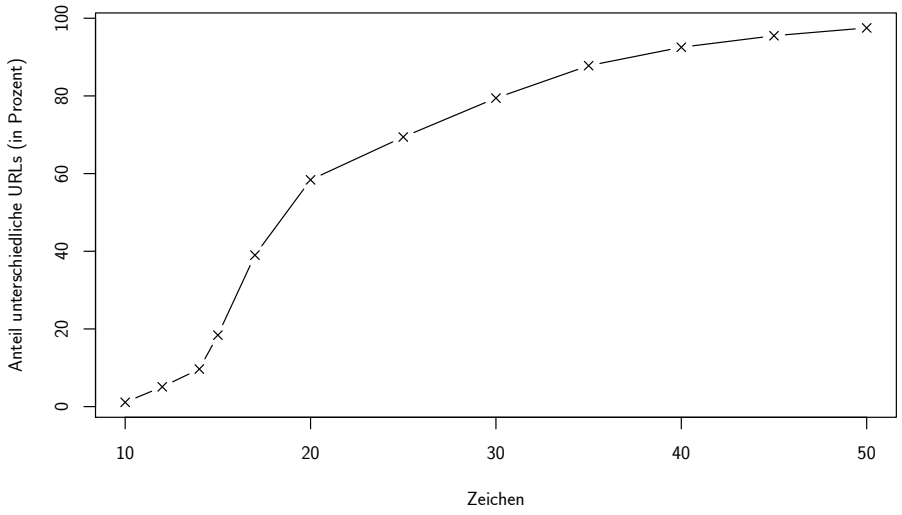


**Abbildung 6.17:** Unterscheidungskraft von Indizes verschiedener Länge

anzeigen lassen, wie diese abgearbeitet wird, welche Indizes zum Einsatz kommen, ob an einer Stelle unnötig viele Vergleiche gemacht werden und noch vieles mehr. Im Falle von Prospector ergab sich, dass die Abfrage bereits optimal gestaltet und alle Maßnahmen zur Beschleunigung durch Indizes bereits ausgeschöpft waren. Aus diesem Grund wurde im nächsten Schritt versucht, Caches einzusetzen.

Aufgrund der Größe des Index für die URLs, musste auch der *Cache für Indizes* angepasst werden. Dieser ist standardmäßig in MySQL auf 8 MB limitiert. Durch Setzen des Parameters `key_cache_size` in der Konfigurationsdatei auf 256 MB konnte er nun alle Indizes aufnehmen. Dies würde während der Abfragen geschehen, zur Steigerung der Leistung kann man aber Indizes vorab in den Cache laden. Mit dem Befehl in Code 6.22 werden die Indizes der beiden Tabellen `links` und `topics` von der Festplatte gelesen und in den Speicher geladen. Diese Maßnahme brachte eine erneute Einsparung an Laufzeit von knapp 40%.

```
LOAD INDEX INTO CACHE links, topics
```

**Code 6.22:** Indizes vorab in den Cache laden

Auch MySQL verfügt über einen *query cache*. Dieser sollte, nachdem in Hibernate bereits sehr viele Abfragen aus dem Cache beantwortet werden können, nicht mehr wirklich zum Tragen kommen. Er wurde mit folgenden Optionen in der Konfigurationsdatei aktiviert: `query_cache_type = 1` und `query-cache-size = 32M`. Tests mit dem Profiler ergaben, dass sich auch durch diese Maßnahme eine Beschleunigung ergab.

Nachdem bereits die Indizes vorab in den Speicher geladen waren, wurde versucht, dies auch mit den Datentabellen zu machen und somit aufwändige Festplattenzugriffe zu sparen. Bei einer Größe der ODP-Daten von rund 230 MB erschien dies machbar und im Verhältnis zum erhofften Geschwindigkeitsgewinn vertretbar. Das Laden erfolgt, wie in Code 6.23 dargestellt,

durch Erzeugen einer neuen Tabelle, die komplett im Speicher gehalten wird, und Füllen mit dem Inhalt der originalen Tabelle. Es stellte sich heraus, dass die Größenangaben der Tabellen der (vermutlich komprimierten) Größe auf der Festplatte entspricht und im Speicher um vieles überschritten werden. Eine Million Zeilen der Tabelle mit den URLs belegten 1,4 GB, und dies machte das Laden aller 4,5 Millionen Datensätze unmöglich.

```
CREATE TABLE links_mem ENGINE=MEMORY AS SELECT * FROM links
```

**Code 6.23:** Tabelle komplett in den Arbeitsspeicher laden

Umgesetzt wurde schließlich ein anderer Ansatz, der nach dem Start die Tabellen einmal vollständig einliest, wodurch sie vom Betriebssystem zumindest für eine gewisse Zeit in einem Cache verbleiben. Möglich ist diese durch Code 6.24. Mit ihm werden Blackhole-Tabellen mit dem gleichen Aufbau wie die originalen angelegt, allerdings mit der Eigenschaft, dass alles, was man in sie speichert, verworfen wird. Anschließend werden die Tabellen einmal vollständig ausgelesen und in diese Blackhole-Tabellen geschrieben.

```
CREATE TABLE links_bh LIKE links;
ALTER TABLE links_bh ENGINE = BLACKHOLE;
INSERT INTO links_bh SELECT * FROM links ORDER BY linkid;

CREATE TABLE topics_bh LIKE topics;
ALTER TABLE topics_bh ENGINE = BLACKHOLE;
INSERT INTO topics_bh SELECT * FROM topics ORDER BY categoryid;
```

**Code 6.24:** Komplettes Einlesen der Tabellen, um sie in den Betriebssystems-Cache zu laden

Durch diese Maßnahme konnten noch einmal 60% der Laufzeit eingespart werden. Allerdings wurde diese Messung direkt nach dem Einlesen durchgeführt. Wie lange dieser positive Effekt anhält, hängt davon ab, ob die Caches des Betriebssystems durch andere Programmaktivitäten wieder geleert werden.

Durch eine Kombination mehrerer Methoden zur Leistungssteigerung bei Datenbankanfragen konnte die Geschwindigkeit der ODP-Klassifizierung enorm gestiegert werden. In der Endversion von Prospector wurden somit die früheren Antwortzeiten auf Suchanfragen von 30 bis 40 Sekunden bei 50 Ergebnissen auf drei bis vier Sekunden bei 100 Ergebnissen reduziert.

# Kapitel 7

# Evaluierung

Wie in der Aufgabenstellung gefordert wurde die weiterentwickelte Version von Prospector einer Evaluierung unterzogen. Interessant hierbei war vor allem, ob es Verbesserungen gegenüber der Evaluierung der zweiten Version gegeben hat. Im Gegensatz zu dieser fand die dritte Studie nicht in einem Labor statt, sondern beobachtete Teilnehmer über einen *längeren Zeitraum* beim *realen Einsatz* des Systems. Prospector wurde von den Testpersonen also wie jede beliebige andere Internet-Suchmaschine im Alltag verwendet.

Die Evaluierung versuchte, *Antworten in drei Bereichen* zu finden: Wie gut ist die Modellierung der Benutzerinteressen? Schafft es Prospector besser als die bisher verwendete Suchmaschine, relevante Ergebnisse ganz oben in der Ergebnisliste zu liefern? Wie gut ist die Gebrauchstauglichkeit von Prospector aus Sicht der Benutzer? Für die letzte Frage wurden die bereits in Abschnitt 2.3 genannten *sechs Kriterien nach Jameson* [Jam03] geprüft: Berechenbarkeit, Verständlichkeit, Steuerbarkeit, Unaufdringlichkeit, Privatheit und Erlebnisreichtum.

Ergänzt werden diese Kriterien durch die von Anand und Mobasher [AM05] genannten *sieben Schlüsseldimensionen der Evaluierung personalisierter Systeme*. Für das Einsatzgebiet der personalisierten Suche lassen sich diese wie folgt konkretisieren:

- *Zufriedenheit* (user satisfaction): Ist der Benutzer mit den Ergebnissen und deren Reihung zufrieden und findet die gesuchten Dokumente?
- *Genauigkeit* (accuracy): Wie klar trifft das Ergebnis die Suchintention? Dies entspricht dem im Information Retrieval (IR) oft verwendeten Begriff der *precision*.
- *Abdeckung* (coverage): Wie umfassend sind die Ergebnisse im Bezug auf die gesamte Zahl relevanter Dokumente? Entspricht dem IR-Begriff des *recall*.
- *Nützlichkeit* (utility): Wie groß sind die wirklichen Verwendungsmöglichkeiten für das System? Dies ist unabhängig von der Zufriedenheit zu sehen.
- *Erklärbarkeit* (explainability): Kann der Benutzer aufgrund der Aktionen des Systems ein gültiges mentales Modell von dessen Funktion bilden?

- *Robustheit* (robustness): Ist das System unanfällig gegen kleine Fehler von Seiten des Benutzers oder absichtliche Attacken von Angreifern? Können die Benutzermodelle oder die Suchergebnisse durch unerwünschte Aktionen negativ beeinflusst werden?
- *Leistung und Skalierbarkeit* (performance and scalability): Wie schnell arbeitet das System, und wie verhält es sich bei einem Ansteigen des Datenvolumens bzw. der Zahl der Benutzer.

Zu Beginn dieses Kapitels werden die zweite Evaluierung und ihre Ergebnisse näher beschrieben. Anschließend werden die Gestaltung und die Ergebnisse der aktuellen Evaluierung erläutert. Besonders eingegangen wird hierbei auf den Ablauf, die verwendeten Evaluierungsinstrumente und die Teilnehmer. Bei den Ergebnissen werden neben demographischen Informationen auch allgemein aufgetretene Probleme sowie die konkreten Ergebnisse entsprechend den meisten der oben genannten Kriterien vorgestellt.

## 7.1 Vorangegangene Evaluierung

Im Oktober 2006 fand die erste größere Evaluierung von Prospector statt. Sie wurde am Department of Technical and Professional Communication der University Twente in Enschede in den Niederlanden durchgeführt. Viele der beschriebenen Entwicklungen an Prospector basieren auf den Erkenntnissen dieser Studie. Nicht zuletzt aufgrund der professionellen Durchführung in einem Usability-Labor ließen sich manche Ergebnisse direkt umsetzen. Im Folgenden werden die Gestaltung der Evaluierung und ihre Ergebnisse kurz beschrieben. Details finden sich in einem wissenschaftlichen Artikel, der im Moment gerade für die Publikation vorbereitet wird [vVPvdG08].

### 7.1.1 Gestaltung

Die formative Studie hatte als Fokus den Benutzer und diente der Erforschung des Erlebens von Adaptivität, und um die Eignung verschiedener Evaluierungsmethoden beim Testen von adaptiven Systemen zu untersuchen. 32 Studierende der Sozialwissenschaften mussten diverse Testaufgaben durchführen und wurden mit folgenden Evaluierungsmethoden dabei untersucht:

- *Thinking-aloud Protokolle*: mit diesen können Probleme mit der Gebrauchstauglichkeit (usability) entdeckt und der Wert der Personalisierung eingeschätzt werden.
- *Fragebögen*: sammeln Meinungen, Vorlieben und Einstellungen.
- *Strukturierte Interviews*: haben ähnliche Ergebnisse wie Fragebögen.
- *Logdateien*: mit ihnen kann das Benutzerverhalten im System nachvollzogen werden.

Die *Aufgaben* beschäftigten sich mit der Planung einer Städtereise, und die Teilnehmer mussten zu diesem Zweck Jugendherbergen und Museen für moderne Kunst in fünf europäischen Städten suchen. Nach anfänglichen demographischen Fragen mussten sie eine Jugendherberge und ein Hotel in der ersten Stadt mit maximal drei Suchbegriffen mittels Google suchen und nur mit der Ergebnisliste jenes Ergebnis auswählen, das sie als erstes betrachten würden. Die Relevanz der ersten fünf Ergebnisse war auf einer siebenstufigen Skala zu bewerten. Diese Relevanzbewertung wurde nach dem Betrachten der Suchergebnisse wiederholt.

Für die nächsten zwei Städte mussten die Teilnehmer wiederum jeweils nach einer Jugendherberge und einem Museum suchen, die ihrem Anschein nach irrelevanten Ergebnisse in der Liste mit der „Unsuitable"-Schalffläche entfernen, die bevorzugten Ergebnisse öffnen und bewerten. Im Anschluss wurde mit einer siebenstufigen Skala die Zufriedenheit mit Prospector in zwei Aspekten gemessen. Die Benutzer durften nun das Benutzermodell betrachten und modifizieren. Danach folgten die Suchen für die vierte Stadt.

Für Stadt fünf mussten die Benutzer in Prospector die selben Suchanfragen (mit dem Städtenamen allerdings verändert) wie bei der ersten Stadt verwenden. Anschließend wurden wieder Relevanz und Zufriedenheit gemessen. Zum Abschluss folgte ein direkter Vergleich mit Google. In der letzten Ergebnisliste mussten die Teilnehmer im Vergleich mit der originalen Google-Reihenfolge angeben, was ihnen auffiel, und ob sie eine der beiden Reihungen bevorzugten und warum.

