Integration einer Volltextsuchmaschine in Meta-Akad

Projektarbeit
Universität Kaiserslautern

Jochen Tuchbreiter

30. August 2003

Inhaltsverzeichnis

1 Suchmaschinen

1.1 Allgemein

Als Suchmaschinen bezeichnet man Dienste, welche anhand von vorgegebenen Schlüsselbegriffen zugehörige Inhalte ermitteln. Von Recherchehilfen in Bibliotheken bis hin zu Helfern bei der Suche nach Internetinhalten existiert eine Vielfalt von verschiedenen Arten. Sie werden durch drei wesentliche Faktoren bestimmt:

- Erfassungsvorgang
  Bevor eine Suchmaschine Antworten auf Anfragen herausgeben kann, muss sie zunächst über eine Datenbank der zu durchsuchenden Dokumente verfügen. Der Aufbau dieser Datenbank kann über automatisches, halbautomatisches oder manuelles Indizieren erfolgen.
  Internet-Suchmaschinen wie Google verwenden vollautomatische Indizierungsprozesse, welche - einmal auf eine Startseite angesetzt - diese Indizieren und anschließend allen Links auf dieser Seite folgen. Dies hat den Vorteil, dass solche Suchmaschinen eine enorm große Suchdatenbank aufbauen und aktuell halten können. Im Gegensatz dazu verwenden Internet-Verzeichnisdienste wie Yahoo! einen manuellen Prozess: Hier trägt eine Redaktion die Inhalte von Hand in ein Verzeichnis ein und bewertet diese.

- Erfasste Inhalte
  Typischerweise erfassen Suchmaschinen Text in verschiedenen Formaten. Es existieren jedoch auch Suchmaschinen zur Suche nach Bildern wie z.B. http://image.google.com oder Audiodateien wie
  http://speechbot.research.compaq.com.
  Neben dem Medientyp ist entscheidend ob den erfassten Inhalten Meta-Daten hinzugefügt werden, wie beispielsweise Informationen über die Qualität des Dokuments. Im Extremfall existieren für einen Inhalt ausschließlich Meta-Daten, beispielsweise wenn die Suchmaschine auf den Inhalt selbst keinen Zugriff hat wie dies bei Printmedien der Fall ist.

- Ermittlung der Suchergebnisse
  Die Art und Weise wie der erfasste Inhalt, beziehungsweise die mit ihm verknüpften Metadaten, durchsucht werden kann variiert stark: So kann beispielsweise die Suche nach Text auf ganze Wörter beschränkt sein, es kann möglich sein bestimmte Wortabfolgen zu suchen oder Suchergebnisse mit bestimmten Schlüsselwörtern von den Ergebnissen auszuschließen.
  Eine wichtige Rolle spielt die Gewichtung der Ergebnisse: Speziell bei Internet-Suchmaschinen wird oft eine große Anzahl von Treffern erzielt. Hier kommt es darauf an, die Suchergebnisse nach ihrer Relevanz zu beurteilen, um den Benutzer die wichtigsten Ergebnisse zuerst zurückliefern zu können. Neben einfachen Bewertungsmöglichkeiten wie die Anzahl der Vorkommen der Suchbegriffe im indizieren Inhalt, existieren speziell für Internet-Suchmaschinen eine Reihe von Verfahren: So kann zum Beispiel die Anzahl der Links anderer indizierter Dokumente auf den Treffer Auskunft über seine Wichtigkeit geben. Eine andere Idee ist die Häufigkeit, mit der Nutzer der Suchmaschine auf bestimmte Treffer klicken, in die Bewertung mit einfließen zu lassen.

In dieser Projektarbeit wird eine Suchmaschine, welche textuelle Inhalte mittels eines automatischen Indizierungsprozesses erfasst, verwendet.

1.2 Aspseek

Für die Durchführung der Volltextsuche in Meta-Akad wurde die Nutzung der Suchmaschine Aspseek [2] vorgeschrieben. Diese besteht aus drei Hauptkomponenten:

- Indizierungsdienst “index”
- Suchdienst “searchd”
- Suchclients: “s.cgi” (CGI-Script) und “mod_aspseek” (Apache-Modul)

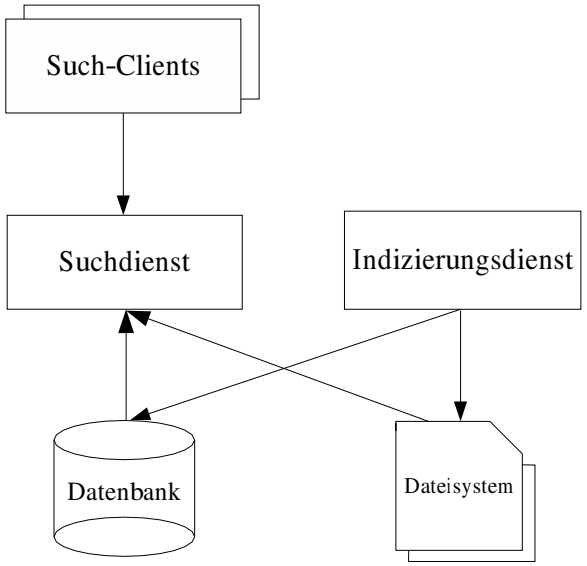

Abbildung 1: Struktur von Aspseek

Sowohl der Indizierungsdienst als auch der Suchdienst arbeiten auf einer gemeinsamen Datenbasis. Diese besteht zum einen aus einer Datenbank, zum anderen aus einer Menge von binären Dateien. Die Datenbank dient zwei Zwecken:

1. Halten von Verwaltungsinformationen wie einer Liste von indizierten URLs mit dem Zeitpunkt ihrer letzten Indizierung.
2. Halten einer Liste aller in den indizierten Dokumenten vorkommender Wörter mit Verweisen auf die zugehörigen Dokumente.

In den Binärdateien werden die Inhalte selbst in komprimierter Form abgelegt.

Weiterhin werden sie auch für den Fall, dass sehr viele Dokumente ein bestimmtes indiziertes Wort enthalten, für die Speicherung dieser Liste verwendet. Hierbei wird dann in der Datenbank, in der Tabelle der Wörter, ein Verweis auf die Datei gespeichert. Der Grund für dieses Vorgehen liegt darin, dass die Dokumentlisten in der Datenbank aus Performanz-Gründen nicht normalisiert abgelegt sind, sondern in ein einziges Feld geschrieben werden. Für lange Listen reicht dessen Grösse jedoch nicht aus.

1.2.1 Der Indizierungsdienst

Der Indizierungsdienst sorgt sowohl für die Aufnahme neuer Inhalte in die Suchdatenbank, als auch für die Aktualisierung selbiger. Er ist in der Lage auf Text- und HTML-Dokumenten zu arbeiten. Stößt der Indizierungsdienst auf andere Formate, so kann er externe Konvertierungsprogramme aufrufen, welche diese in Text oder HTML übersetzen. Weiterhin kann der Indizierungsdienst Verweisen in HTML-Dokumenten folgen und somit selbstständig weitere zu indizierende Inhalte ermitteln. Aufgrund dieser Funktionalität ist es ihm möglich, ausgehend von einer einzigen Startseite, komplette Websites zu indizieren. Dabei kann die maximale Tiefe der Verweisverfolgung ebenso wie eine Positivliste von zur Indizierung freigegebenen Domains konfiguriert werden.

Bei der Suche nach Änderungen im Datenbestand wird versucht ohne eine erneute Übertragung der Dokumente auszukommen: Der Suchdienst fordert über eine HTTP “HEAD” Anfrage vom Webserver zunächst nur die Kopfzeilen-Informationen über das Dokument an und vergleicht dann die “Last modified” Angabe der Antwort mit dem Zeitpunkt der letzten Indizierung. Wurde die Datei laut der Antwort des Webservers geändert, so lädt er das Dokument herunter, bildet eine Prüfsumme darüber und vergleicht diese mit der zuvor gebildeten Prüfsumme der indizierten Version. Nur wenn auch die Prüfsummen nicht übereinstimmen wird das Dokument als geändert erkannt und erneut indiziert.

1.2.2 Der Suchdienst

Der Suchdienst bildet das Backend für die Durchführung von Suchanfragen auf dem Datenbestand. Er ist als Daemon-Prozess realisiert und wird über einen Socket gesteuert. Ein Such-Client baut eine Verbindung zu ihm auf und überträgt seine Anfrage in einem proprietären Format. Nach Durchführung der Suche sendet der Suchdienst das Ergebnis und schließt die Verbindung daraufhin. Die Suche selbst wird mit Hilfe der in der Datenbank vorhandenen Wortlisten durchgeführt: Dort sind alle in indizierten Dokumenten vorkommenden Wörter mit den URLs der Dokumente abgelegt. Nach dem Feststellen der Treffer holt der Suchdienst Extrakte der Dokumente an den entsprechenden Stellen. Da die Dateien, welche diese Extrakte enthalten, in einem komprimierten Format vorliegen, müssen sie erst dekomprimiert werden, was einen großen Teil der zur Bearbeitung der Suchanfrage insgesamt benötigten Zeit ausmachen kann. Die maximale Anzahl von Treffern, welche pro Anfrage ermittelt wird, ist konfigurierbar.