### 7.1.2 Ergebnisse

Neben den bereits in 5.2.2 besprochenen Erkenntnissen ergaben sich Ergebnisse in zwei Bereichen: für quantitative Messungen und für qualitative Messungen. Bei ersteren zeigten sich zunächst die Auswirkungen der Umreihung, gemessen am *Rang des bevorzugten Ergebnisses*. Bei Prospector (2.97 bei Jugendherbergen und 3.16 bei Museen) lag dieses jeweils um beinahe einen vollen Rang vor dem bei Google (3.80 bzw. 4.06), doch der Unterschied war statistisch nicht signifikant.

Zu Messung der *empfundenen Relevanz der Suchergebnisse* waren ja die ersten fünf Ergebnisse bei je zwei Google und Prospector Suchen auf einer siebenstufigen Skala vor und nach Betrachten derselben gemessen worden. Der Durchschnitt dieser Relevanzbewertungen war bei Prospector immer niedriger, signifikant allerdings nur bei Suchen nach Jugendherbergen vor dem Betrachten (5.52 zu 5.13) und bei Museen nach dem Betrachten (3.69 zu 2.78).

Die *Zufriedenheit der Benutzer* mit dem System sank zwischen den zwei Messungen von durchschnittlich 4.52 auf einer siebenstufigen Skala auf 4.23. Dieser Unterschied ist allerdings nicht signifikant. Die Hypothese, dass mit steigender Personalisierung die Zufriedenheit steigt, musste verworfen werden.

Die *qualitativen Ergebnisse* gaben einige Einblicke in mögliche Gründe dafür, warum die quantiativen Messungen nicht für Prospector sprachen, und lieferten auch Anregungen für dessen Weiterentwicklung. Ein erster Grund für die niedrigere Relevanz der Ergebnisse von

Prospector betrifft das Themengebiet „news“. Viele Teilnehmer zeigten daran Interesse, und so kamen zahlreiche (teilweise veraltete) Nachrichtenartikel in die obersten Ränge. Dieses Problem trat auch bei „shopping“ mit e-Commerce-Seiten auf. Für Unzufriedenheit sorgte zudem, wenn viele sehr ähnliche Seiten eines Themas an oberster Stelle standen oder erwartete wichtige Ergebnisse nicht weit genug oben auftauchten.

Im Bereich der *Benutzeroberfläche* und der *Interaktion* kam es zu folgenden Problemen: Viele Teilnehmer verstanden die Namen der ODP-Themengebiete nicht, sowohl im Profil als auch im Benutzermodell. Sie fanden sie zu vage und mehrdeutig. Für Schwierigkeiten sorgte auch die hierarchischen Struktur des Benutzermodells, die Beziehung von Themen zu ihren Unterthemen, und dass man diese durch Klicken ausklappen kann. Positiv wurde hingegen hervorgehoben, unpassende Ergebnisse aus der Liste entfernen zu können und Relevanzen angezeigt zu bekommen.

Die Frage nach der Berechenbarkeit von Prospector ergab keine klare Aussage, doch die Möglichkeit, das Benutzermodell zu betrachten, erhöhte diese für die Benutzer. Im Bezug auf die Verständlichkeit wurde herausgefunden, dass die Teilnehmer zwar die grundlegende Funktionsweise des Systems verstanden, nicht aber bestimmte Teile der Interaktion. Die Steuerbarkeit wurde positiv bewertet, nicht zuletzt, da das Benutzermodel verändert werden konnte.

Viele Benutzer fanden, dass der *Mehraufwand in der Bedienung* nur gering war, doch es herrschte keine Einigkeit, ob dieser dem Nutzen entsprach. Zwei Drittel der Teilnehmer sahen ihre *Privatsphäre* durch Prospector nicht beeinträchtigt, unter anderem weil sie die gesammelten Daten als wenig nützlich für jemand anders erachteten. Bei einigen gab es aber auch Sorgen, dass die Daten für Marketingzwecke verwendet würden. Bei der Frage, ob ihr *Sucherlebnis* durch Prospector eingeschränkt wurde, gab es hingegen keine klare Aussage, ebensowenig wie bei jener, ob Prospector die Interessen erfolgreich beim Umreihen berücksichtigte.

Für genauere Aussagen, und um die gesamten Auswirkungen der Personalisierung untersuchen zu können, wäre eine längere Interaktion der Benutzer mit dem System nötig gewesen. Auch wurde einer der Hauptmechanismen, nämlich das Initialisieren leerer Benutzermodelle aus Gruppenmodellen, nicht berücksichtigt. Aus diesem Grund wurde mit der neuen Version von Prospector erneut eine Evaluierung durchgeführt, die in der Folge beschrieben wird.

## 7.2 Gestaltung

In diesem Abschnitt wird das *Design der aktuellen Evaluierung* erklärt. Um dieses nachvollziehen zu können, ist es wichtig, den genauen Ablauf, die verwendeten Instrumente zur Messung quantitativer und qualitativer Merkmale sowie die teilnehmenden Personen zu kennen.

### 7.2.1 Ablauf

Im ersten Schritt wurden die *Teilnehmer rekrutiert*. Die genaue Vorgehensweise wird weiter unten näher erklärt. Anschließend wurde der erste *Fragebogen vorbereitet* und in dem Online-Dienst für Umfragen SurveyMonkey[1] eingerichtet. Im Vergleich zu anderen Diensten bot dieser für ein geringes monatliches Entgelt die beste Leistung im Bezug auf Funktionalität und Bedienkomfort.

Der nächste Schritt war, eine *öffentlich zugängliche Version von Prospector* zu installieren. Die Teilnehmer sollten das System von überall aus benützen können, daher wurde es auf einem Rechner mit einer öffentlichen IP-Adresse und einem entsprechenden DNS-Eintrag (`prospector.fim.uni-linz.ac.at`) aufgesetzt. Über einen verschlüsselten Fernzugriff wurden von diesem Rechner täglich die Datenbank und die Protokolle der Systemüberwachung gesichert.

Zum *Start der Evaluierung* erhielten die Teilnehmer eine Mail, in der zuerst das Prinzip von Prospector und insbesondere die Gruppen-Funktionalität noch einmal erklärt wurden. Hingewiesen wurde auch auf die beigelegte Datenschutzerklärung (siehe Anhang A auf Seite VI), die versicherte, dass die Benutzung anonym ist, und die anfallenden Daten nur für die Studie verwendet, an keine Dritten weitergegeben werden, und gegen unerlaubte Zugriffe, Veränderung oder Löschen geschützt werden.

Die Teilnehmer wurden darauf hingewiesen, die für das reibungslose Funktionieren von Prospector nötigen Cookies in ihrem Browser zu aktivieren. Weiters wurde die Möglichkeit, mittels der OpenSearch-Definition direkt aus dem Browser heraus zu suchen, erklärt. Im Anschluss folgte eine Beschreibung, wie weiter vozugehen sei:

1. *Hilfeseite durchlesen*: in ihr wurden die nötigen Schritte zum Registrieren, Angeben der Interessen, Suchen, Betrachten und Filtern der Ergebnisse, Umreihen, Bewerten und Bearbeiten des Benutzermodells, unterstützt durch entsprechende Screenshots erklärt.
2. *Fragebogen ausfüllen*: der Link zum Online-Fragebogen in SurveyMonkey und ein Hinweis auf die veranschlagte Dauer (5–10 Minuten) waren angegeben.
3. *Registrieren bei Prospector*: es wurde darum gebeten, sich wie in der Hilfe beschrieben über den Link zu Prospector zu registrieren und die Interessen anzugeben.
4. *Hinweis auf OpenSearch*: dieser Schritt war als optional gekennzeichnet.
5. *Verwendung von Prospector*: die Teilnehmer wurden gebeten, das System anstelle von oder in Kombination mit ihrer bevorzugten Suchmaschine zu verwenden.

Fünf Tage nach dem Start der Evaluierung wurde eine Mail mit dem Link zum *zweiten Fragebogen* ausgesendet. Die Teilnehmer wurden gebeten, diesen gemäß den ersten Erfahrungen, die sie mit Prospector bereits sammeln durften, auszufüllen. Ein Hinweis auf die Dauer von 5–10 Minuten war erneut gegeben. Weitere sechs Tage später wurde die *abschließende Mail*

[1] `http://www.surveymonkey.com/`

versendet. Diesmal sollten die Teilnehmer den Fragebogen im Hinblick auf die Erfahrungen seit dem letzten Ausfüllen beantworten. Es wurde auch der Hinweis gegeben, dass Prospector noch für rund eine Woche nach dem Ende der Studie zur Verfügung steht und bei Interesse die Ergebnisse zugesendet werden. Zum Schluss wurde ein Dank für die Mithilfe bei der Evaluierung ausgesprochen. Da nach ein paar Tagen noch immer nicht alle Teilnehmer geantwortet hatten, wurde noch einmal eine Erinnerung mit dem Link zum dritten Fragebogen ausgesendet.

### 7.2.2 Instrumente

Bei der Evaluierung kamen zwei Messinstrumente zum Einsatz: *Fragebögen* und *Benutzungsprotokolle*. Über die Studie verteilt wurden insgesamt drei Fragebögen eingesetzt. Diese sollten zuerst einen Überblick über die Eigenschaften, Fähigkeiten und Vorlieben der Teilnehmer geben, dann nach Erfahrungen mit Prospector fragen und zum Schluss Änderungen in den Einstellungen messen und Probleme bei der Gebrauchstauglichkeit aufdecken. Im Anhang ab Seite VII finden sich die vollständigen Fragebögen, hier nur ein kurzer Überblick zur Ausrichtung der Fragen:

- *Fragebogen 1*: demographische Daten (Geschlecht, Alter, Bildung), Computererfahrung, Online-Tätigkeit, Erfahrung mit Personalisierung, bevorzugte Suchmaschine (Häufigkeit der Verwendung, Personalisierung, Bewertung), Privatsphäre im Internet, Erwartungen an Prospector und die Funktion zum Betrachten und Verändern des Benutzermodells.
- *Fragebogen 2*: Verwendungshäufigkeit (Prospector und bevorzugte Suchmaschine), Bewertung, Vergleich der Suchleistung, gute/schlechte Erfahrungen, bevorzugtes System, Verständlichkeit und Genauigkeit des Modells, Änderungen am Modell.
- *Fragebogen 3*: wie Fragebogen 2 (um Veränderungen festzustellen), zusätzlich Kennzahlen der Gebrauchstauglichkeit (Berechenbarkeit, Verständlichkeit, Steuerbarkeit, Unaufdringlichkeit, Privatheit, Erlebnisreichtum).

Als zweites Messinstrument wurde die in Abschnitt 6.5 beschriebene Systemüberwachung dazu verwendet, umfassende Protokolle der Systembenutzung durch die Teilnehmer zu erstellen. Die einzelnen Schritte der Interaktion konnten somit im Anschluss klar nachvollzogen werden. Zusätzlich entwickelt wurde speziell für die Evaluierung eine Möglichkeit, festzustellen, wann die Hilfe verlassen wurde, und welche anonymen Zugriffe (also nicht über eine Benutzer-ID verfolgbar) zu der gleichen Sitzung gehörten. Somit war es auch möglich zu überprüfen, ob die Teilnehmer den Anweisungen in der Mail zum Start der Studie Folge leisteten.

### 7.2.3 Teilnehmer

Für die Teilnehmer wurde auf ein sogenanntes „*convenience sample*“ zurückgegriffen. Dieses ergibt sich aus jenen Personen, zu denen man unmittelbar Kontakt hat, also meistens Freunde, Bekannte und Kollegen. Hierbei ist immer die Gefahr gegeben, dass diese Auswahl das

Ergebnis verzerren könnte. Aufgrund eines knappen Zeitbudgets war dies aber die einzige Möglichkeit, rasch an ausreichend Teilnehmer zu kommen.

Die *Aufforderung zur Teilnahme* wurde an 126 Personen versendet. In ihr wurde Prospector kurz vorgestellt, erklärt, was von den Teilnehmern erwartet würde, und dass die Studie vollkommen anonym und online abläuft. Der Zeitplan wurde detailliert dargelegt und auch noch einmal unterstrichen, dass keine Verpflichtung besteht, bei einer Teilnahme das System immer zu verwenden. Hingewiesen wurde auch darauf, dass die Daten an keine Dritten weitergegeben werden. Zum Schluss wurde um ehestmögliche Rückmeldung bei Interesse gebeten und Kontaktinformation für weitere Fragen gegeben. Ein Screenshot von Prospector lag zur Illustration bei (siehe Abbildung 6.12 auf Seite 105).

*Rückmeldungen* mit konkretem Interesse kamen von 33 Personen. Dies entspricht einer Rücklaufrate von etwas mehr als 25%. Einige der Personen schränkten jedoch ihre prinzipielle Bereitschaft zur Teilnahme ein, weil sie beispielsweise während der Zeit der Studie teilweise keinen Zugang zum Internet hatten. Sie wurden aufgefordert, trotzdem zu versuchen, so oft es geht mit Prospector zu arbeiten. Vor diesem Hintergrund ist auch die etwas niedrigere Zahl an Rückmeldungen auf die einzelnen Fragebögen zu erklären.

## 7.3 Ergebnisse

Der Fokus der Evaluierung für die Zwecke dieser Arbeit lag klar auf den *qualitativen Rückmeldungen*. Aus diesem Grund werden diese in der Beschreibung der Ergebnisse auch im Vordergrund stehen. Die Protokolle der Systemüberwachung können weitere wertvolle Hinweise zur Systembenutzung liefern; ihre Daten wurden bis jetzt aber nur in kleinen Teilen ausgewertet.

Insgesamt antworteten auf den ersten Fragebogen 22 Teilnehmer und auf die anderen zwei jeweils 21. In den folgenden Unterabschnitten wird zuerst die Demographie und Internet-Nutzung der Teilnehmer beschrieben und dann auf Schwierigkeiten und Herausforderungen eingegangen, die bei der Evaluierung auftraten, und die Ergebnisse beeinflussten. Anschließend werden die konketen Erkenntnisse, gegliedert nach grundlegenden Kriterien, näher erläutert.

### 7.3.1 Demographie und Nutzung von Online-Diensten

Bei den 22 Teilnehmern, die den ersten Fragebogen ausgefüllt haben, handelt es sich um eine sehr junge Gruppe mit einem durchschnittlichen Alter von 25,9 Jahren (Standardabweichung 2,7). 20 Personen waren männlichen und 2 weiblichen Geschlechts. Mehr als die Hälfte (12) hatte schon ein Studium abgeschlossen, eine Person ein College und neun Personen die Mittelschule. Bei jenen, die ein konkretes Studienfach angegeben haben, das sie zum Zeitpunkt der Studie studierten oder bereits abgeschlossen hatten, lag Informatik (9) vor Wirtschaftswissenschaften (4), Wirtschaftsinformatik (3), Rechtswissenschaften (2) und Mechatronik (1).

Alle Teilnehmer gaben an, überdurchschnittlich gut mit dem *Computer* und Standard-Programmen wie Word und Internet Explorer umgehen zu können. Mehr als 70% gaben sich dabei auf einer siebenstufigen Skala von „sehr schlecht“ bis „sehr gut“ die höchste Bewertung. Graphische Darstellungen der Daten zu diesen und einigen bisher beschriebenen Ergebnissen finden sich in Abbildung 7.1 auf der nächsten Seite.

Befragt nach der Häufigkeit, mit der die Teilnehmer bestimmte *Tätigkeiten im Internet* ausführen zeigte sich folgendes Bild (siehe Abbildung 7.2 auf Seite 139): Alle verwenden täglich (19) oder nahezu täglich (3) E-Mail. Bei der Suche nach Informationen im Web zeigte sich mit 16 (täglich) und 6 (nahezu täglich) Teilnehmern ein ähnliches Bild. Über 70% chatten zumindest einmal die Woche, arbeiten mit Online-Banking oder konsumieren Filme, TV bzw. Musik im Internet. Ein bisschen unter diesem Wert liegt die Häufigkeit für das Herunterladen von Software.

Ein starker Anwendungsbereich ist auch das Lesen von Nachrichten mit über 90% täglicher bzw. nahezu täglicher Nutzung. Eher wenig genutzt werden e-Government-Dienste, Internet-Telephonie und Mehrbenutzerspiele bzw. virtuelle Welten. Insgesamt kann man angesichts dieser Zahlen davon ausgehen, dass die Teilnehmer durchwegs aktive Nutzer des Internet und seiner vielfältigen Angebote sind.

Bei der *Nutzung personalisierter Dienste* zeigte sich, dass große Anbieter wie Amazon und iGoogle sehr vielen Teilnehmern bekannt waren. Ein bisschen mehr als die Hälfte kannte hingegen Last.fm nicht, bei Movielens waren es sogar 77%. Genutzt wurde vor allem Amazon (14), Last.fm und iGoogle jedoch von nur jeweils drei Teilnehmern. Keiner der Befragten benutzte Movielens. Zu sehen sind die Verhältnisse auch in Abbildung 7.3 auf Seite 139. Bei den sonst noch verwendeten Systemen mit Personalisierungsfunktion wurden das Soziale Netzwerk Facebook, das Online-Auktionshaus eBay und die Handytarif-Suchmaschine Fonito.at angegeben. Das konkrete Ausmaß der Personalisierung bei diesen Systemen wurde allerdings nicht geprüft.

Alle Teilnehmer verwendeten Google als *bevorzugte Suchmaschine*, nur einmal wurde auch Yahoo als gleichwertig genannt. Über 60% nützten diese täglich, 30% sogar mehr als 15 mal. Ein bisschen mehr als ein Drittel (8) gab allerdings nur eine Nutzung von mehrmals pro Woche an. Vergleichen lassen sich die Zahlen anhand Abbildung 7.4 auf Seite 139. Wenig genutzt wurden die personalisierten Funktionen der bevorzugten Suchmaschine. Nur drei Teilnehmer bedienten sich dieser, zwölf hingegen nicht, sieben Teilnehmern waren diese unbekannt. Bei der Frage nach anderen verwendeten Suchmaschinen wurde Yahoo zweimal, Altavista einmal und ebenfalls einmal Spezialsuchmaschinen (z. B. für wissenschaftliche Artikel) genannt.

Aus qualitativer Sicht wurde die bevorzugte Suchmaschine überdurchschnittlich gut bewertet. Auf einer siebenstufigen Skala („stimme gar nicht zu“ bis „stimme voll zu“) ergaben sich Mediane von 6 für die Fähigkeit, sinnvolle Informationen zu finden, 7 für das schnelle Finden und 6 für die Effizienz. In Abbildung 7.5 auf Seite 142 sind die entsprechenden Verteilungen als Boxplots für die Frage zur bevorzugten Suchmaschine im ersten Fragebogen und

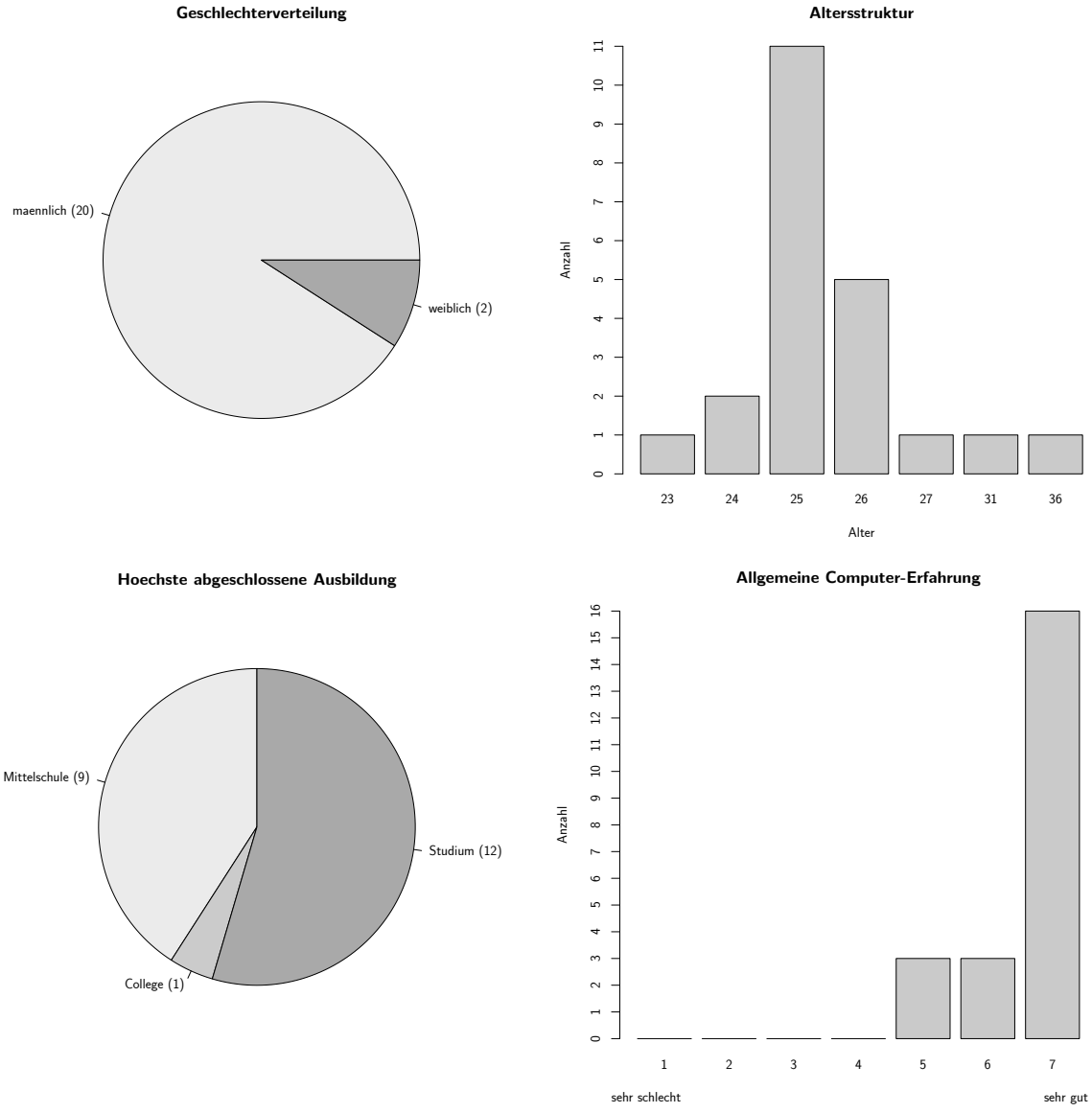


**Abbildung 7.1:** Demographische Daten der Evaluierungsteilnehmer

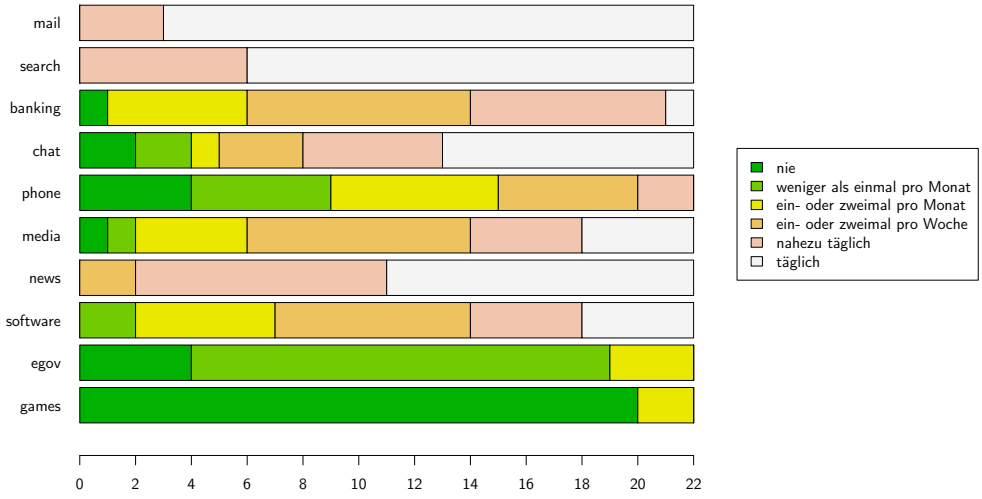


**Abbildung 7.2:** Nutzung des Internet durch die Evaluierungsteilnehmer

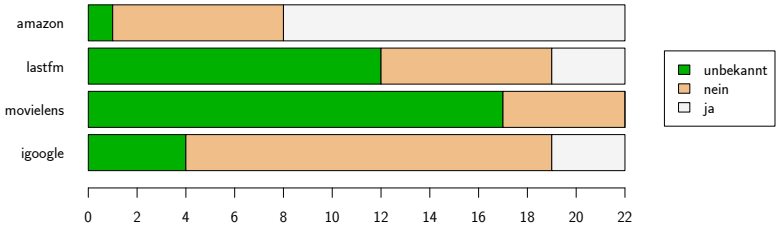


**Abbildung 7.3:** Nutzung personalisierter Dienste durch die Evaluierungsteilnehmer

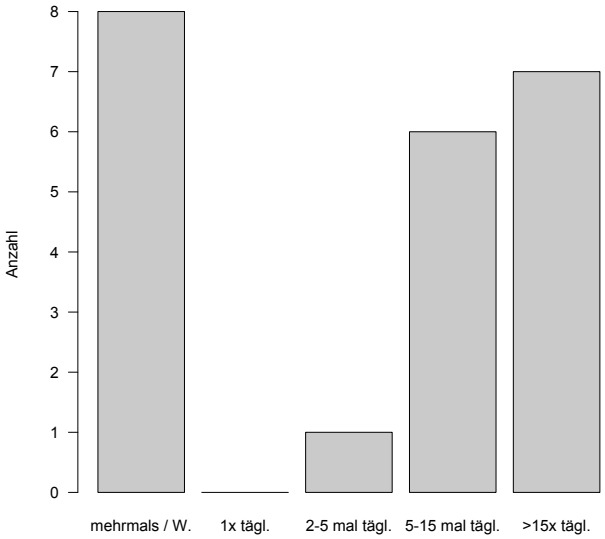


**Abbildung 7.4:** Nutzung der bevorzugten Suchmaschine durch die Evaluierungsteilnehmer

zu Prospector im zweiten und dritten dargestellt. Details zu den Ergebnissen finden sich in Unterabschnitt 7.3.5 und Unterabschnitt 7.3.4.

Acht Teilnehmer beantworteten die optionale Frage nach Bedenken bezüglich ihrer *Privatsphäre im Internet*. Drei davon hatten keine oder kaum Bedenken, die anderen fünf sorgten sich vor allem um Aufzeichnung und Auswertung von Verhaltensprofilen und persönlicher Daten. Aufgrund der Fragestellung kann angenommen werden, dass die 14 Teilnehmer, die nicht geantwortet haben, keine speziellen Bedenken haben.