1.2.3 Such-Clients

Da der Suchdienst selbst nur über ein proprietäres, nicht dokumentiertes Format anzusprechend ist, liefert Aspseek zwei Such-Clients mit: Ein CGI-Programm und ein Plugin für den Webserver Apache. Abgesehen von der Anbindung an ihre Umwelt gleichen sich beide Clients. Sie sind dafür ausgelegt eine Abfrage der indizierten Daten über einen Web-Browser zu ermöglichen. Das von Ihnen verwendete Layout ist über eine Konfigurationsdatei einstellbar.

2 Integration in Meta-Akad

2.1 Meta-Akad

2.1.1 Anforderungen und Ziele

Das Projekt Meta-Akad hat das Ziel, Lernenden und Lehrenden den möglichst einfachen, umfassenden und schnellen Zugriff auf Lehrmaterial zu ermöglichen. Dabei werden verschiedene, über die Aspekte einer reinen Internet-Suchmaschine hinausgehende Aspekte berücksichtigt: Neben dem Aufbau einer umfangreichen und repräsentativen Sammlung von Lerndokumenten sollen diese mittels bibliothekarischer Methoden erschlossen und mit umfangreichen Metadaten, wie beispielsweise einer inhaltlichen Einordnung, versehen werden. Um dem Problem der fraglichen Qualität von Dokumenten aus dem Internet gerecht zu werden, bietet Meta-Akad die Möglichkeit diese durch Beguchtachtungsverfahren sicherzustellen. Aufgrund dieses Mehrwerts versteht sich das Projekt als virtuelle, über das Internet erreichbare Bibliothek.

Der Zugriff auf die erfassten Dokumente ist durch eine Web-basierte Schnittstelle realisiert: Diese soll sowohl die Möglichkeit einer Suche durch Angabe von Schlüsselwörtern, als auch das Blättern in der Dokumentsammlung erlauben. Eine Suche nach Schlüsselwörtern soll neben den Meta-Daten auch den gesamten textuellen Inhalt der Dokumente betreffen. Die Integration der Volltextsuche in den bereits vorhandenen Meta-Daten Suchvorgang ist das Kernthema dieser Projektarbeit.

2.1.2 Realisierung und Architektur

Meta-Akad ist als Java 2 Enterprise Edition (J2EE) Anwendung implementiert. Die Verwendung dieses Rahmenkonzepts erlaubt die saubere Umsetzung der komplexen Anforderungen, während das System leicht erweiterbar und damit zukunftssicher ist. Die mit Meta-Akad erfassten Daten werden in einer relationalen Datenbank abgelegt und können als XML-Dokumente abgerufen werden.

Entsprechend der allgemeinen für J2EE empfohlenen Architektur teilt sich Meta-Akad in drei Schichten auf:

1. Präsentationslogik (“Web-Tier”)
   Die Präsentationslogik übernimmt die Aufgabe der Darstellung. Im Falle der Komponente zum Durchsuchen des Datenbestandes (“Web-Search”), der Administrationskomponente (“Web-Maintain”) und der unterstützenden Dienste (“Web-Services”) geschieht diese Darstellung Web-basiert. Die zur Erfassung von Dokumenten dienende Komponente (“Indexing Tool”) ist als Java Swing Anwendung realisiert (????).

2. Anwendungslogik (“EJB-Tier”)
   Die Anwendungslogik implementiert die Hauptfunktionen des Systems: Neben der Anfrageverarbeitung (“QueryEngine”) und der dazugehörigen Ergebnisverarbeitung (“Result Set Processor”), werden hier auch Funktionen zum XML Dokumentmanagement bereitgestellt (“XML Processor”, “XML/R Mappingsupport”).

3. Datenhaltungskomponente (“EIS-Tier”)
   Diese Komponente ist für die Bereitstellung von Daten für die höheren

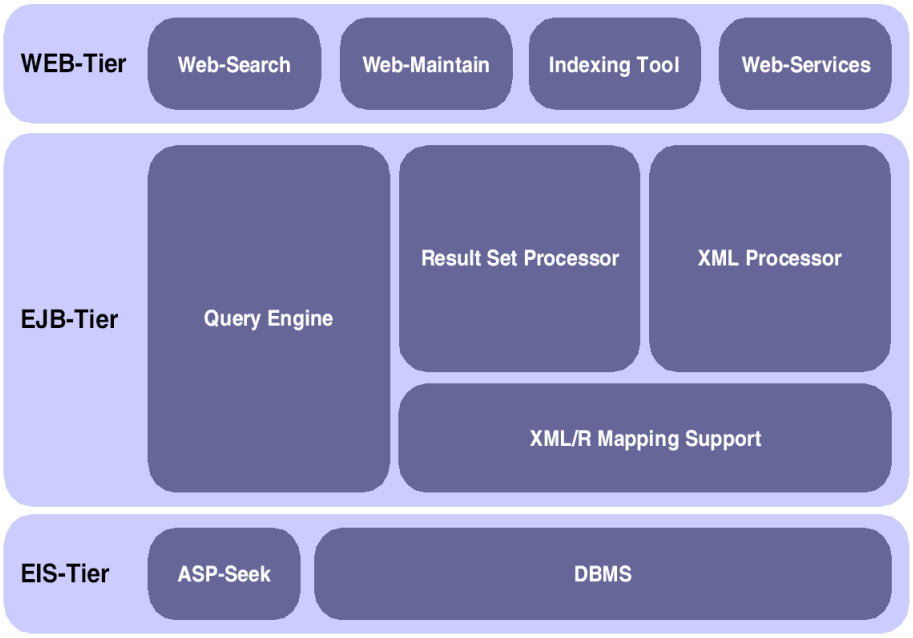

Abbildung 2: Meta-Akad Architektur

Schichten verantwortlich. Sie besteht aus einem relationalen Datenbanksystem sowie dem Datenspeicher der Volltextsuche.

2.2 Integration in den Suchprozess

Eine Suche in Meta-Akad verläuft in verschiedenen Phasen: Nach dem Anstoßen der Suche über das Web-Tier, übernimmt die Komponente "QueryEngine" im EJB-Tier das Finden der zugehörigen Dokumente und materialisiert alle Treffer in einer Tabelle des relationalen Datenbanksystems. anschließend übergibt sie den Namen der Ergebnistabelle an die Komponente "Result Set Processor", welche dem Web-Tier den Zugriff auf das Such-Ergebnis erlaubt. Das eigentliche Finden der Dokumente läuft also in der Komponente "QueryEngine" ab.

Der Ablauf der Anfrageverarbeitung in dieser Komponente ist in Abbildung 3 dargestellt: Bei der Analyse der Anfrage durch den QueryParser wird diese in ein internes Format, in Form eines Baums, umgewandelt. anschließend wird dieser vom LogicTransformer standardisiert und optimiert. Im dritten Schritt wird ein konkreter Anfrageplan, welcher die Grundlage für spätere SQL-Anfragen zur Suche nach Meta-Daten darstellt, ermittelt. Anfragen an die Volltextsuchmaschine werden hier gestellt und Antworten in Form von in Tabellen materialisierten Suchergebnissen erwartet. Nach einem Schritt zur Optimierung des Anfrageplans, wird dieser dann vom "QueryProcessor" ausgeführt und die notwendigen SQL-Anfragen zur Meta-Daten Suche sowie zur Integration der Ergebnisse der

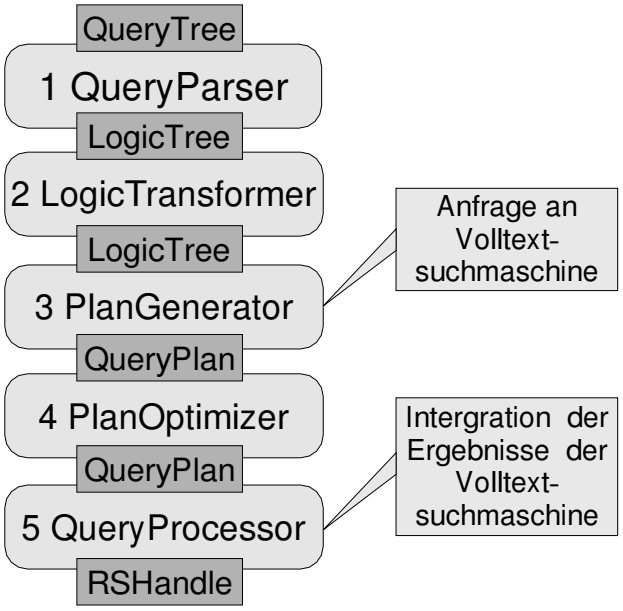

Abbildung 3: Anfrageverarbeitung durch die Komponente "QueryEngine"

Volltextsuche ausgeführt.

Die beschriebenen Schritte der Anfrageverarbeitung werden durch ein Enterise Java Bean "QECtrl" durchgeführt. Dabei greift dieses für die Durchführung der jeweiligen Schritte auf weitere Komponenten zurück. Im Falle der Volltextsuche geschieht dies durch die Verwendung des "Plangenerator" Beans, welche die Suchanfrage an eine "FulltextCacheDAO" genannte Komponente stellt. Diese wiederum leitet die Anfrage an die eigentliche Volltext-Komponente "FTCache" mittels eines Aufrufs der Methode "getCacheTableNameForFTQuery" weiter. Sie bekommt den Suchbegriff als Parameter und liefert den Namen einer Tabelle mit den Suchergebnissen zurück. Falls für den Suchbegriff kein Eintrag im Cache vorhanden ist, muss dazu eine Volltextsuchanfrage an die Suchmaschine gestellt werden.