### 7.3.2 Probleme

Bevor die eigentlichen Ergebnisse besprochen werden, müssen noch einige *erschwerende Umstände* erklärt werden, von denen angenommen wird, dass sie auf diese einen Einfluss hatten. Der wichtigste Aspekt betrifft die Vergleichbarkeit der Suchergebnisse: als Quell-Suchmaschinen standen den Teilnehmern die Meta-Suchmaschine von eTools (standardmäßig gewählt) und die zwei bekannten Systeme Yahoo! und MS Live Search zur Verfügung. Die Teilnehmer gaben aber geschlossen an, Google als bevorzugte Suchmaschine zu verwenden, weshalb ihre Vergleiche bei der Benutzung von Prospector auch diese als Referenz hatten. Sollte die Qualität der Quell-Suchmaschinen schlechter sein, so würde dies, selbst ohne Umreihen, Prospector angelastet.

Ein zweites Problem betrifft ebenfalls die Suchergebnisse, mit denen Prospector arbeitete. Diese wurden ohne Angabe einer gewünschten Sprache oder eines geopolitischen Gebiets von den Quell-Suchmaschinen abgerufen. Sie unterschieden sich daher sehr grundlegend von jenen, die Google liefern würde, wo ja bei Suchen aus einem österreichischen Browser bevorzugt deutschsprachige Ergebnisse geliefert werden. Auch die Funktion zur Korrektur falsch geschriebener Suchbegriffe war nicht wie in Google gegeben. Diese Mankos wurden von etlichen Teilnehmern wiederholt bemängelt und schufen eine schlechte Ausgangsbasis für Prospector.

### 7.3.3 Erwartungen

Zu Beginn wurden die Benutzer gefragt, was sie sich von *Prospector* auf Basis der zur Verfügung gestellten Informationen und Hilfe-Dokumente erwarten. Am öftesten genannt wurden dabei bessere (5) und für sie persönlich relevante Ergebnisse (5). Das System sollte schnell (5) und effizient (4) arbeiten, dabei die relevanten Ergebnisse entsprechend ihrer Relevanz umreihen (2) und die unwichtigen aussortieren (2). Positiv für das Einschätzen der Wichtigkeit wurde auch die Relevanzanzeige gesehen (1). Einmal genannt wurde die Unterstützung bei schlecht formulierten und komplexen Fragestellungen. Manche Teilnehmer waren aber auch einfach neugierig (1) oder gespannt aber skeptisch (1).

Vergleich man die Angaben der Teilnehmer, so stimmen diese recht gut mit den Eigenschaften von Prospector überein, die dem System in den zur Verfügung gestellten Unterlagen zugeschrieben werden. Die Teilnehmer scheinen die grundlegende Ausrichtung von Prospector

erkannt zu haben. Zu Beachten ist dabei, dass manche Teilnehmer mehrere Aspekte genannt haben und die Werte daher insgesamt nicht der Teilnehmerzahl entsprechen.

Danach gefragt, ob sie es als nützlich erachten, ihre *Benutzermodell zu betrachten und zu verändern*, beantworteten elf Teilnehmer mit Ja, einer mit Nein, einer sah dies als nicht nützlich aber interessant an, zwei nur teilweise und sieben enthielten sich einer Wertung. Unter den positiv gestimmten wurden unter anderem die Möglichkeit zur genaueren Anpassung, vermehrte Transparenz und das Ausgleichen seltener Suchen als Anwendungsmöglichkeiten genannt. Der skeptische Teilnehmer meinte, für eine sinnvolle Veränderung sie zu viel Wissen erforderlich. Alle diese Antworten zeigen legitime Szenarien aber auch berechtigte Bedenken auf.

Die abschließende Frage, *wie* sich die Teilnehmer konkret vorstellen könnten, das *Modell zu verändern*, wurde von weniger als der Hälfte aussagekräftig beantwortet. Zwei mal genannt wurde der Fall, dass bei unpassenden Ergebnissen eine bessere Anpassung helfen könnte. Eine Person wollte damit die Einflüsse zweier anderer Personen, die ebenfalls seinen Rechner benutzen, ausgleichen. Ein Teilnehmer hatte (vermutlich aufgrund von Erfahrungen mit der personalisierten Google Suche) das falsche Bild einer „History“, die er bearbeiten könnte. Eine interessanter Ansatz war, es gar nicht selbst zu verändern, sondern eigenständig wachsen zu lassen.

### 7.3.4 Nützlichkeit

Die Teilnehmer wurden im dritten Fragebogen gefragt, ob Prospector es schafft, *relevante Suchergebnisse zu liefern*. Vier Teilnehmer beantworteten dies mit Ja, sechs mit kleinen Abschlägen, und zwei bewerteten die Ergebnisse als nur ein bisschen besser als die von Google. Vier Personen fand dies nur teilweise gegeben, drei überhaupt nicht, zwei gaben keine oder keine gültige Antwort. Insgesamt mehr als die Hälfte der Teilnehmer fand also, dass Prospector (ziemlich) relevante Ergebnisse lieferte, teilweise sogar ein bisschen besser als Google. Diese leicht positive Tendenz lässt auf eine grundlegende Nützlichkeit des Systems schließen.

In einer geschlossenen Frage, wo die Teilnehmer die Fähigkeit von Google und Prospector, sinnvolle Informationen zu finden, auf einer siebenstufigen Skala von „stimme gar nicht zu“ bis „stimme voll zu“ angeben mussten, relativierte sich das vorangegangenen Ergebnis wieder. Die Lage der Einschätzungen für Google (erster Fragebogen) und Prospector (zweiter und dritter Fragebogen) ist in Abbildung 7.5 auf der nächsten Seite unter „useful*n*“ zu sehen. Die daraus entstehende Vermutung, dass Prospector schlechter bewertet wurde, ließ sich statistisch untermauern. Wilcoxon Rangsummentests bestätigten, dass Prospector mit einer Irrtumswahrscheinlichkeit von weniger als 1% niedrigere Wertungen erhielt. Aufgrund der Art, wie dies gemessen wurde (geschlossene Frage) ist dieses Ergebnis als aussagekräftiger als das vorhergehende zu bewerten.

Untersucht wurde auch, ob sich die Einschätzung von Prospector zwischen den zwei Fragebögen änderte. Mit gepaarten Wilcoxon Rangsummentests konnte gezeigt werden, dass diese

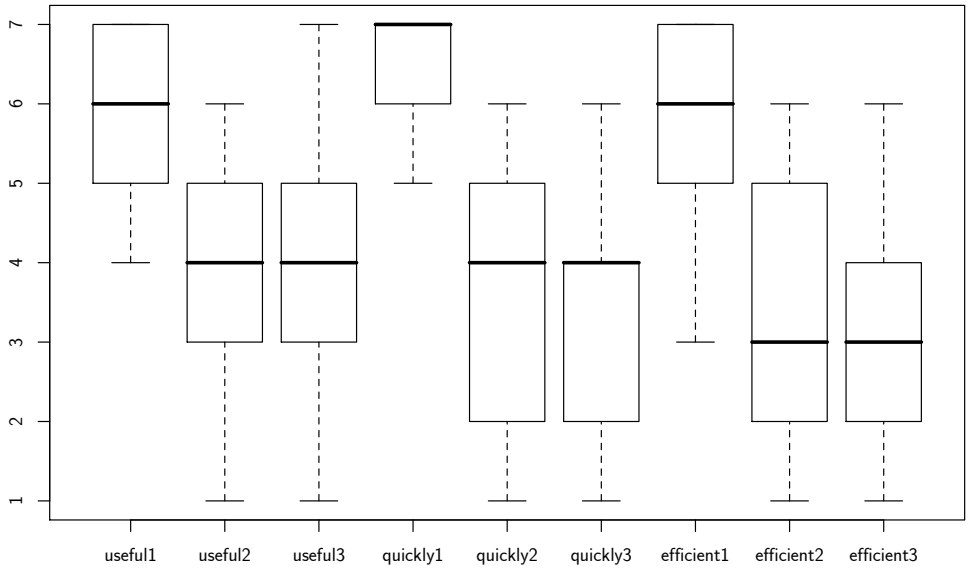


**Abbildung 7.5:** Einschätzungen zur bevorzugten Suchmaschine und zu Prospector

Unterschiede nicht mit einer Irrtumswahrscheinlichkeit von 5% oder weniger statistisch signifikant sind. Die Einschätzung von Prospector hat sich also im Verlauf der Evaluierung nicht merklich verändert.

### 7.3.5 Suchleistung

Die Suchleistung setzt sich aus einigen Kriterien zusammen, die nicht getrennt untersucht wurden. Konkret sind dies die Genauigkeit des Ergebnisses, sein Umfang und die Systemgeschwindigkeit. Mit geschlossenen Fragen wurden die Teilnehmer in gleicher Weise wie in Unterabschnitt 7.3.4 beschrieben aufgefordert, einzuschätzen, ob die jeweilige Suchmaschine schnell Informationen findet und ob das System effizient ist. Zu sehen ist die Lage der einzelnen Einschätzungen wiederum in Abbildung 7.5 unter „quickly$n$“ und „efficient$n$“.

Wie zuvor ergab sich eine mit einer Irrtumswahrscheinlichkeit von weniger als 1% statistisch signifikant niedrigere Wertungen von Prospector im Vergleich zu Google. Die Unterschiede zwischen den zwei Messungen von Prospector waren auch dieses Mal nicht signifikant (Irrtumswahrscheinlichkeit größer als 5%). Der Grund für die schlechtere Bewertung von Prospector liegt mit großer Wahrscheinlichkeit daran, dass Google mit üblichen Antwortzeiten unter einer Sekunde deutlich schneller wirkte als Prospector, bei dem die Teilnehmer im Durchschnitt drei bis vier Sekunden auf ein Ergebnis warten mussten.

Zur Untersuchung der *Leistung des Prospector-Algorithmus* an sich wurde gefragt, welche Suchmaschine das/die gesuchte(n) Ergebnis(se) weiter vorne in der Liste lieferte. Sowohl beim zweiten als auch beim dritten Fragebogen gaben 15 Teilnehmer an, dass die gesuchten Ergebnisse bei ihrer bisherigen Suchmaschine (d. h. Google) weiter vorne gereiht waren. Drei

Personen bewerteten die Suchleistung als gleichwertig bzw. gaben sich unentschieden. Lediglich im dritten Fragebogen stimmten zwei Teilnehmer für Prospector. Bei jenen, die auch einen Grund für ihre Wahl angegeben haben, fanden sich zumeist die oben beschriebenen, fehlenden Funktionalitäten von Google (Einschränkung auf Deutsch bzw. Österreich, Korrektur von Tippfehlern). Ein Teilnehmer bemerkte auch, dass er mit Prospector immer länger benötigte, um sich zurechtzufinden und die gewünschten Seiten zu überfliegen.

### 7.3.6 Zufriedenheit

Die Teilnehmer wurden in Fragebogen zwei und drei jeweils nach ihrem *besten und schlechtesten Erlebnis mit Prospector* gefragt. Positiv hervorgehoben wurde dabei in etlichen Antworten die Klassifizierung der Suchergebnisse in der ODP-Ontologie. Als überaus nützlich erachteten hier die Teilnehmer die Möglichkeit zum Filtern über die Themenhierarchie auf der linken Seite. Besonders hilfreich war Prospector laut einem Teilnehmer bei Suchen nach Begriffen, die in verschiedenen Themengebieten Ergebnisse liefern können. Einige bebachteten auch die erfolgreiche Auflösung von Mehrdeutigkeiten bei Suchanfragen und die entsprechende Umreihung nach ihren Interessen. Dies ist ein wertvoller Indikator, dass Prospector tatsächlich die versprochene Funktionalität bietet.

Gute Hinweise für die Gründe, warum Prospector die Teilnehmer insgesamt nicht überzeugen konnte, gaben die Antworten auf die Frage nach den *schlechtesten Erfahrungen* mit dem System. Ganz zuoberst wurden hier von rund einem Drittel schlechte Ergebnisse mit keinen oder kaum relevanten (aber oft englischssprachigen) Seiten genannt. Negativ erwähnt wurde auch, dass Startseiten und große Portale, anders als bei Google, nicht weit vorne gereiht waren. Schlechte Erfahrungen machten Teilnehmer beim Suchen mit Teilen einer URL in der Anfrage, denn Prospector reihte die Seiten mit den passenden Adressen nicht ganz vorne. Von technischer Seite wurde bemängelt, dass zeitweise Exceptions auftreten, das System zu stark von Cookies abhängig ist und langsamer als Google arbeitet.

In Kommentaren zu anderen Fragen zeigte sich ein ähnliches Bild: auch hier wurde kritisiert, dass Prospector schlecht mit deutschen Suchbegriffen umgehen kann und wenige Ergebnisse in deutscher Sprache liefert. Widerum erwähnt wurde zudem, dass Startseiten und jene, wo der Suchbegriff direkt in der URL vorkommt, vorgereiht werden müssten. Ein ganz grundsätzliches Problem wurde auch angesprochen: es sind zu wenig Seiten klassifiziert und können daher nicht bewertet werden.

Konkret gefragt, *welche Suchmaschine sie bevorzugen*, wählten zehn Teilnehmer beim zweiten Fragebogen Google, zwei Prospector, drei legten sich nicht eindeutig fest und sechs machten keine Angabe. Beim dritten Fragebogen verstärkte sich diese Tendenz noch mit 17 Stimmen für Google, einer für Prospector und drei Enthaltungen. Die Hauptgründe für diese Wahl sind mit besseren Ergebnissen, besserer Reihung, den Funktionen zur regionalen Einschränkung der Suche und einer schnelleren Verarbeitung konsistent zu den schon genannten. Neu war die Angabe von Vorlieben für Google, weil es vertrauter ist und eine Bildersuche bietet.

Insgesamt ergibt sich für *Suchleistung und Zufriedenheit* mit Prospector kein besonders positives Bild, wenngleich manche Teilnehmer am Ende Prospector bevorzugten. Viele der angeführten Gründe sind jedoch rein technisch gesehen nicht sonderlich schwierig zu verbessern, sofern eine entsprechende Unterstützung (z. B. beim Einschränken auf eine Sprache) durch die Quell-Suchmaschinen gegeben ist. Trotzdem stimmte positiv, dass eine der Hauptsorgen, zu wenig klassifizierte Ergebnisse, nur am Rande als Problem identifiziert wurde. Interessant ist auch, dass nach lediglich zwei Personen, die beim zweiten Fragebogen keine schlechteste Erfahrung mit Prospector hatten, beim dritten Fragebogen schon neun Teilnehmer kein besonderes Erlebnis berichteten. Ganz allgemein zeigte sich, dass personalisierte Suchfunktionen keine der mittlerweile schon zum Standard gewordenen Funktionalitäten aufwiegen kann, sondern dass diese ebenfalls implementiert sein müssen.

### 7.3.7 Modellierung

Die Evaluierung von Prospector zeigte auch eine Reihe positiver Aspekte auf. Ein für die Entwicklung eines adaptiven Systems überaus erfreuliches Ergebnis ist, dass laut den Teilnehmern die *Benutzermodelle* ihre *Interessen und Suchpräferenzen widerspiegelten.* Acht Teilnehmer beantworteten die entsprechende Frage in Fragebogen zwei mit Ja, für vier traf dies eher schon zu, für zwei eher nicht, und fünf waren nicht der Meinung. Im dritten Fragebogen bewerteten zehn Teilnehmer den Inhalt ihres Modells positiv, vier eher positiv, zwei neutral, einer eher negativ, und drei fanden, dass das Modell nicht korrekt war.

Als zusätzliche Überprüfung wurden die Teilnehmer auch gefragt, ob ihnen das *Modell klar* war. Knapp zwei Drittel (14) beantworteten diese Frage mit Ja, drei weiteren war es im Großen und Ganzen klar, einem nur teilweise. Lediglich zwei Personen war nicht klar, wie dieses Modell entstanden ist, einer enthielt sich. Trotz dieses grundlegenden Verständnisses für die Funktion des Modells gaben nur zwei Teilnehmer an, daran *Änderungen vorgenommen* zu haben. Dies kann als Bestätigung dafür gesehen werden, dass die Inhalte des Modells dem entsprechen, was die Benutzer interessiert.

### 7.3.8 Usability

Da ein beträchtlicher Teil der Arbeit in die Verbesserung der Gebrauchstauglichkeit von Prospector geflossen war, wurde diese mit fünf der in in Abschnitt 2.3 genannten sechs Kriterien nach Jameson [Jam03] geprüft.

#### Verständlichkeit

Die Teilnehmer wurden gefragt, ob sie verstünden, wie Prospector funktioniert. Hierbei ergab sich ein wenig aussagekräftiges Ergebnis: acht antworteten mit ja, drei mit eher ja, drei verstanden Prospector nur teilweise und sieben gar nicht. Interessant ist auch der Hinweis einer

Person, es gäbe zu viele Einstellmöglichkeiten. Dies deutet darauf hin, dass der Inhalt des Benutzermodells zwar klar ist, dessen Auswirkungen auf die Suchergebnisse jedoch nicht.

Ein ausgeprägteres Ergebnis ergab sich bei der Frage, ob es Teile oder Funktionen in Prospector gibt, die schwer zu verstehen seien: mehr als zwei Drittel (15) der Teilnehmer waren der Ansicht, dass dies nicht der Fall war. In zwei Kommentaren wurde die Bedienung als „sehr intuitiv“ und „selbsterklärend“ bezeichnet. Konkrete Fragen gabe es danach, wie die Ergebnisse klassifiziert werden, warum schlechtere Ergebnisse geliefert werden, wenn doch auf bestehende Suchmaschinen zurückgegriffen wird und erneut, warum keine deutschsprachigen Seiten priorisiert werden.

#### Berechenbarkeit

Wenig aussagekräftig waren die Ergebnisse bei der Frage, ob die Teilnehmer der Meinung wären, dass Prospector berechenbar arbeitet: fünf sagten ja, drei eher ja, einer teilweise, einer eher nein, vier nein und sieben gaben keine Antwort. Ein Teilnehmer gab konkret an, das Prinzip verstanden zu haben, aber nicht zu sehen, ob sich sein Profil in den Suchergebnissen widerspiegelt. Ein anderer sah die Einflüsse des Modells und des Profils nur manchmal gegeben.

#### Steuerbarkeit

Die Teilnehmer wurden gefragt, ob sie der Meinung seien, bei der Verwendung von Prospector die Kontrolle zu haben. Hier ergab sich ein leicht positives Ergebnis mit sechs Übereinstimmungen, drei Personen für eher ja, drei für teilweise, zwei für wenig und vier für nein bei drei Enthaltungen. Ein Teilnehmer relativierte dies mit „bei den eingestellten Themen schon, außerhalb der Einstellungen nicht“. Angemerkt wurde auch, dass die Suche natürlich stark von der dahinterliegenden Quellsuchmaschine beeinflusst wird. Ein Teilnehmer meinte gar (leicht scherzend), er wolle gar keine Kontrolle, es solle einfach immer das richtige Suchergebnis kommen.

#### Unaufdringlichkeit

Befragt wurden die Teilnehmer auch, ob sie fänden, dass die Benutzung von Prospector viel Aufwand bedeutet. Jene, die dies bejahten, und jene, die es verneinten, hielten sich in etwa die Waage. Angesichts der in früheren Kapiteln erwähnten Zurückhaltung vieler Benutzer, mehr als nur das Nötigste bei einer Suche aufzuwenden, ist dies eigentlich schon ein gutes Ergebnis. Eine mögliche Erklärung, warum trotzdem rund die Hälfte die Bedienung von Prospector als aufwändig betrachtete, ist, dass das System im Verhältnis dazu nicht jene guten Ergebnisse lieferte, die erwartet wurden. Konkret genannt wurden auch eine langsamere Antwortzeit,

mehr nötige Klicks im Vergleich zu Google, und auch, dass oft doppelt gesucht werden musste. Positiv zu sehen ist der Kommentar, dass die Bewertung keinen großen Zusatzaufwand darstellt, und die Einsicht, Prospector müsse halt erst „angelernt“ werden.

#### Erlebnisreichtum

Geteilter Meinung waren die Studienteilnehmer bei der Frage, ob das System die Vielfalt der Suchergebnisse herabsetzt: acht meinten ja, zwei eher ja, einer eher nein und neun nein (eine Enthaltung). Anhand der Kommentare lässt sich erkennen, dass einige Teilnehmer fokussiertere Ergebnisse aber nicht als Einschränkung sondern als wünschenswerte Eigenschaft verstanden. Prospector steuere schließlich ja auf bestimmte Interessen hin. Bemängelt wurde auch das Fehlen einer Bildersuche.

#### Privatheit

Das eindeutigste Ergebnis lieferte die Frage, ob die Verwendung von Prospector die Privatsphäre beeinträchigt: über 70% der Teilnehmer gaben an, dass sie nicht dieser Meinung seien. Nur zwei waren anderer Meinung, und zwei meinten, dass dies nicht problematischer sei als bei anderen Diensten. Ein Teilnehmer meinte, dass eine große Menge persönlicher Vorlieben gesammelt würden, man aber dem Hersteller der Suchmaschine vertrauen müsse. Optimistisch gab sich ein anderer mit der Ansicht, die Einstellungen seien grundsätzlich für niemand anderen sichtbar bzw. nutzbar. Angesichts der klaren Darstellung der gesammelten Daten im Benutzermodell ist die insgesamte Einschätzung der Teilnehmer aber überraschend.

# Kapitel 8

# Zusammenfassung und Ausblick

Personalisierte Suche ist in der Informatik ein aktuelles Thema. Dies beweisen nicht nur zahlreiche wissenschaftliche Arbeiten sondern auch konkret im Einsatz befindliche Systeme. Vor diesem Hintergrund fand die in dieser Arbeit beschriebene Weiterentwicklung der personalisierten Meta-Suchmaschine Prospector statt. Nach der Erstellung des ersten Prototypen und dessen starker Verbesserung in Bezug auf die Algorithmen zur Modellierung der Benutzerinteressen und Personalisierung der Ergebnisreihenfolge stellt diese Version einen weiteren Schritt in Richtung bessere Ergebnisse und ansprechendere Bedienung dar.

## 8.1 Zusammenfassung

Ein Hauptziel der Entwicklung war die Schaffung einer *soliden technologischen Basis* für den Algorithmus von Prospector. Durch die Verwendung des Carrot$^2$-Frameworks und damit zusammenhängender Entwicklungen konnte dies erreicht werden. Neben den klaren Strukturen für die Implementierung und Verbindung der einzelnen Teile des Prospector-Systems brachte dieser Schritt aber auch weitere Vorteile. So verfügt Prospector nicht mehr nur über einen Zugriff auf die Suchergebnisse von Google, sondern auch von etlichen anderen aktuellen Suchmaschinen.

Auch die *Web-Oberfläche* von Carrot$^2$ sorgte für einen Qualitätssprung bei Prospector. Nun war es möglich, die Klassifizierungen der einzelnen Ergebnisse hierarchisch darzustellen und zum Filtern zu verwenden. Die Evaluierung hat gezeigt, dass diese Funktionalität von den Benutzern geschätzt wird. Auf Basis der Web-Oberfläche wurden die bisherigen Möglichkeiten, mit dem System zu interagieren, in zeitgemäßer Weise verbessert. Hierbei lag der Fokus vor allem auf der Vereinfachung für den Benutzer, wobei viele der Entscheidungen aufgrund der Ergebnisse der zweiten Evaluierung getroffen wurden.

Die Anzeige der Relevanzen wurde derart gestaltet, dass die Benutzer nicht mit internen Werten des Systems konfrontiert sind. Dieser Ansatz wurde auch erfolgreich bei den Dialogen der Gruppen-Affinitäten und dem Benutzermodell umgesetzt. Schieberegler wurden durchgehend verwendet und anstelle von numerischen Werten wurde auf eine einfach zu verstehende Skala mit farblicher Unterstützung zurückgegriffen. Durch den Einsatz von Dialogen anstelle

ganzer Webseiten konnten in diesem Fall und auch bei Login, Logout und Registrieren eine benutzerfreundlichere Bedienung geschaffen werden.

Eine weitere wichtige Änderung ist die Möglichkeit, nun direkt in der Ergebnisliste positiv und negativ zu bewerten. Aufgrund diverser Probleme mit dem Bewertungsbereich über den angezeigten Seiten in der zweiten Version wurde auf diesen nun verzichtet. Stattdessen werden die Benutzer bei der Rückkehr zur Ergebnisliste visuell auf die Möglichkeit zu Bewerten aufmerksam gemacht. Ebenfalls neu ist die dynamische Umreihung der Ergebnisse nach Gruppenmodellen, ohne die komplette Seite neu laden zu müssen. Für moderne Browser ermöglicht die nun verfügbare OpenSearch-Definition zudem das Suchen direkt aus dessen Suchfeld heraus, wie man es von der dort standardmäßig eingestellten Suchmaschine Google kennt.

Doch nicht nur an der Oberfläche gab es Weiterentwicklungen bei Prospector. Auch der *Algorithmus* wurde verbessert und kann nun eine beliebige Zahl von Themen, in denen ein Ergebnis klassifiziert ist, sowohl beim Berechnen der Relevanz als auch beim Bewerten verarbeiten. Zuvor wurde eine nicht immer funktionierende Heuristik zum Ermitteln des wichtigsten Themas verwendet. Für die Berechnung der Relevanz werden die Gewichtungen der einzelnen Klassifizierungen nun mit der Gruppen-Affinität gewichtet, zur endgültigen Relevanz aufsummiert und somit die Interessen des Benutzers auch an dieser Stelle mit einbezogen.

Eine sehr wichtige Neuerung ist auch, dass die originale Reihenfolge der Ergebnisse stärker in die Reihung einfließt. Zuvor war die Reihung einzig von der berechneten Relevanz abhängig, was zu einem sprunghaften Verhalten bezüglich der Ränge einzelner Ergebnisse führen. Durch die Berücksichtigung der originalen Reihenfolge wurde ein feiner abgestuftes Verhalten erreicht werden. Große Sprünge einzelner Ergebnisse in der Liste, sofern diese nicht durch das Benutzermodell gerechtfertigt sind, werden verhindert.