Bis auf die Komponente "FTCache" waren zum Zeitpunkt dieser Projektarbeit bereits alle anderen Komponenten der Anfrageverarbeitung realisiert und sind in [4] näher beschrieben.

2.3 Verwaltung von Inhalten

In Meta-Akad besteht die Möglichkeit der Erschließung neuer Lehrinhalte, sowie der Entfernung selbiger. Diese Aktionen müssen auch Auswirkungen auf die Volltextsuchmaschine haben: Neu hinzugefügte Inhalte müssen indiziert, alte Inhalte gelöscht werden. Die Liste mit neuen und gelöschten URLs muss dazu auf dem von der Volltextsuchmaschine genutzten System zur Verfügung gestellt

werden.

2.4 Der Indizierungsprozess

Um die Volltextsuchmaschine mit Inhalten zu füllen und diese aktuell zu halten, ist ein regelmäßiger Indizierungsvorgang notwendig. Bei der Integration des Indizierungsprozesses in Meta-Akad galt es verschiedene Probleme zu bewältigen: Zum einen muss es möglich sein, von einem Suchergebnis auf den dazugehörigen Lehrinhalt und damit die dazugehörige, einmalig im System vergebene ID, rückschließen zu können. Zum anderen liegen die über Meta-Akad erfassten Inhalte in von Aspseek nicht direkt unterstützten Formaten, wie beispielsweise Postscript oder dem Rich Text Format, vor und müssen daher zur Indizierung konvertiert werden.

2.4.1 Zuordnung indizierter Dokumente zu Lehrinhalten

Um den Rückschluss von einem Suchergebnis der Volltextsuche auf die in Meta-Akad vergebene Dokument-ID zu ermöglichen, muss diese aus den von Aspseek gelieferten Trefferdaten rekonstruierbar sein. Hierfür bietet sich zunächst die Verwendung der URL an. Allerdings ist dann ein eindeutiger Rückschluss nicht möglich, da die gleiche URL in Meta-Akad mehrere IDs haben kann. Aus diesem Grund werden die URLs der zu indizierenden Dokumente vor dem Start des Indizierungsprozesses um ihre Dokument-ID erweitert. Bei der Analyse von Suchergebnissen wird diese dann extrahiert und entfernt.

2.4.2 Formatkonvertierung

Beim Indizierungsvorgang können Dokumente, welche nicht in den von Aspseek nativ unterstützen Formaten Text und HTML vorliegen, über externe Filterprogramme konvertiert werden. Um festzustellen, welcher Filter angewendet werden soll, verwendet Aspseek den vom Webserver zurückgegebenen MIME-Typ. Eine Kaskadierung von Filtern ist nicht vorgesehen, jeder eingetragene Filter muss also direkt in eines der nativen Formate übersetzen.

Diese durch Aspseek gemachten Vorgaben warfen erhebliche Probleme auf:

1. Der vom Webserver zurück gelieferte MIME-Typ ist oft nicht aussagekräftig: Oft sind die MIME-Einstellungen fehlkonfiguriert und liefern beispielsweise, unabhängig vom Dokumenttyp, einen einzigen MIME-Typ zurück.

2. Eine der Anforderungen war die nahtlose Integration von Lehrinhalten, welche in komprimierter- oder auch Archivform vorlagen. Fehlende Kaskadierungsmöglichkeiten von Filtern in Aspseek erschweren die Implementierung.

3. Einige der verwendeten Filter sind nicht in der Lage direkt ein von Aspseek akzeptiertes Format auszugeben.

Zur Lösung dieser Probleme wurde der Ansatz gewählt einen “Superfilter” zu konstruieren, welcher jegliche unterstützte Formate umwandeln kann und auch Funktionen, wie den Umgang mit Archiven und die Verkettung mehrerer Filter, zur Verfügung stellt.

3 Realisierung der Integration

3.1 Integration in den Suchprozess

Um die in 2.2 beschriebene Funktionalität bereitzustellen, greift das "FTCache" Enterprise Java Bean auf zwei Komponenten zurück:

1. FTDBCacheOperator
   Die Hauptaufgabe dieses Java Beans besteht in der Instandhaltung des Meta-Akad internen Caches von Volltext-Anfragen. Dazu wird sichergestellt, dass die Anzahl der Einträge im Cache stets unter der in der Konfigurationsoption "maxnumberofcachetable" eingestellten Anzahl bleibt. Weiterhin wird dafür gesorgt, dass Einträge, welche länger als in "tableusetimeout" eingestellt unbenutzt sind, gelöscht werden.
   Eine andere Funktion dieses Java Beans ist festzustellen ob die aktuelle Suchanfrage aus dem Cache bedient werden kann.

2. FTSearchEngine
   Dieses Java Bean übernimmt den Versand der Volltextsuchanfrage an Aspseek sowie die Analyse und Speicherung der Antwort. Dies geschieht in fünf Schritten:

   (a) Suchstring transformieren
   Hier wird der Suchstring für die Verarbeitung von Aspseek aufbereitet. Dies verhindert Fehlermeldungen aufgrund von Zeichenfolgen, welche Aspseek selbst zu interpretieren versucht. Beispiele für solche Zeichenfolgen sind eine ungleiche Anzahl an öffnenden und schließenden Klammern im Suchstring oder zwei auf einander folgende Minus-Zeichen.

   (b) HTTP-GET String erzeugen
   In diesem Schritt wird die an Aspseek zu stellende Anfrage in Form eines HTTP-GET Strings erzeugt. Dazu wird an die, unter der Konfigurationsoption "aspseek_location" gespeicherte URL zur Volltextsuchmaschine, der entsprechend kodierte Suchstring angehängt.

   (c) Cache-Tabelle anlegen
   Da das neue Suchergebnis in jedem Fall in den Cache aufgenommen werden soll, wird hier die notwendige Tabelle angelegt.

   (d) Abfrage durchführen
   Führt die Abfrage der Volltextsuchmaschine durch und analysiert die Antwort. Die Analyse wird mit Hilfe eines Kindes der abstrakten Klasse "ASPSeekReplyParser" durchgeführt. Derzeit existiert hierfür nur eine Implementierung "ASPSeekReplyParserDOM", welche das Document Object Model zur Analyse der XML-Inhalte verwendet. Eine weitere Implementierung, z.B. mit Hilfe der performanteren Simple API for XML ist leicht einzufügen. Das Ergebnis der Analyse ist ein Sammelobjekt, welches die Treffer enthält.

   (e) Ergebnis in die Datenbank schreiben
   Schreibt die im Sammelobjekt gespeicherten Treffer der Suchanfrage in die in Schritt c) angelegte Cache-Tabelle.

In Abbildung 4 ist eine Suchanfrage nach dem Begriff “chaos”, welche aus dem Cache beantwortet werden kann, dargestellt. Bevor die eigentliche Suchanfrage von FTCache verarbeitet wird, ruft dieser zuerst Funktionen zur Instandhaltung des Caches im FTDBCacheOperator auf. Erst danach erfolgt die Anfrage ob für den Suchbegriff “chaos” bereits ein Eintrag im Cache existiert. Da dies hier der Fall ist, gibt der FTDBCacheOperator den Tabellennamen, hier “ftcache_4711”, zurück. Dieser wird dann von FTCache an den Aufrufer weitergereicht.

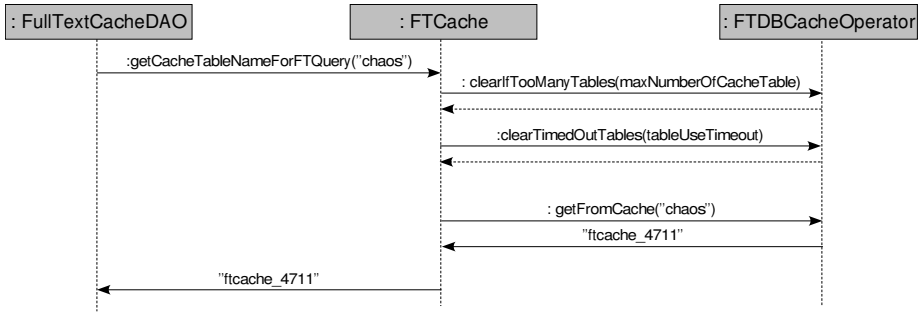

Abbildung 4: Suchanfrage, welche aus dem Cache beantwortet wird.

Der kompliziertere Fall wird in Abbildung 5 dargestellt: Hier existiert kein Cache-Eintrag für den Suchbegriff. Nach Aufruf der Cache Instandhaltungsroutinen gibt der FTDBCacheOperator auf die Anfrage, ob ein Eintrag für den Suchbegriff “chaos” im Cache existiert, “null” zurück. Folglich bedient sich FTCache des Java Beans FTSearchEngine um eine Volltextsuche auszuführen. Dieses führt nacheinander die fünf oben geschilderten Schritte der Suche aus und bedient sich der ASPSeekReplyParserDOM-Klasse um die Rückgabe von Aspseek zu analysieren. Nach dem erfolgreichen Schreiben des Suchergebnisses in eine neue Tabelle, wird diese beim FTDBCacheOperator angemeldet und steht damit für zukünftige Anfragen nach dem gleichen Suchbegriff zur Verfügung. Als Ergebnis liefert das FTSearchEngine den Namen der neu generierten Tabelle an FTCache zurück, welches diesen dann an den Aufrufer weiterreicht.

3.2 Verwaltung von Inhalten

Die Bereitstellung von Listen hinzugefügter und gelöschter Inhalte wurde durch das Servlet “FTCrawlPageGenerator” umgesetzt: Abhängig von Aufrufparametern kann es die Liste der ab einem angegebenen Datum hinzugefügten oder gelöschten URLs in zwei Formaten, HTML oder Komma separiert, ausgeben. Um die nötigen Informationen zu erhalten greift es auf die Klasse “ETController” im Enterprise Tier zurück.

Das Hinzufügen und das Löschen von Lehrinhalten wurde jeweils in einem im Unterverzeichnis “maintain/” befindlichen Perl-Script realisiert.

Das Script “urladder.pl” fügt zu diesem Zweck zur Liste der durch Aspseek zu indizierenden Dokumente einen Aufruf des FTCrawlPageGenerator-Servlets

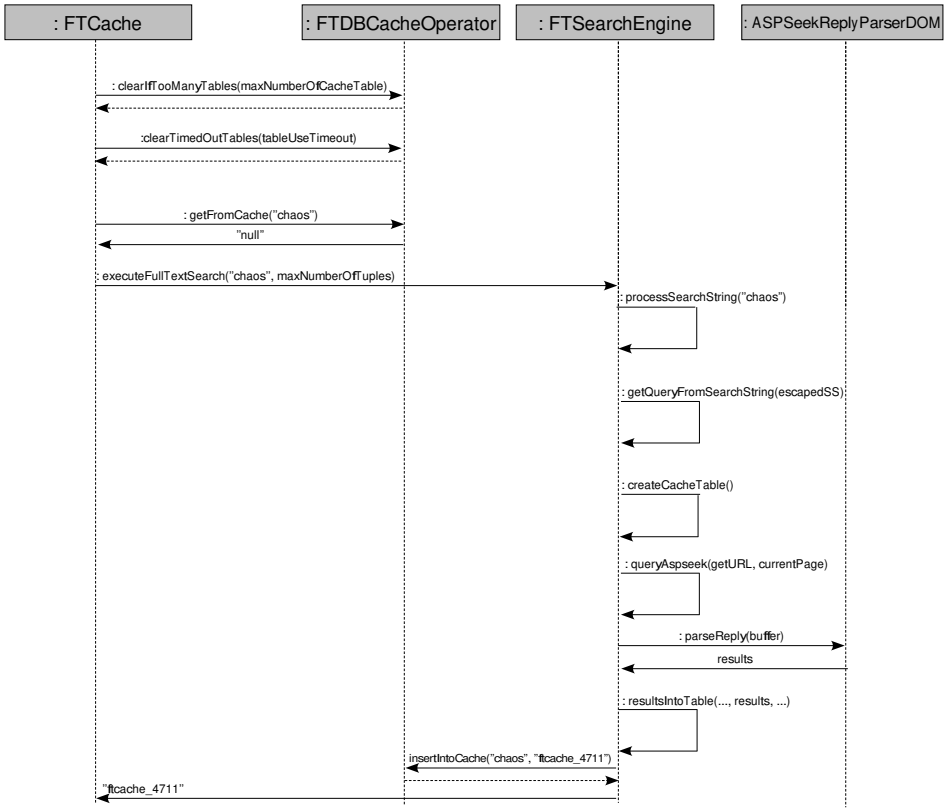

Abbildung 5: Suchanfrage, welche nicht aus dem Cache beantwortet wird.

hinzu. Dabei wird als Ausgabeformat HTML, als Listentyp die hinzugefügten URLs und als Datum das Datum des letzten Aufrufs von "urladder.pl" gewählt. Gleichzeitig werden eventuelle andere Aufrufe des Servlets aus der Liste der durch Aspseek zu indizierenden Dokumente entfernt. Beim nächsten Indizierungslauf werden die seit dem letzten Lauf neu hinzugekommenen URLs also mit erfasst.

Auch das Script "urlremover.pl" greift auf das FTCrawlPageGenerator-Servlet zu. Allerdings lässt es sich die Liste der seit dem letzten Aufruf gelöschten URLs im kommaseparierten Format ausgeben und analysiert diese. anschließend ruft es für jede enthaltene URL einen Befehl zum Löschen aus der Aspseek-Datenbank auf. Dies ist insofern suboptimal, als das jeder Löschvorgang eine mehrere Sekunden dauernde Umstrukturierung der Datenbasis zur Folge hat. Leider ist das Löschen von mehreren URLs gleichzeitig jedoch nicht in Aspseek vorgesehen.

Sowohl "urladder.pl" als auch "urlremover.pl" werden einmal täglich ausgeführt.

3.3 Der Indizierungsprozess

Der Indizierungsprozess sollte täglich gestartet werden. Dies wird durch einen Eintrag in der crontab des Benutzers “aspseek” sichergestellt.

3.3.1 Zuordnung indizierter Dokumente zu Lehrinhalten

Die Kodierung der Dokument-ID in die URL erfolgt durch Einfügen von “lrid—NR” hinter den Protokoll-Identifier, wobei NR durch die jeweilige ID ersetzt wird. Für ein Script mit der Dokument-ID 4711, welches unter der URL “http://www.uni-kl.de/script1.ps” zu finden ist, wird diese also zu “http://lrid—4711/www.uni-kl.de/script1.ps”.

Um es dem Indizierungsprozess ohne Eingriff in den Quellcode zu ermöglichen solche kodierten URLs zu verarbeiten, wird ein Proxy verwendet. Dieser filtert die Dokument-ID vor Weiterleitung der Anfrage aus. Realisiert ist dies durch den Einsatz der freien Proxy-Software “Squid” zusammen mit einem speziell zugeschnittenen URL-Redirector, welcher die eigentliche Filterung übernimmt. Der URL-Redirector ist relativ zum Aspseek Verzeichnis unter “redirector/redirector.pl” zu finden. Er erwartet die Eingabe von URLs auf der Standardeingabe und schreibt die gefilterten URLs auf die Standardausgabe.

Eine andere Form der Kodierung, nämlich das Anhängen mittels des Anker-Zeichens “#”, war nicht umsetzbar, da Aspseek solche Anker beim Indizierungsprozess verwirft und diese folgerichtig in Trefferlisten nicht mehr enthalten sind.

3.3.2 Formatkonvertierung

Das für die Formatkonvertierung notwendige ”Superfilter”-Programm wurde in Perl realisiert. Es besitzt einen Konfigurationsabschnitt, wo zur weiteren Filterung verwendete Programme eingetragen werden können. Das aktuelle Format der Datei wird über einen Aufruf des Unix-Befehls “file” festgestellt. Abhängig vom ermittelten Format wird ein Filterprogramm selektiert und ausgeführt. Danach wird erneut der Dateityp mittels “file” festgestellt. Dieser Vorgang wird so lange wiederholt bis die Daten im Format “Text” vorliegen, oder kein Filter für das aktuelle Format gefunden wurde, was einen Abbruch zur Folge hat. Dieses Vorgehen realisiert eine automatische Konkatenation von Filtern bis zur Erreichung des Zielformats. In Abbildung 6 ist dies am Bespiel dargestellt.

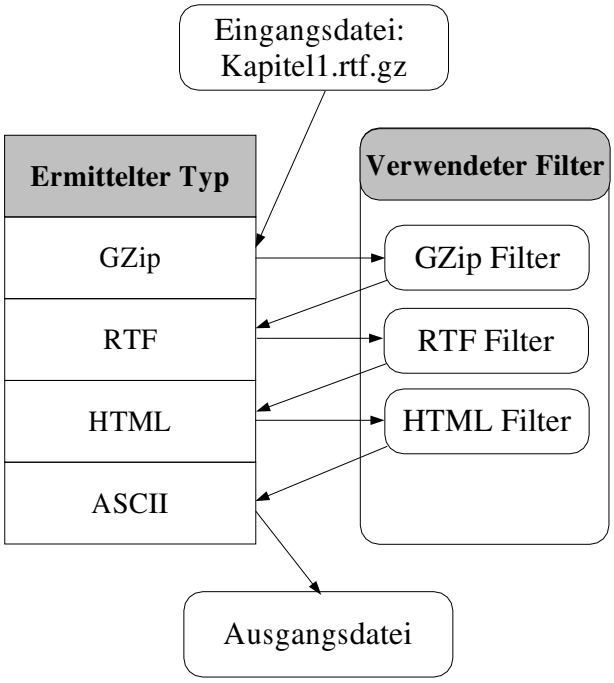

Abbildung 6: Beispiel eines Konvertierungsvorgangs des Superfilters

Filter, welche Archive verarbeiten, werden gesondert behandelt, da sie aus einer Quelldatei potentiell mehrere zu indexierende Dateien erzeugen. In diesem Fall wird der Inhalt des Archivs in ein temporäres Verzeichnis extrahiert und anschließend der Konvertierungsvorgang für jede im Archiv enthaltene Datei angestossen. Resultate werden konkateniert.