Eine der wichtigsten nicht-funktionalen Verbesserungen betrifft die *Verarbeitungsgeschwindigkeit*. Hier konnten die Flaschenhälse des Systems durch umfangreiche Profiling-Maßnahmen erfolgreich erkannt und mit gezielten Optimierungen behoben werden. Wo Prospector mit Antwortzeiten von einer halben Minute und mehr nicht in der Praxis einsetzbar wäre, präsentiert das System nun nach durchschnittlich drei bis vier Sekunden ein Ergebnis und steht somit anderen Suchmaschinen um nichts nach.

Ebenfalls weiterentwickelt wurde die *Systemüberwachung und -protokollierung*. Sie ermöglicht nun für Zwecke der Evaluierung die genaue Verfolgung alle Aktionen im System, egal ob sie Interaktionen von Benutzern oder die Veränderungen in Modellen betreffen. Durch den nunmehrigen Einsatz einer Datenbank als Speicherort für die Protokolle liegen diese sofort in einer strukturierten und durchsuchbaren Weise vor. Besonderes Augenmerk wurde auch darauf gelegt, dass das System nicht durch die Protokollierung verlangsamt wird.

Um die gesamten Verbesserungen zu testen, wurde eine *Evaluierung* der neuen Version von Prospector durchgeführt. Im Gegensatz zu bisherigen Studien fand diese über einen längeren Zeitraum im realen, alltäglichen Einsatz statt. Dabei zeigten sich sowohl positive als auch negative Ergebnisse. Als besonders positiv wurden die Qualität der Benutzermodellierung

und Punkte der Gebrauchstauglicheit und Bedienfreundlichkeit hervorgehoben. Negativ wirkte sich aus, dass die originalen Suchergebnisse auf Englische Benutzer zugeschnitten waren und im Gegensatz zu Google keine Einschränkung auf deutschsprachige Inhalt möglich war. Dieses Manko, gepaart mit kleineren Defiziten in der Reihung, ließ die Teilnehmer weiterhin ihre bisherige Suchmaschine Google favorisieren. Glücklicherweise sind viele der angesprochenen Schwachstellen technisch nicht allzu schwer lösbar. Positiv ist auch, dass in der zweiten Evaluierung genannte Probleme fast überhaupt nicht mehr auftraten.

## 8.2 Ausblick

Die Ergebnisse der Evaluierung und auch außertürliche, positive Rückmeldungen von Teilnehmern schaffen für Prospector eine Perspektive für die Zukunft. Insbesondere in der Endphase dieser Arbeit wurden neue *Ideen für die Verbesserung des Systems* geboren. Diese beziehen sich vor allem auf die algorithmischen Aspekte von Prospector. Eine kleine Auswahl davon sei in der Folge vorgestellt.

Eine Maßnahme, die zur weiteren Vereinfachung des Systems führen könnte, wäre die *Vereinigung der Gruppen-Affinitäten mit dem Benutzermodell*. Zu diesem Zweck würde jenen ODP-Themengebieten oberster Ebene, von denen auch eine gleichnamige Gruppe existiert, im Benutzermodell eine Sonderstellung zukommen. Ihre Gewichtung entspräche gleichzeitig auch der Affinität zu dieser Gruppe. Die Möglichkeit zum Verändern dieser Affinitäten wäre somit in die Ansicht des Benutzermodells integriert und für die Benutzer müsste nicht einmal sichtbar werden, dass es sich hierbei um spezielle Werte für ihre Beziehung zu Gruppen handelt.

Eine weitere Idee wäre, nicht nur für die Themen oberster Ebene Gruppen zu schaffen. Interessant wäre es auch, für *jede Sprache und jedes Land eine Gruppe* zu haben. Durch diese Maßnahme könnte ein Teil der bei der Evaluierung aufgetretenen Probleme mit nicht lokalisierten Ergebnissen vermieden werden. Mit diesem Ansatz würde auch der Tatsache Rechnung getragen, dass ein großer Teil der Seiten im ODP-Datenbestand in den Bereichen „World“ und „Regional“ klassifiziert sind.

Ebenfalls möglich wäre die Implementierung einer *dynamischen Gruppierung* von Benutzern. Diese könnten auf der Basis von Ähnlichkeiten zwischen ihren Benutzermodellen zusammengefasst werden und sich so gegenseitig beeinflussen. Die Literatur zu schon bestehenden Systeme, die mit hierarchischen Benutzermodellen arbeiten, enthält einige Möglichkeiten zur Berechnung von Ähnlichkeitsmetriken. Durch diese dynamische Gruppierung würde das System flexibler, aber auch komplexer und weniger berechenbar.

Mit den nunmehr geschaffenen, umfangreichen Möglichkeiten zur Überwachung von Aktionen im System wäre es in Zukunft auch möglich, *implizite Aktionen* der Benutzer in die Modellierung einfließen zu lassen. Obwohl diese Methode auf den ersten Blick im Vergleich mit expliziten Bewertungen weniger genau wirkt, konnte nachgewisen werden, dass sie meist ebenso gut funktioniert. Implizites Feedback wird in vielen in der Literatur beschriebenen

Systemen verwendet und könnte einen weiteren Schritt zur Verbesserung der Bedienfreundlichkeit bedeuten.

Viel Potenzial bieten auch die *zusätzlichen Verknüpfungen zwischen den Themengebieten* des ODP. Hinweise auf verwandte Themen und das selbe Thema in anderen Sprachen könnten in Zukunft in die Berechnungen von Prospector einfließen und noch mehr semantische Informationen bieten. Auch für die Klassifizierung von Seiten, die nicht im ODP-Datenbestand verzeichnet sind, gibt es in der Literatur Ideen. Ein Ansatz wäre hier, einen Klassifizier-Algorithmus mit dem Inhalt der im ODP verlinkten Seiten zu trainieren und anhand der Snippets bei der Suche eine ungefähre Einordnung zu finden. Dies stellt jedoch einen beträchtlichen technischen Aufwand dar und bedarf tiefergehender Forschung.

Nach der Integration von Prospector in Carrot$^2$ gäbe es noch weitere Möglichkeiten, Aussagen über *nicht im ODP klassifizierte Ergebnisse* zu treffen. Mithilfe von der ursprünglichen Clustering-Algorithmen wäre es beispielsweise denkbar, Ergebnisse im Hintergrund zu Clustern zusammenfassen zu lassen. Verfügen nun eines oder mehrere der klassifizierten Ergebnisse über eine hohe bzw. niedrige berechnete Relevanz, so könnten davon Relevanzen für andere, nicht klassifizierte Ergebnisse im selben Cluster abgeleitet werden. Derartige Kombinationen schon vorhandener Techniken sind in vielen Ausprägungen möglich.

Da Prospector nun in einem durchaus präsentablen Zustand ist, wäre es auch möglich, das System *auf Open Source-Basis zu veröffentlichen* und gemeinsam mit interessierten Freiwilligen weiterzuentwickeln. Auch der Einsatz als Framework in Lehrveranstaltungen wäre möglich. So könnten Studenten ohne großen Aufwand Algorithmen zum Modellieren und Personalisieren von Suchergebnissen implementieren und testen. Durch die nunmehrige Architektur ließen sich die dafür notwendigen Komponenten bereits leicht austauschen.

Auch für die *Evaluierung* von Prospector gibt es einige Ideen. Eine davon ist, nach der nunmehrigen „freien“ Studie im realen Einsatz wieder eine eher *aufgabenorientierte Untersuchung* zu machen. Diese wäre von kürzerer Dauer und in einer kontrollierten Umgebung. Aus diesem Grund wäre der Einsatz von Google als Quellsuchmaschine wieder denkbar, sofern nicht auch der Zugriff über die ursprüngliche API dem Verbot des Umreihens unterliegt. Bei der aktuellen Evaluierung war dies ja nicht möglich, da es bei den vorhandenen Zugängen zur Abfrageschnittstelle eine Beschränkung der täglich abgerufenen Suchergebnisse gab.

Interessant wäre schlussendlich auch, wie viel der in beiden Evaluierungen bemerkten Bevorzugung von Google nur an dem Namen der Suchmaschine und nicht an den konkreten Ergebnissen hängt. Eine Studie hat gezeigt, dass den Ergebnissen unter dem Namen Google auch dann eine gute Qualität zugesprochen wird, wenn sie künstlich falsch gereiht waren [KOS08]. Eine *Blindstudie* könnte hier Aufschlüsse liefern, ob dieser Effekt auch bei der Bewertung von Prospector eine Rolle gespielt hat.

# Literaturverzeichnis


[AM05] Sarabjot Singh Anand and Bamshad Mobasher. Intelligent Techniques for Web Personalization. In Mobasher and Anand [MA05], chapter 1, pages 1–36.

[BKN07] Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors. *The Adaptive Web*. Lecture Notes in Computer Science (Volume 4321). Springer-Verlag, Berlin / Heidelberg, 2007.

[BM07] Peter Brusilovsky and Eva Millan. *User Models for Adaptive Hypermedia and Adaptive Educational Systems*, chapter 1, pages 2–53. In Brusilovsky et al. [BKN07], 2007.

[Bru96a] Peter Brusilovsky. Adaptive Hypermedia: An Attempt to Analyze and Generalize. In *MHVR '94: Selected papers from the First International Conference on Hypermedia, Multimedia, and Virtual Reality: Models, Systems, and Applications*, pages 288–304, London, UK, 1996. Springer-Verlag.

[Bru96b] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2):87–129, July 1996.

[Bru01] Peter Brusilovsky. Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 11(1-2):87–110, 2001.

[Car08] Carrot2 – Open Source Search Results Clustering Engine, July 2008. http://project.carrot2.org/.

[CBY04] Fidel Cacheda and Ricardo A. Baeza-Yates. An Optimistic Model for Searching Web Directories. In Sharon McDonald and John Tait, editors, *Advances in Information Retrieval, Proceedings of the 26th European Conference on IR Research, ECIR 2004*, volume 2997 of *Lecture Notes in Computer Science*, pages 364–377, Berlin / Heidelberg, 2004. Springer-Verlag.

[CS07] Maurice Coyle and Barry Smyth. Supporting intelligent Web search. *ACM Transactions on Internet Technology (TOIT)*, 7(4):20, 2007.

[DMKSH93] Hartmut Dieterich, Uwe Malinowski, Thomas Kühme, and Matthias Schneider-Hufschmidt. State of the Art in Adaptive User Interfaces. In Matthias Schneider-Hufschmidt, Thomas Kühme, and Uwe Malinowski, editors, *Adaptive User Interfaces*, chapter 1, pages 13–48. Elsevier Science Publishers B. V., 1993.

[FLGD87] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, 1987.

[FS94] E.A. Fox and J.A. Shaw. Combination of multiple searches. In *Proceedings of the 2nd Text REtrieval Conference (TREC-2)*, National Institute of Standards and Technology Special Publication 500-215, 1994.

[GM07] Evgeniy Gabrilovich and Shaul Markovitch. Harnessing the Expertise of 70,000 Human Editors: Knowledge-Based Feature Generation for Text Categorization. *The Journal of Machine Learning Research*, 8:2297–2345, 2007.

[Goo08a] Google Directory, July 2008. http://directory.google.com.

[Goo08b] Google Inc. AJAX Search API – Terms of Use, October 2008. http://code.google.com/apis/ajaxsearch/terms.html (Abruf 10.8.2008).

[Goo08c] Google Inc. – The Official Google Blog. More transparency in customized search results, July 2008. http://googleblog.blogspot.com/2008/07/more-transparency-in-customized-search.html (Abruf 13.8.2008).

[Goo08d] Google Inc. – The Official Google Blog. We knew the web was big ..., July 2008. http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html (Abruf 7.8.2008).

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, 1995.

[Gru09] Thomas R. Gruber. Ontology. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*. Springer-Verlag, Berlin / Heidelberg, to be published in 2009.

[GSCM07] Susan Gauch, Mirco Speretta, Aravind Chandramouli, and Alessandro Micarelli. User Profiles for Personalized Information Access. In Brusilovsky et al. [BKN07], chapter 2, pages 54–89.

[Jam03] Anthony Jameson. Adaptive Interfaces and Agents. In Julie A. Jacko and Andrew Sears, editors, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, chapter 15, pages 305–330. Lawrence Erlbaum Associates, Inc., 2003.

[jQu08] jQuery, September 2008. http://www.jquery.com/ (Abruf 2.9.2008).

[JSO08] JSON, September 2008. http://www.json.org/ (Abruf 4.9.2008).

[Kay00] Judy Kay. Stereotypes, Student Models and Scrutability. In *ITS '00: Proceedings of the 5th International Conference on Intelligent Tutoring Systems*, volume 1839 of *Lecture Notes in Computer Series*, pages 19–30, London, UK, June 2000. Springer-Verlag.

[KL05] Kevin Keenoy and Mark Levene. Personalisation of Web Search. In Mobasher and Anand [MA05], chapter 11, pages 201–228.

[KOS08] Mark T. Keane, Maeve O'Brien, and Barry Smyth. Are people biased in their use of search engines? *Communications of the ACM*, 51:49–52, 2008.

[LB07] Jiahui Liu and Larry Birnbaum. Measuring Semantic Similarity between Named Entities by Searching the Web Directory. In *WI '07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 461–465, Washington, DC, USA, 2007. IEEE Computer Society.