In der aktuellen Version unterstützt der "Superfilter" die Formate Postscript, Portable Document Format, Microsoft Word und Rich Text Format. Diese können in Archiven vom Typ "zip" sowie "gzip" enthalten sein. Eine Erweiterung um andere Filter- und Archivprogramme ist leicht möglich.

Um sicherzustellen, dass der "Superfilter" für alle nicht nativ unterstützten Dokumenttypen von Aspseek genutzt wird, wurde er in der Konfigurationsdatei für eine große Zahl von MIME-Typen eingetragen. Dieser Abschnitt der Konfigurationsdatei konnte leider nicht kompaktiert werden, da Aspseek hier keine Unterstützung für Wildcards bietet.

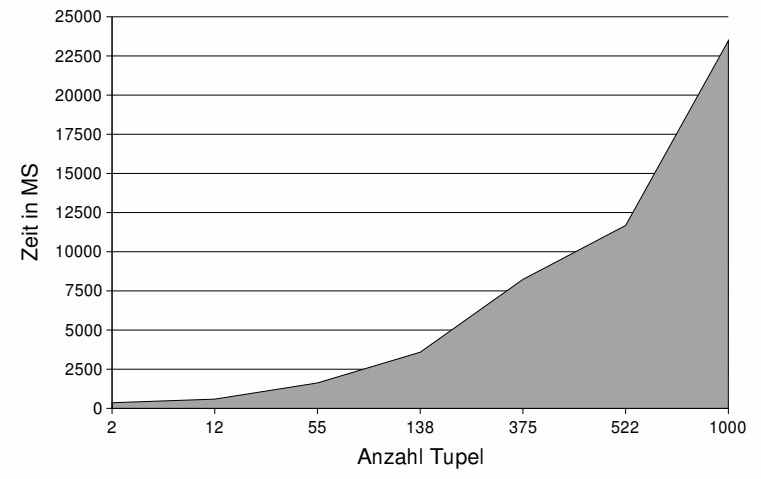

Abbildung 7: Dauer der Volltextsuche abhängig von der Anzahl der Treffer

4 Resultate und Ausblick

4.1 Performanz

Um die Volltextsuche überhaupt benutzbar zu machen, muss deren Antwortzeit akzeptabel sein. Abbildung 7 stellt die Arbeitszeit der derzeitigen Implementierung für den Fall eines Cache-Misses, abhängig von der Anzahl der gefundenen Tupel, dar. Dabei fällt auf, dass diese Zeit bei ca. 100 Treffern einen kritischen Wert erreicht.

Auf die Frage nach den Hauptursachen hierfür geht Abbildung 8 ein. Diese Abbildung stellt den Anteil des in Kapitel 2e beschriebenen Schrittes “e” der Volltextsuche, in welchem die Suchergebnisse in die Datenbank eingefügt werden, in Relation zur Anzahl der Treffer dar. Dieser steigt mit größer werdender Tupelzahl steil an und hat bei 100 Tupeln bereits 70 Prozent erreicht. Für grössere Treffermengen steigt dieser Anteil sogar bis auf den Wert von 85 Prozent.

Offensichtlich bietet sich hier also das grösste Potential für zukünftige Optimierungen. Ein Ansatz wäre die Benutzung von so genannten “Batch-Insert” Techniken zum schnelleren Einspeisen der Suchergebnisse. Weiterhin wäre es auch denkbar auf die Anforderung, mit jedem Suchergebnis ein Textextrakt zu liefern, zu verzichten und somit die in die Tabelle einzufügende Datenmenge drastisch zu verringern. Schlussendlich könnte auch der im Projekt sowieso geplante Einsatz einer anderen Datenbanksoftware entscheidende Geschwindigkeitszunahmen in diesem Schritt mit sich bringen.

Der Einfluss der Volltextsuche auf die Option “Einfache Suche” ist in Abbildung 9 dargestellt: Hier wird sichtbar, dass die Volltextsuche einen großen Anteil an der für eine Suche benötigten Zeit besitzt. Dabei ist anzumerken, dass der für die Suche mit 375 Treffern eingesetzte Suchbegriff sehr viele Treffer in der

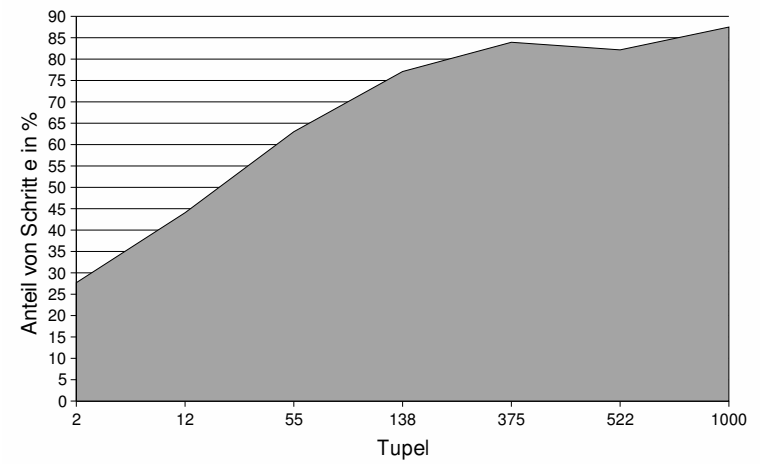

Abbildung 8: Anteil des Schrittes “e“ an der Volltextsuche

Metadatensuche erzeugt. Der fallende Wert trotz steigender Tupelzahl resultiert daraus.

Für den Fall, dass eine Suchanfrage aus dem Cache beantwortet werden kann, ergibt sich eine feste Antwortzeit der Volltextsuche. Diese liegt im Bereich von 100 Millisekunden.

Um die Antwortzeiten bei Einsatz der Volltextsuche in einem vernünftigen Rahmen zu halten wird empfohlen die in 5.4 beschriebene Konfigurationsoption “Maximale Treffer der Volltextsuche” auf 100 einzustellen.

4.2 Implementierungsentscheidungen

Während der Implementierung der Volltextsuche wurden einige Entscheidungen getroffen, welche im Folgenden kurz zusammengefasst sind.

Um die Skalierbarkeit zu erhöhen wurde die Umgebung für die Volltextsuche von der Laufzeitumgebung für Meta-Akad komplett getrennt: Die Volltextsuche kann auf einem eigenständigen Rechner laufen. Die notwendige Kommunikation mit Meta-Akad findet ausschließlich per HTTP, mit Hilfe der in Kapitel 3 beschriebenen Mechanismen statt. Somit wurde eine äußerst lose Kopplung der beiden Systeme realisiert.

Ausgehend von der festen Anforderung als Volltextsuchemaschine die Software Aspseek [2] einzusetzen, mussten verschiedene dadurch aufgeworfene Probleme gelöst werden: Da die von Aspseek gebotenen Möglichkeiten zur Indizierung von in Fremdformaten vorliegenden Dokumenten als unzureichend befunden wurden, wurden eigene Filtermechanismen mittels eines Superfilter genannten Perl-Programms, wie in 3.3.2 beschrieben, realisiert. Da das von Aspseek genutzte Protokoll zur Kommunikation mit dem Suchdienst proprietär und undokumentiert war, wurde das als CGI-Script mitgelieferte Frontend genutzt und

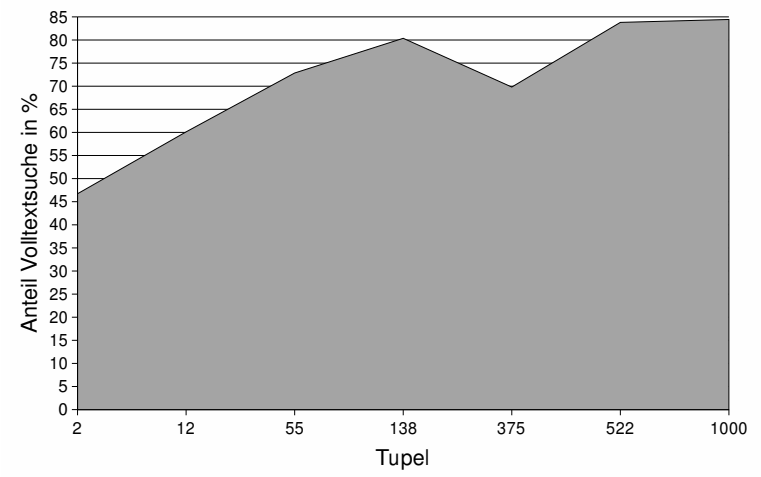

Abbildung 9: Anteil der Volltextsuche an der Gesamtsuchzeit

so angepasst, dass es Suchergebnisse als XML-Dokument ausgibt. Die Nutzung des ebenfalls mitgelieferten Apache-Moduls wäre ebenfalls möglich gewesen, hätte jedoch keinen nennenswerten Einfluss auf die Performanz gehabt und eine Abhängigkeit von Apache als HTTP-Server bedeutet.

Um die Zuordnung von Dokument-IDs zu den von Aspseek zurückgegebenen Treffern zu gewährleisten, wurde wie in 3.3.1 beschrieben die Dokument-ID bei der Indizierung in die URL kodiert, was den Einsatz eines Proxy-Servers beim Indizierungsvorgang notwendig machte: Dieser entfernt die Dokument-ID vor Weiterleitung des HTTP-Aufrufs.

4.3 Ausblick

Obwohl die im Rahmen dieser Projektarbeit entwickelte Anbindung einer Volltextsuche an Meta-Akad durchaus benutzbar und auch praxistauglich ist, bieten sich doch zahlreiche Möglichkeiten der Verbesserung.

Als wichtigstes Feld sollte die Verbesserung der Performanz gesehen werden. Diese kann mit den in Kapitel 4.1 beschriebenen Vorschlägen sicherlich noch signifikant erhöht werden. Weiterhin wäre auch eine Parallelisierung von Schritten der Metadaten- und Volltextsuche denkbar. Eine Implementierung des FTCache Enterprise Java Beans als Message Driven Bean bietet sich hier an.

Ein anderer interessanter Bereich ist die Verbesserung der Skalierbarkeit: Durch die sehr lose Kopplung der Umgebung für die Volltextsuche an Meta-Akad wäre hier beispielsweise der Einsatz von mehreren Rechnern für die Volltextsuche, welche im Round-Robin Verfahren angesprochen werden, denkbar. Dies ist insbesondere deshalb interessant, weil der zeitlich kritische Vorgang der Suche einen nur-lese Zugriff auf den Volltextdatenbestand darstellt und es somit leicht fallen dürfte Volltextdatenbanken auf mehreren Rechnern konsistent zu

halten.

5 Installationshinweise

5.1 Anpassung von Pfaden

Die für die Anbindung an Meta-Akad zuständigen Scripte benötigen verschiedene gemeinsame Einstellungen, welche in der Datei "common/config.pm" festgelegt werden können.

- Pfad zur Aspseek-Installation ("getAspSeekPath"). Standardwert: "/usr/local/aspseek"
- URL zum FTCrawlServlet ("getFTCrawlServlet").

5.2 Suchmaschine Aspseek

Die Konfiguration von Aspseek geschieht mittels der im Verzeichnis "etc/" enthaltenen Dateien. Für die Integration mit Meta-Akad sind folgende Konfigurationsdateien von besonderem Interesse:

- aspeek.conf: Einstellungen für den Indizierungsvorgang

  Folgende Parameter sollten abweichend von den Standardeinstellungen konfiguriert werden:

  - Maximale Größe von indizierten Dokumenten ("MaxDocSize"): Wert erhöhen um sicherzustellen, dass auch große Dateien indiziert werden. Empfohlener Wert: 104857600 - entspricht 100MB.
  - Zeitraum bis zur Re-Indizierung von Dokumenten ("Period"): Sollte auf einen großen Wert gestellt werden, da Re-Indizierung manuell angestossen wird. Empfohlener Wert: 1y
  - Maximale Linktiefe von der Start-URL ("MaxHops"): Da die Links aller zu indizierender Dokumente auf einer einzigen Seite vorhanden sind, sollten diese nicht weiter verfolgt werden. Empfohlener Wert: 1
  - Maximale Anzahl von pro Server indizierten Dokumenten ("MaxDocsPerServer"): Hohen Wert einstellen um zu verhindern, dass Dokumente nicht indiziert werden. Empfohlener Wert: 5000
  - Erhöhung der Linktiefe bei Redirects ("MaxDocsPerServer"): Empfohlener Wert: no
  - Beachtung der Datei robots.txt ("Robots"): Da davon ausgegangen wird, dass nur Dokumente erschlossen werden, deren Bereitstellung gewünscht ist sollte die Datei robots.txt nicht beachtet werden. Empfohlener Wert: no
  - URLs, welche nicht explizit zur Indizierung freigegeben wurden, löschen (DeleteNoServer): Empfohlener Wert: no
  - Links auf andere Server folgen ("FollowOutSide"): Da die Links auf der Seite mit den zu indizierenden Inhalten alle vom Server wegführen muss Aspseek so konfiguriert werden, dass es diese Links auch verfolgt. Empfohlener Wert: yes

  - Proxy (“Proxy”): Hier muss die Adresse des Rechners, auf welchem der zur Indizierung verwendete Proxy läuft, eingetragen werden. Der hier eingestellte Proxy muss das spezielle URL-Format, welches die Dokument-ID kodiert, wie in 3.3.1 beschrieben filtern.
  - Zu indizierende Server (“Server”): Da die zu indizierenden Dokumente wie in 3.2 beschrieben zur Datenbasis hinzugefügt werden, sollten alle hier eingestellten Werte auskommentiert werden.
  - Zur Indizierung verbotene URLs (“Disallow”): Da Aspseek von vornherein nur auf zu indizierenden Dokumenten arbeitet sollten keine URLs verboten werden, was durch Auskommentieren aller Disallow-Zeilen erreicht wird.
  - Konvertierungsprogramme (“Converter”): Hier sollte für jeden gewünschten MIME-Typ der in 3.2 beschriebene Superfilter eingetragen werden. Beispiel: “Convert application/postscript text/plain /usr/local/aspseek/filter/filter.pl $in $out”

  Eine Beispielkonfiguration findet sich im Anhang in 6.1.

- searchd.conf: Einstellungen für die Suchdienst

  - Konfigurationsdatei der Stopwörter (“include stopwords.conf”): Auskommentieren, da Stopwörter bei der Suche zu Fehlermeldungen führen können.

  Eine Beispielkonfiguration findet sich im Anhang in 6.2.

- s.htm: Einstellungen für den Suchclient

  - Das Ausgabeformat des Such-Clients muss für die Integration in Meta-Akad speziell angepasst werden, damit es anstatt einer HTML-Seite XML generiert. Diese Datei sollte also zunächst durch die in Meta-Akad enthaltene Version ersetzt werden.
  - Volltextausschnitte (“MaxExcerpts”): Bestimmt wie viele Ausschnitte eines Dokuments mit Treffern maximal angezeigt werden. Da die Generierung dieser Ausschnitte viel CPU-Zeit kostet, wird dringend empfohlen diesen Wert auf 0 zu stellen.

  Eine Beispielkonfiguration findet sich im Anhang in 6.3.

5.3 Indizierungsumgebung

5.3.1 Proxy Squid

Um die in 3.3.1 beschriebene Funktionalität zu realisieren wird die Proxy-Software “Squid” eingesetzt. Nach einer Standardinstallation ist die Anpassung folgender Parameter notwendig:

- Minimalgröße des persistenten Caches (“cache_swap_low”): 0
- Maximalgröße des persistenten Caches (“cache_swap_high”): 0
- Externer URL-Filter (“redirect_program”): /usr/local/aspseek/redirector/redirector.pl

Eine Beispielkonfiguration findet sich im Anhang in 6.4.

5.3.2 Formatkonvertierung

Neue Dateiformate mit entsprechenden Filtern können in der Datei "filter/filter.pl" in der Konfigurations-Sektion definiert werden.

- Dateiformate ("%unixFileTypes"): Damit der Filter neue Dateiformate erkennt muss hier eine Zeile "*regexp*" => "*formatname*", eingefügt werden. Bei *regexp* handelt es sich um einen regulären Ausdruck, welcher die Ausgabe des "file" Kommandos bei Anwendung auf eine Datei im neuen Format beschreibt. "*formatname*" hingegen legt den internen Namen für das definierte Dateiformat fest.
  Beispiel: *"PDF Document" => "pdf",*

- Filter ("%converters"): Hier werden Filter auf den eingetragenen Dateiformaten definiert. Neue Filter werden durch Einfügen einer Zeile "formatname" => { "program" => "*Programmname*", "stdout" => "*std_ ausgabe*", "last" => "*letzter_filter*", "morefiles" => "*mehr_ dateien*" }, eingetragen. Folgende Parameter müssen dabei ersetzt werden:

  - *Programmname:* Name des auszuführenden Programms. Vorkommen von "!IN" werden durch den Namen der Eingabedatei ersetzt, "!OUT" durch den Namen der erwarteten Ausgabedatei.
  - *std_ ausgabe*: Signalisiert ob das Filterprogramm die Eingabe auf der Standardeingabe erwartet und das Ergebnis auf die Standardausgabe schreibt. Wert "1" signalisiert, dass dies der Fall ist, Wert "0" bedeutet, dass der Filter selbst eine Ausgabedatei anlegt.
  - *letzter_filter*: Gibt an ob der Filter in ein von Aspseek verstandenes Format konvertiert (Wert "1") oder nicht (Wert "0").
  - *mehr_ dateien*: Bestimmt ob es sich bei dem Filter um einen Archivfilter handelt (Wert "1") oder nicht (Wert "0"). Archivfilter erzeugen potentiell mehrere Dateien und werden daher in vorher angelegten temporären Verzeichnissen ausgeführt.

  Beispiel:
  "ps" => { "program" => "/usr/local/aspseek/filter/converters/bin/pstotext", "stdout" => "1", "last" => "1", "morefiles" => 0 },

5.4 Konfigurationsoptionen in Meta-Akad

Folgende Einstellungen lassen sich in der Meta-Akad internen Konfiguration vornehmen. Sie werden in der SQL-Tabelle "configurationstore" gehalten.