[LTM+06] D. Lillis, F. Toolan, A. Mur, L. Peng, R. Collier, and J. Dunnion. Probability-based fusion of information retrieval result sets. *Artificial Intelligence Review*, 25(1-2):179–191, 2006.

[MA05] Bamshad Mobasher and Sarabjot Singh Anand, editors. *Intelligent Techniques for Web Personalization*, volume 3169 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin / Heidelberg, 2005.

[MGSG07] Alessandro Micarelli, Fabio Gasparetti, Filippo Sciarrone, and Susan Gauch. Personalized Search on the World Wide Web. In Brusilovsky et al. [BKN07], chapter 6, pages 195–230.

[ODP08] Open Directory Project (ODP), July 2008. http://www.dmoz.org (Abruf 7.8.2008).

[Ope07] Open Directory Project (ODP). What Topic Do You Care About?, October 2007. http://blog.dmoz.org/2007/10/24/what-topic-do-you-care-about/ (11. 8. 2008).

[Ope08] OpenSearch Specification 1.1 Draft 3, September 2008. http://www.opensearch.org/Specifications/OpenSearch/1.1.

[OW07] Stanislaw Osinski and Dawid Weiss. Introducing Usability Practices to OSS: The Insiders' Experience. In Joseph Feller, Brian Fitzgerald, Walt Scacchi, and Alberto Sillitti, editors, *Open Source Development, Adoption and Innovation, Proceedings of the Third International Conference on Open Source Systems*, volume 234 of *IFIP International Federation for Information Processing*, pages 313–318, Boston, 2007. Springer-Verlag.

[Per08] Saverio Perugini. Symbolic links in the Open Directory Project. *Information Processing and Management: an International Journal*, 44(2):910–930, 2008.

[PKSvV08] Alexandros Paramythis, Florian König, Christian Schwendtner, and Lex van Velsen. Using thematic ontologies for user- and group-based adaptive personalization in web searching. In *Proceedings of the 6th International Workshop on Adaptive Multimedia Retrieval (AMR 2008)*, Lecture Notes in Computer Science. Springer-Verlag, 2008. In Redaktion.

[PPP+04] Dimitrios Pierrakos, Georgios Paliouras, Christos Papatheodorou, Vangelis Karkaletsis, and Marios Dikaiakos. Web Community Directories: A New Approach to Web Personalization. In *Web Mining: From Web to Semantic Web, Proceedings of the First European Web Mining Forum, EWMF 2003*, volume 3209 of *Lecture Notes in Computer Science*, pages 113–129, Berlin / Heidelberg, 2004. Springer-Verlag.

[RS03] M. Elena Renda and Umberto Straccia. Web metasearch: rank vs. score based rank aggregation methods. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 841–846, New York, NY, USA, 2003. ACM.

[SB06] Barry Smyth and Evelyn Balfe. Anonymous personalization in collaborative web search. *Information Retrieval*, 9(2):165–190, 2006.

[SB08] Anne Sunikka and Johanna Bragge. What, Who and Where: Insights into Personalization. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 283, Washington, DC, USA, 2008. IEEE Computer Society.

[SKP06] Christian Schwendtner, Florian König, and Alexandros Paramythis. Prospector: An adaptive front-end to the Google search engine. In Klaus-Dieter Althoff and Martin Schaaf, editors, *Proceedings of the 14th Workshop on Adaptivity and User Modeling in Interactive Systems (ABIS 2006)*, volume 1/2006 of *Hildesheimer Informatik-Berichte*, pages 56–61. University of Hildesheim, Institute of Computer Science, 2006.

[SKSC06] Barry Smith, Waclaw Kusnierczyk, Daniel Schober, and Werner Ceusters. Towards a Reference Terminology for Ontology Research and Development in the Biomedical Domain. In Olivier Bodenreider, editor, *KR-MED 2006, Formal Biomedical Knowledge Representation, Proceedings of the Second International Workshop on Formal Biomedical Knowledge Representation: „Biomedical Ontology in Action" (KR-MED 2006)*, volume 222 of *CEUR Workshop Proceedings*, Baltimore, Maryland, USA, November 2006. CEUR-WS.org.

[Smi04] Barry Smith. Beyond Concepts: Ontology as Reality Representation. In Achille Varzi and Laure Vieu, editors, *Proceedings of FOIS 2004. International Conference on Formal Ontology and Information Systems*, Turin, Italy, November 2004. IOS Press.

[SPAS98] Constantine Stephanidis, Alexandros Paramythis, Demosthenes Akoumianakis, and Michael Sfyrakis. Self-Adapting Web-based Systems: Towards Universal Accessibility. In Constantine Stephanidis and Annika Waern, editors, *UI4ALL-98: Proceedings of the 4th ERCIM Workshop on User Interfaces for All*, Stockholm, Sweden, October 1998.

[Sul98] Danny Sullivan. NewHoo: Yahoo Built By The Masses. Searchenginewatch.com, July 1998. http://searchenginewatch.com/showPage.html?page=2166381 (Abruf 11. 8. 2008).

[SW01] Barry Smith and Christopher Welty. Ontology – towards a new synthesis. In *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, pages iii–ix, New York, NY, USA, 2001. ACM.

[SW03] Jerzy Stefanowski and Dawid Weiss. Carrot2 and Language Properties in Web Search Results Clustering. In Ernestina Menasalvas Ruiz, Javier Segovia, and Piotr S. Szczepaniak, editors, *Advances in Web Intelligence, Proceedings of the First International Atlantic Web Intelligence Conference, AWIC 2003*, volume 2663 of *Lecture Notes in Computer Science*, pages 240–249, Madrid, Spain, May 2003. Springer-Verlag.

[TB87] Robert Trevellyan and Dermot P. Browne. A self-regulating adaptive system. *SIGCHI Bulletin*, 18(4):103–107, 1987.

[TM02] F. Tanudjaja and L. Mui. Persona: A Contextualized and Personalized Web Search. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, volume 3, page 67, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[vVPvdG08] Lex van Velsen, Alexandros Paramythis, and Thea van der Geest. Advances in Intelligent Search. User-centered formative Evaluation of a Personalized Internet Meta-search Engine. In Vorbereitung, Oktober 2008.

[Wor04] World Wide Web Consortium. RDF/XML Syntax Specification, February 2004. http://www.w3.org/TR/rdf-syntax-grammar/ (Abruf 27.10.2008).

# Annex A: Datenschutzerklärung

Prospector ist eine personalisierte Suchmaschine. Um die Reihung der Suchergebnisse für den jeweiligen Benutzer personalisieren zu können, müssen Informationen zu dessen Interessen vorliegen. Diese werden durch Bewertungen des Benutzers und explizite Angaben der Interessen im Profil in Erfahrung gebracht.

Um für die Studie Daten zum Suchverhalten der Benutzer zu gewinnen, werden Ihre Aktionen im System mitprotokolliert. Mit Ihrem Einverständnis, an der Studie teilzunehmen, stimmen Sie der Verwendung dieser Daten entsprechend der Angaben in „Datenverwendung“ weiter unten zu.

Wir haben spezifische Vorkehrungen getroffen, sodass die gesammelten Informationen nicht verwendet werden kann, um Sie persönlich zu identifizieren. Diese Maßnahmen umfassen (sind aber nicht beschränkt auf) folgende Eckpunkte:

- Wir speichern keine Information (wie beispielsweise Ihre IP-Adresse), mit der Ihre Identität in Erfahrung gebracht werden kann.
- Beim Registrieren werden Benutzername und Passwort nur verschlüsselt gespeichert. Dies verhindert, dass die Benutzerdaten mit dem Benutzernamen verknüpft werden.

Unsere Richtlinie sieht außerdem ausdrücklich vor, dass während der Sammlung und Auswertung der Daten kein Versuch unternommen werden wird, persönliche Informationen der Teilnehmer zu extrahieren oder abzuleiten.

**Datenverwendung** Die gesammelten Daten werden nur im Rahmen der Studie, an der Sie teilnehmen, verwendet, um statistische und andere Arten von Analysen zu machen. Ohne Ihr ausdrückliches Einverständnis werden die Daten zu keinem weiteren Zweck verwendet.

**Datenweitergabe** Die gesammelten Daten werden unter keinen Umständen an Personen oder Organisationen weitergegeben, die nicht direkt in dieser Studie involviert sind.

**Datensicherheit** Die gesammelten Daten werden in solcher Weise gespeichert, dass ein vernünftiger Schutz gegen unautorisierten Zugriff, Veränderung und Löschen gegeben ist. Die Daten werden, gemäß dem aktuellen Stand der Technik, ebenso derart übertragen (im Falle einer Datenübertragung), sodass Sicherheit gegen unautorisierten Zugriff, Veränderung und Löschen besteht. Nur autorisierte Personen, die direkt mit der Evaluierung betraut sind, werden Zugriff auf die Daten haben; diese Personen sind and die hier dargelegten Bestimmungen gebunden.

**Impressum** Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM), Johannes Kepler Universität Linz, Altenbergerstraße 69, 4040 Linz, Österreich

# Annex B: Fragebögen

Im Folgenden finden sich die Inhalte der drei verwendeten Fragebögen. Zu jeder Frage werden folgende Angaben gemacht:

- *Nummer der Frage*: durchlaufend, von 1 beginnend, mit Buchstaben für Teilfragen.
- *Fragetext der Hauptfrage*
- *Fragetext der Unterfrage*: bezieht sich auf die Hauptfrage und spezifiziert diese näher.
- *Typ der erwarteten Antwort*: hierbei wird unterschieden zwischen
  - *Zahl*: numerische Eingabe.
  - *Kurztext*: einzeilige Eingabemöglichkeit für Text.
  - *Langtext*: mehrzeilige Eingabemöglichkeit für Text.
  - *Auswahl*: aus mehreren Optionen kann eine gewählt werden.
- *Mögliche Werte*: entsprechend dem Typ der erwarteten Antwort.

Die meisten Fragen waren verpflichtend zu beantworten. Lediglich die kursiv dargestellten konnten von den Teilnehmern übersprungen werden.

# Fragebogen 1

| # | Frage | Unterfrage | Typ | Mögliche Werte |
|---|---|---|---|---|
| 1 | Geschlecht | | Auswahl | männlich, weiblich |
| 2 | Alter | | Zahl | 18–100 |
| 3 | Höchste abgeschlossene Bildung | | Auswahl | Grundschule, Mittelschule, College, Studium (Universität, FH), Postgraduales Studium |
| 4 | *Studienfach bzw. Studienfächer (falls zutreffend)* | | *Kurztext* | |
| 5 | Wie gut können Sie einen Computer und Standard-Programme wie Microsoft Word und Internet Explorer bedienen? | | Auswahl | 7-stufig („sehr schlecht“ bis „sehr gut“) |
| 6 | Wie oft erledigen Sie folgende Tätigkeiten im Internet? | E-Mails senden und empfangen | Auswahl | täglich, nahezu täglich, ein- oder zweimal pro Woche, ein- oder zweimal pro Monat, weniger als einmal pro Monat, nie |
| | | Im Web nach Informationen suchen | | |
| | | Online-Banking | | |
| | | Chatten (z. B. mit MSN Messenger) | | |
| | | Telephonieren (z. B. mit Skype) | | |
| | | Filme, TV oder Musik online ansehen/anhören | | |
| | | Nachrichten lesen (Politik, Sport, . . . ) | | |
| | | Software herunterladen | | |
| | | Amtliche Tätigkeiten (e-Government) | | |
| | | Mehrbenutzerspiele (wie World of Warcraft) spielen oder in virtuellen Welten (wie Second Life) agieren | | |
| 7 | Verwenden Sie öfters eines dieser Systeme und ihre Personalisierungsfunktionen? | Amazon mit seinen personalisierten Buch- und Musikempfehlungen | Auswahl | ja, nein, ich kenne dieses System nicht |
| | | Last.fm mit seinen personalisierten Playlists | | |
| | | Movielens mit seinen personalisierten Filmempfehlungen | | |
| | | Die personalisierte iGoogle-Homepage | | |
| 7a | *Gibt es andere Systeme mit Personalisierungsfunktionen, die Sie oft nützen?* | | *Langtext* | |

**Fragebogen 1 (fortgesetzt)**

| # | Frage | Unterfrage | Typ | Mögliche Werte |
|---|---|---|---|---|
| 8 | Welche Suchmaschinen verwenden Sie bevorzugt? | | Kurztext | |
| 9 | Wie oft verwenden Sie eine Suchmaschine? | | Auswahl | mehrmals pro Woche, einmal täglich, zwei bis fünf mal täglich, fünf bis 15 mal täglich, mehr als 15 mal täglich |
| 10 | Falls Ihre Suchmaschine personalisierte Suchfunktionen bietet (z. B. Google Personalized Search), verwenden Sie diese? | | Auswahl | ja, nein, ich wusste nicht, dass es das gibt |
| 11 | *Verwenden Sie andere Suchmaschinen außer der oben genannten bevorzugten, und wenn ja, warum?* | | *Langtext* | |
| 12 | Wie stehen Sie zu den folgenden Aussagen in Bezug auf Ihre bevorzugte Suchmaschine? | Mit dieser Suchmaschine finde ich schnell Informationen | Auswahl | 7-stufig („stimme gar nicht zu“ bis „stimme voll zu“) |
| | | Die Suchmaschine ist effizient | | |
| | | Die Suchmaschine hilft mir, sinnvolle Informationen zu finden | | |
| 13 | *Haben Sie Bedenken bezüglich Ihrer Privatsphäre, wenn Sie das Internet benutzen, und wenn ja warum?* | | *Langtext* | |
| 14 | Können Sie in ein paar Worten die Erwartungen an Prospector beschreiben, basierend auf dem, was sie bis jetzt über das System gelesen haben? | | Langtext | |
| 15 | *In Prospector gibt es die Möglichkeit, das Modell, welches das Systems von Ihren Suchinteressen hat, zu betrachten und zu verändern. Erachten Sie das als nützlich?* | | *Langtext* | |
| 16 | *Wie können Sie sich vorstellen, Ihr persönliches Modell zu betrachten und zu verändern (wenn überhaupt gewünscht)?* | | *Langtext* | |