Parametername	Standardwert	Bedeutung
maxnumberofcachetable	50	Maximale Anzahl an Tabellen im Cache. Jede Tabelle speichert das Ergebnis genau einer Suchanfrage.
maxnumberoftuples	100	Maximale Anzahl von Treffern der Volltextsuche, welche Berücksichtigung finden.
tableusetimeout	600	Zeitraum nach welchem Cache-Einträge als ungültig angesehen werden in Sekunden.
ftcachetable_prefix	ftcache_	Prefix der für das Caching verwendeten Tabellen.
aspseek_location	-	URL der Aspseek-Installation. Zeigt auf das Suchscript (s.cgi)

Tabelle 1: Konfigurationsoptionen der Volltextsuche

6 Konfigurationsdateien

6.1 aspseek.conf

```
# This is the config file for ASPseek 1.2 index.
# See man pages aspseek.conf(5) and index(1) for details.
# Use '#' to comment out lines
# All command names are case insensitive (Proxy=PROXY=proxy)
# You may use '\' character to prolong current command to next line
# when it is required.
######
Include db.conf

# Debug level (none, error, warning, info, debug)
# Default value is info
DebugLevel info

# MaxDocSize bytes
## 100MB:
MaxDocSize 104857600

# Unicode charsets configuration
Include ucharset.conf

# Stopwords configuration
Include stopwords.conf

# Reindex period
Period 1y

#MaxHops <number>
# Maximum path in hops (or "mouse clicks") from start url.
MaxHops 1

# Limits the number of documents retrieved from Server.
MaxDocsPerServer 5000

# Allow/disallow index to increment hops value when redirects are
# encountered. Applies only to redirects generated by Location headers.
IncrementHopsOnRedirect no

# Allows/disallows using robots.txt and <META NAME="robots">
Robots no

# Use it to choose whether delete or not those URLs which have no
# correspondent "Server" commands.
DeleteNoServer no

# Allow/disallow index to walk outside servers given in config file.
```

FollowOutside yes

#Proxy your.proxy.host[:port]
Use proxy rather then connect directly
Proxy localhost:3128

#DeltaBufferSize <Number>
Size of buffer for each of 100 delta files in kilobytes.
DeltaBufferSize 64

Converter text/rtf text/plain /usr/local/aspseek/filter/filter.pl $in $out

.
.
.

(Converter-Anweisungen für alle bekannten MIME-Typen)

6.2 searchd.conf

This is the configuration file for ASPseek 1.2 search daemon (searchd)
See man pages searchd.conf(5) and searchd(1) for details.
#
#
Lines starting with # are comments
Include db.conf
######
AllowFrom 127.0.0.1
######
Unicode charsets configuration
Include ucharset.conf
######

6.3 s.htm

<!–
This is the Akleon Template for Aspseek
–>
<!–variables
'variables' section defines some global parameters.
Each line starting with # is a comment.
#
#DaemonAddress is used by s.cgi to connect to searchd
#
WARNING: You can't put DNS name (www.myhost.com) in DaemonAddress, only
numeric IP address. This is because name lookup will slow down s.cgi
considerably.
DaemonAddress 127.0.0.1

```
# ResultsPerPage defines how many results will be shown on one page.
ResultsPerPage 1000

# s.cgi puts a small pieces from resulting document (a.k.a. excerpts)
# with found words/phrases highlighted.
# MaxExcerpts is limiting the number of such excerpts.
# If MaxExcerpts is set to zero (or no excerps were found)
# the first few lines from resulting document are shown.
#
# Please note that excerpt generation takes some CPU time, but it is hard
# to estimate it now. Each result page is fully unzipped during excerpt
# generation and scanned until required excerpts are found.
#
# Default value is 0.
#MaxExcerpts 1
# MaxExcerptLen is the upper limit of length of each excerpt
# Default value is 200
# MaxExcerptLen 200

#Error messages
#
ER_STOPWORDS Only stopword(s) are used in query.
ER_EXTRASYMBOL Extra symbols at the end
ER_EMPTYQUERY Empty query
ER_TOOSHORT Too few letters or digits are used at the beginning of pattern
ER_NOQUOTED Unmatched string quote
ER_NOPARENTHESIS Unmatched parenthesis
ER_NOSEARCHD Can not connect to search daemon
# End of variables section
–>
<!–top–>
<?xml version="1.0" encoding="iso-8859-1"?>
<searchresults>
<!–/top–>
<!–hiopen–>
<!–/hiopen–>
<!–hiclose–>
<!–/hiclose–>
<!–hiexopen–>
<!–/hiexopen–>
<!–hiexclose–>
<!–/hiexclose–>
<!–exopen–>
<!–/exopen–>
<!–exclose–>
<!–/exclose–>
<!–hicolors–>
<!–/hicolors–>
<!–cachetop–>
<!–/cachetop–>
```

```
<!–sizeb–>
<!–/sizeb–>
<!–sizek–>
<!–/sizek–>
<!–ressites–>
<!–/ressites–>
<!–resurls–>
<!–/resurls–>
<!–restop–>
<matches start="$f" stop="$l" total="$t"/>
<!–/restop–>
<!–res–>
<result number="$DN">
<url><![CDATA[$DU]]></url>
<rank>$DR</rank>
<extract><![CDATA[$DX]]></extract>
</result>
<!–/res–>
<!–res2–>
<result number="$DN">
<url><![CDATA[$DU]]></url>
<rank>$DR</rank>
<extract><![CDATA[$DX]]></extract>
</result>
<!–/res2–>
<!–res0–>
<result0 $DT $DX $DU $DZ $CC $CL>
<!–/res0–>
<!–resbot–>
<!–/resbot–>
<!–clone–>
<!–/clone–>
<!–cached–>
<!–/cached–>
<!–textversion–>
<!–/textversion–>
<!–navigator–>
<!–/navigator–>
<!–navleft–>
<!–/navleft–>
<!–navleft0–>
<!–/navleft0–>
<!–navbar1–>
<!–/navbar1–>
<!–navbar0–>
<!–/navbar0–>
<!–navright–>
<!–/navright–>
<!–navright0–>
<!–/navright–>
```

<!–complexPhrase–>
complex phrase error
<!–/complexPhrase–>
<!–complexExpression–>
complex expression error
<!–/complexExpression–>
<!–queryerror–>
<error>$E</error>
<!–/queryerror–>
<!–inres–>
<!–/inres–>
<!–notfound–>
<matches start="0" stop="0" total="0"/>
<!–/notfound–>
<!–noquery–>
<error>No words to search for</error>
<!–/noquery–>
<!–error–>
<error>$E</error>
<!–/error–>
<!–bottom–>
</searchresults>
<!–/bottom–>

6.4 squid.conf

hierarchy_stoplist cgi-bin ?

acl QUERY urlpath_regex cgi-bin \?
no_cache deny QUERY

cache_swap_low 0
cache_swap_high 0

redirect_program /usr/local/aspseek/redirector/redirector.pl

acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl SSL_ports port 443 563
acl Safe_ports port 80 # http
acl Safe_ports port 21 # ftp
acl Safe_ports port 443 563 # https, snews
acl Safe_ports port 70 # gopher
acl Safe_ports port 210 # wais
acl Safe_ports port 1025-65535 # unregistered ports
acl Safe_ports port 280 # http-mgmt
acl Safe_ports port 488 # gss-http
acl Safe_ports port 591 # filemaker
acl Safe_ports port 777 # multiling http

```
acl CONNECT method CONNECT
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localhost
http_access deny all
icp_access allow all
```

7 Klassendokumentation

7.1 Package metabase.queryengine.ejb.ftcache

Package Contents *Page*

7.1.0.1 Interface **FTCache**

Remote Interface for QueryExecutionController Bean

Declaration

public interface FTCache **implements** javax.ejb.EJBObject

Methods

- *getCacheTableNameForFTQuery*

  `public String` **getCacheTableNameForFTQuery**`(`
  `java.lang.String` **searchstring** `)`

  - **Usage**
    - Get table name of cached full text query results
  - **Parameters**
    - `searchstring` - text that was send to the full text engine
  - **Returns** -
    String object, which contains the table name

- *getMaxNumberOfCacheTable*

  `public int` **getMaxNumberOfCacheTable**`( )`

  - **Usage**
    - Retrieve MaxNumberOfCacheTable out of configuration store
  - **Returns** -
    MaxNumberOfCacheTable

- *getMaxNumberOfTuples*

  `public int` **getMaxNumberOfTuples**`( )`

  - **Usage**
    - Retrieve MaxNumberOfTuples out of configuration store
  - **Returns** -
    MaxNumberOfTuples

- *getTableUseTimeout*

  `public int` **getTableUseTimeout**`( )`

  - **Usage**
    * Retrieve MaxNumberOfTuples out of configuration store
  - **Returns** -
    MaxNumberOfTuples

- *invalidateAll*

  `public void` **invalidateAll**`( )`

  - **Usage**
    * Invalidate the whole cache

- *invalidateCacheForFTQuery*

  `public void` **invalidateCacheForFTQuery**`( java.lang.String` **searchstring** `)`