## Fragebogen 2

| # | Frage | Unterfrage | Typ | Mögliche Werte |
|---|---|---|---|---|
| 1 | Wann haben Sie begonnen, Prospector zu benutzen? | | Datum | |
| 2 | Seit Sie begonnen haben, Prospector zu benützen, welchen Prozentsatz Ihrer Suchen (muss nicht exakt sein) haben Sie ... | ... nur mit Prospector gemacht? | Auswahl | 0%–100% in 10%-Schritten |
| | | ... mit Prospector und Ihrer Standard-Suchmaschine gemacht? | | |
| | | ... nur mit Ihrer Standardsuchmaschine gemacht? | | |
| 3 | Wie stehen Sie zu den folgenden Aussagen in Bezug auf Prospector? | Prospector hilft mir, sinnvolle Informationen zu finden | Auswahl | 7-stufig („stimme gar nicht zu“ bis „stimme voll zu“) |
| | | Mit Prospector finde ich schnell Informationen | | |
| | | Prospector ist effizient | | |
| 4 | Bei Suchanfragen, die Sie sowohl an Prospector als auch die Suchmaschine, die Sie ansonsten verwenden, gestellt haben – welche Suchmaschine lieferte das/die Ergebnis(se), das/die Sie suchten weiter vorne in der Liste? | | Langtext | |
| 5 | Was war Ihre schlechteste Erfahrung mit Prospector, seit Sie es benutzen? | | Langtext | |
| 6 | Wann war Prospector besonders hilfreich? | | Langtext | |
| 7 | Rückblickend auf Ihre bisherigen Erfahrungen, welches System bevorzugen Sie? Ihre bisherige Suchmaschine oder Prospector? Können Sie erklären warum? | | Langtext | |
| 8 | Wenn Sie das Modell Ihrer persönlichen Interessen in Prospector betrachten, ist Ihnen klar, was angezeigt wird? | | Langtext | |
| 9 | Wenn Sie Ihr Benutzermodell betrachten, spiegelt es Ihre Interessen und Suchpräferenzen wider? | | Langtext | |
| 10 | *Haben Sie irgendwelche Änderungen in Ihrem Benutzermodell vorgenommen? Wenn ja, welche Arten von Änderungen und warum?* | | *Langtext* | |

## Fragebogen 3

| # | Frage | Unterfrage | Typ | Mögliche Werte |
|---|---|---|---|---|
| 1 | Seit Sie den vorherigen Fragebogen ausgefüllt haben, welchen Prozentsatz Ihrer Suchen (muss nicht exakt sein) haben Sie ... | ... nur mit Prospector gemacht? | Auswahl | 0%–100% in 10%-Schritten |
| | | ... mit Prospector und Ihrer Standard-Suchmaschine gemacht? | | |
| | | ... nur mit Ihrer Standardsuchmaschine gemacht? | | |
| 2 | Wie stehen Sie zu den folgenden Aussagen in Bezug auf Prospector? | Prospector hilft mir, sinnvolle Informationen zu finden | Auswahl | 7-stufig („stimme gar nicht zu“ bis „stimme voll zu“) |
| | | Mit Prospector finde ich schnell Informationen | | |
| | | Prospector ist effizient | | |
| 3 | Bei Suchanfragen, die Sie sowohl an Prospector als auch die Suchmaschine, die Sie ansonsten verwenden, gestellt haben – welche Suchmaschine lieferte das/die Ergebnis(se), das/die Sie suchten weiter vorne in der Liste? | | Langtext | |
| 4 | Was war Ihre schlechteste Erfahrung mit Prospector, seit Sie den vorherigen Fragebogen beantwortet haben? | | Langtext | |
| 5 | Wann war Prospector besonders hilfreich, seit Sie den vorherigen Fragebogen beantwortet haben? | | Langtext | |
| 6 | Rückblickend auf Ihre Erfahrungen, welches System bevorzugen Sie? Ihre bisherige Suchmaschine oder Prospector? Können Sie erklären warum? | | Langtext | |
| 7 | Wenn Sie Ihr Benutzermodell betrachten, spiegelt es Ihre Interessen und Suchpräferenzen wider? | | Langtext | |
| 8 | *Haben Sie irgendwelche Änderungen in Ihrem Benutzermodell vorgenommen? Wenn ja, welche Arten von Änderungen und warum?* | | *Langtext* | |

**Fragebogen 3 (fortgesetzt)**

| | | | | |
|---|---|---|---|---|
| 9 | Sind Sie der Meinung, dass Prospector berechenbar arbeitet? | | Langtext | |
| 10 | Haben Sie den Eindruck, zu verstehen, wie Prospector arbeitet? | | Langtext | |
| 11 | Gibt es bestimmte Teile oder Funktionen von Prospector, die Sie als schwer zu verstehen empfinden? Wenn ja, welche? | | Langtext | |
| 12 | Sind Sie der Meinung, dass Sie die Kontrolle haben, wenn Sie Prospector benutzen? | | Langtext | |
| 13 | Denken Sie, dass die Verwendung von Prospector für Sie viel Aufwand bedeutet? | | Langtext | |
| 14 | Sind Sie der Meinung, dass die Verwendung von Prospector Ihre Privatsphäre beeinträchtigt? | | Langtext | |
| 15 | Sind Sie der Meinung, dass die Verwendung von Prospector die Vielfalt Ihrer Suchergebnisse herabsetzt? | | Langtext | |
| 16 | Sind Sie der Meinung, dass Prospector es schafft, für Sie relevante Suchergebnisse zu liefern? | | Langtext | |

# Annex C: Curriculum Vitae

## Persönliche Daten

| | |
|---|---|
| *Name* | Florian König |
| *Akademischer Grad* | Bakk. techn. |
| *Geburtsdatum/-ort* | 27. April 1983 in Kirchdorf a. d. Krems |
| *Wohnort* | Pulvermühlstraße 41/417, 4040 Linz |
| *Telefonnummer* | +43 (699) 11 0 11 926 |
| *e-Mail-Adresse* | koenig@fim.uni-linz.ac.at |
| *Familienstand* | ledig |

## Ausbildung

| | |
|---|---|
| *2007 –* | Masterstudium Informatik an der Johannes Kepler Universität Linz |
| *2002 – 2007* | Bakkalaureatsstudium Informatik an der JKU Linz |
| *1993 – 2001* | Humanistisches Stiftsgymnasium Kremsmünster, OÖ |

## Sprachkenntnisse

| | |
|---|---|
| *Englisch* | fließend, mit regelmäßiger Verwendung in Wort und Schrift |
| *Französisch* | Grundkenntnisse |
| *Russisch* | gute Kenntnisse mit dem Lernziel eines längeren Auslandsaufenthalts |

## Fachkenntnisse

| | |
|---|---|
| *Systeme* | Microsoft Windows, GNU/Linux, Mac OS X |
| *Formale Sprachen* | Java(script), C, C#, Prolog, SQL, XML, XSLT, HTML, CSS |
| *Frameworks/Bibliotheken* | Swing, .NET, Servlet/JSP, Hibernate, jQuery |
| *Produkte* | Eclipse, Typo3, Apache (Tomcat), MySQL, Microsoft Office, Photoshop CS3, InDesign CS3, Alfresco, Microsoft SharePoint, Microsoft SQL Server 2005 Suite |

## Berufserfahrung

| | |
|---|---|
| *Jänner 2007 –* | Wissenschaftlicher Mitarbeiter am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM): Weiterentwicklung von Prospector im Rahmen des ASCOLLA-Projekts |
| *Okt. 2006 – Juni 2007* | Wissenschaftlicher Mitarbeiter am FIM: Forschung zur Zukunft der Mensch-Maschine-Kommunikation im Bereich e-Government (im Auftrag des Fabasoft Institute of Technology) |
| *Sep. 2006 – Aug. 2007* | Freier Dienstnehmer der SKF Österreich AG: Entwicklung einer Data-Warehousing Anwendung mit OLAP u. Reporting |
| *Apr. 2004 – Juni 2006* | Wissenschaftlicher Mitarbeiter am FIM: Weiterentwicklung eines Editors zur Erstellung von Lernpaketen (jCAPT) |
| *2003 – 2007* | Tutor für die KV Propädeutikum und die Vorlesung Betriebssysteme im Bakkalaureatsstudium Informatik |
| *März 2003 – Dez. 2003* | Wissenschaftlicher Mitarbeiter am FIM: Unterstützung beim Erstellen von Lehrmaterialien und bei Online-Kursen |
| *Juli 2001 – Dez. 2001* | Praktikum bei der Firma Infoniqa Informationstechnik GmbH: Verfeinerung der Portierung der Hauptprodukte auf Linux und Entwicklung eines Bandbreitenbegrenzers |
| *Aug. 2000 – Sept. 2000* | Praktikum bei Infoniqa: Portierung von Produkten auf Linux und begleitende Forschung für die Fachbereichsarbeit |

## Publikationen

Johannes Schönböck, Florian König, Gabriele Kotsis, Dominik Gruber, Emre Zaim, Albrecht Schmidt. *MirrorBoard – An Interactive Billboard.* In M. Herczeg and M. C. Kindsmüller, editors, Proceedings of the Mensch & Computer 2008: Viel Mehr Interaktion, p. 217–226. Oldenbourg Verlag, München, 2008.

Alexandros Paramythis, Florian König, Christian Schwendtner, and Lex van Velsen. *Using thematic ontologies for user- and group-based adaptive personalization in web searching.* In Proceedings of the 6th International Workshop on Adaptive Multimedia Retrieval (AMR 2008), Lecture Notes in Computer Science. Springer-Verlag, 2008. In Redaktion.

Christian Schwendtner, Florian König, and Alexandros Paramythis. *Prospector: An adaptive front-end to the Google search engine.* In Klaus-Dieter Althoff and Martin Schaaf, editors, Proceedings of the 14th Workshop on Adaptivity and User Modeling in Interactive Systems (ABIS 2006), volume 1/2006 of Hildesheimer Informatik-Berichte, p. 56–61. University of Hildesheim, Institute of Computer Science, 2006.

Florian König. *jCAPT – Lernpakete nach Maß*, Tagungsband zum Usability Day IV: Informationen nutzbar machen. FH Dornbirn, 2006.

## Fortbildungen

| | |
|---|---|
| *Sep. 2007* | Seminar Zeitungsdesign der OÖ Journalistenakademie |
| *Sep. 2006 – Nov. 2006* | 12-tägiger Journalismuskurs der OÖ Journalistenakademie |
| *Juli 2006* | 4-wöchiger Intensivsprachkurs in Russland (Nizhnij Novgorod) mit den Themen Grammatik, Kommunikation, Kultur |
| *Mai 2006 – Nov. 2006* | 10-tägige praxisorientierte Grundausbildung Printjournalismus bei der Katholischen Medien Akademie (KMA) |
| *März 2006 –* | Besuch von Uni-Kursen zur Aneignung rechtlicher und wirtschaftlicher Grundlagen<br>• Öffentliches Recht, Privatrecht, e-Business-Recht<br>• Finanzierung, Investition, Steuern<br>• Individuum, Gruppe, Organisation<br>• Buchhaltung, Kostenrechnung, Unternehmensgründung |
| *2003 –* | Absolvierung persönlichkeitsbildender Seminare zu<br>• Teamfähigkeit und Persönlichkeitsstruktur<br>• Flipchart, Bewerbung und Assessment Center<br>• Zielklärung und -erreichung in Leben und Beruf |

## Zusatzqualifikationen

| | |
|---|---|
| *Juli 2006 – Juni 2007* | Öffentlichkeitsreferent für das 11-köpfige Präsidium von Österreichs größtem Akademikerverband mit den Aufgabenbereichen<br>• Homepage (Aufbau, Weiterentwicklung und Wartung)<br>• Publikationen (Zeitung, Festschrift, Einladungen)<br>• Pressearbeit (Moderationen, Ankündigungen, ...)<br>• interne Kommunikation (Forum, Mail-System, ...) |
| *Juli 2003 – Juni 2007* | Interessensvertretung für rund 1.200 Informatik-Studenten als ehrenamtlicher Studierendenvertreter |
| *Jan. 2002 – Aug. 2002* | Ableistung des Präsenzdienstes in der Stabskompanie des Panzerstabsbataillon 4 in Ebelsberg/Linz |

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplom- bzw. Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Des weiteren versichere ich, dass ich diese Diplom- bzw. Magisterarbeit weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Ort und Datum Florian König