  - **Usage**
    * Invalidate the cache for the given query
  - **Parameters**
    * `searchstring` - text that was send to the full text engine

- *pregenerateCacheTableForFTQuery*

  `public void` **pregenerateCacheTableForFTQuery**`(`
  `java.lang.String` **searchstring** `)`

  - **Usage**
    * pregenerate result of distinct fullt text query and store it into a table
  - **Parameters**
    * `searchstring` - text that was send to the full text engine

- *setMaxNumberOfCacheTable*

  `public void` **setMaxNumberOfCacheTable**`( int` **size** `)`

  - **Usage**
    * Store MaxNumberOfCacheTable persistently into configuration store,

      i.e. MaxNumberOfCacheTable represents the cache size and the overall

      amount of tables should not be without any reason greater than this value
  - **Parameters**

* `size` - MaxNumberOfCacheTable

- *setMaxNumberOfTuples*

  `public void` **setMaxNumberOfTuples**`( int  size )`

  - **Usage**
    * Store MaxNumberOfTuples persistently into configuration store,

      i.e. the amount of full text result entries is strictly limited

      to the given value. If there are more than MaxNumberOfTuples

      entries the rest is discarded.
  - **Parameters**
    * `size` - MaxNumberOfTuples

- *setTableUseTimeout*

  `public void` **setTableUseTimeout**`( int  size )`

  - **Usage**
    * Store TableUseTimeout persistently into configuration store,

      i.e. the specify the timeout after a query could be invalidated
  - **Parameters**
    * `size` - TableUseTimeout

7.1.0.2 Interface **FTCacheHome**

The home of the Controller Bean

providing a full text query cache

Declaration

public interface FTCacheHome **implements** javax.ejb.EJBHome

Methods

- *create*

  `public FTCache` **create**`( )`

7.1.0.3 Class **FTCacheEJB**

Implementation of the full text query cache controller interface

Declaration

public class FTCacheEJB **extends** java.lang.Object **implements** javax.ejb.SessionBean

Serializable Fields

- private ConfigDAO m_ConfigDAO
  –

Constructors

- *FTCacheEJB*

  `public` **FTCacheEJB**`( )`

Methods

- *ejbActivate*

  `public void` **ejbActivate**`( )`
- *ejbCreate*

  `public void` **ejbCreate**`( )`
- *ejbPassivate*

  `public void` **ejbPassivate**`( )`
- *ejbRemove*

  `public void` **ejbRemove**`( )`
- *getCacheTableNameForFTQuery*

  `public String` **getCacheTableNameForFTQuery**`(`
  `java.lang.String` **searchstring** `)`

  – **Usage**

    * Holt den Namen der Cache-Tabelle für den angegebenen Suchstring.

      Falls noch keine Cache-Tabelle existiert, wird zuerst eine Volltextsuche
      durchgeführt und diese dadurch erzeugt.
  - **Parameters**
    * `searchstring` - Suchstring
  - **Returns** -
    Name der Tabelle, welche die Ergebnisse der Volltextsuche enthält

---

- *getConfigurationData*

  `public void` **getConfigurationData**`( )`

  - **Usage**
    * Holt die persistent abgelegten Konfigurationsparameter des FTCaches

---

- *getMaxNumberOfCacheTable*

  `public int` **getMaxNumberOfCacheTable**`( )`

  - **Usage**
    * Ermittelt die maximale Anzahl von Einträgen im Cache
  - **Returns** -
    Maximale Anzahl von Einträgen im Cache

- *getMaxNumberOfTuples*

  `public int` **getMaxNumberOfTuples**`( )`

  - **Usage**
    * Holt maximale Anzahl Treffern der Volltextsuche die Berücksichtigung finden
  - **Returns** -
    Maximale Anzahl Treffern der Volltextsuche die Berücksichtigung finden

---

- *getTableUseTimeout*

  `public int` **getTableUseTimeout**`( )`

  - **Usage**
    * Holt Timeout für Cache-Einträe
  - **Returns** -
    size Timeout für Cache-Einträge

- *invalidateAll*

  `public void` **invalidateAll**`( )`

  - **Usage**
    - Invalidiert den gesamten Cache

- *invalidateCacheForFTQuery*

  `public void` **invalidateCacheForFTQuery**`( java.lang.String` **searchstring** `)`

  - **Usage**
    - Invalidiert den Cache-Eintrag für einen Suchstring
  - **Parameters**
    - `searchstring` - Suchstring

- *pregenerateCacheTableForFTQuery*

  `public void` **pregenerateCacheTableForFTQuery**`( java.lang.String` **searchstring** `)`

  - **Usage**
    - Generiert Cache-Tabelle für angegebenen Suchstring
  - **Parameters**
    - `searchstring` - Suchstring

- *setMaxNumberOfCacheTable*

  `public void` **setMaxNumberOfCacheTable**`( int` **size** `)`

  - **Usage**
    - Setzt maximale Anzahl von Einträgen im Cache
  - **Parameters**
    - `size` - Maximale Anzahl von Einträgen im Cache

- *setMaxNumberOfTuples*

  `public void` **setMaxNumberOfTuples**`( int` **size** `)`

  - **Usage**
    - Setzt maximale Anzahl Treffern der Volltextsuche die Berücksichtigung finden
  - **Parameters**

        * `size` - Maximale Anzahl Treffern der Volltextsuche die Berücksichtigung finden

---

- *setSessionContext*

  `public void` **setSessionContext**`( javax.ejb.SessionContext` **context** `)`

- *setTableUseTimeout*

  `public void` **setTableUseTimeout**`( int` **size** `)`

  - **Usage**
    * Setzt Timeout für Cache-Einträe
  - **Parameters**
    * `size` - Timeout für Cache-Einträge

- *setValueInConfigurationSetore*

  `public void` **setValueInConfigurationSetore**`( java.lang.String` **name**`,`
  `int` **value** `)`

  - **Usage**
    * Setzt den angegebenen parameternamen im configurationstore auf den angegebenen Wert
  - **Parameters**
    * `name` - Name des Parameters im configurationstore
    * `value` - Neuer Wert des Parameters im configurationstore

- *unsetEntityContext*

  `public void` **unsetEntityContext**`( )`

7.2 Package metabase.queryengine.beans.ftcache

Package Contents *Page*

Classes

7.2.0.4 Class **ASPSeekReplyParser**

Superklasse für Antwort-Parser für Aspseek

Declaration

public abstract class ASPSeekReplyParser **extends** java.lang.Object

Constructors

- *ASPSeekReplyParser*

  `public` **ASPSeekReplyParser**`( )`

Methods

- *getLrIdFromUrl*

  `public int` **getLrIdFromUrl**`( java.lang.String` **url** `)`

  - **Usage**
    - Ermittelt aus der URL die lrId.

      Aufbau der URL:
      http://domain.tld/lrid—4711/somepath für lrid 4711
  - **Parameters**
    - `url` - Die Url
  - **Returns** -
    Die ermittelte lrId

- *htmlunescape*

  `public static String` **htmlunescape**`( java.lang.String` **s1** `)`

  - **Usage**
    - Wandelt einige HTML-Zeichen in ihre UTF-Entsprechung um
  - **Parameters**
    - `s1` - String zum umwandeln
  - **Returns** -
    Unescapten String

- *parseReply*

  `public abstract HashMap` **parseReply**`( java.io.BufferedReader` **buf** `)`

  - **Usage**
    - Von den Subklassen zu implementierender Parser

- *xmlEscape*

  `public static String` **xmlEscape**`( java.lang.String` **s1** `)`

  - **Usage**
    - Sorgt dafür dass die Eingabe so escaped wird, dass sie XML-Dokumente nicht mehr non-wellformed machen kann
  - **Parameters**
    - `s1` - Zu escapender String
  - **Returns** -
    Escpater String

7.2.0.5 Class **ASPSeekReplyParserDOM**

Parser für ASPSeek: Version in DOM

Declaration

public class ASPSeekReplyParserDOM **extends** metabase.queryengine.beans.ftcache.ASPSeekReplyParser

Constructors

- *ASPSeekReplyParserDOM*

  `public` **ASPSeekReplyParserDOM**`( )`

Methods

- *parseReply*

  `public HashMap` **parseReply**`( java.io.BufferedReader` **buf** `)`

  - **Usage**
    - Parsed die Antwort von ASPSeek mittels DOM
  - **Parameters**
    - `buf` - ASPSeek-Antwort
  - **Returns** -
    HashMap von FTSearchResults der Ergebnisse

METHODS INHERITED FROM CLASS
`metabase.queryengine.beans.ftcache.ASPSeekReplyParser`

(in 7.2.0.4, page 39)

- *getLrIdFromUrl*

  `public int` **getLrIdFromUrl**`( java.lang.String` **url** `)`

  - **Usage**
    - Ermittelt aus der URL die lrId.

      Aufbau der URL:
      http://domain.tld/lrid—4711/somepath für lrid 4711
  - **Parameters**
    - `url` - Die Url
  - **Returns** -
    Die ermittelte lrId

- *htmlunescape*

  `public static String` **htmlunescape**`( java.lang.String` **s1** `)`

  - **Usage**
    - Wandelt einige HTML-Zeichen in ihre UTF-Entsprechung um
  - **Parameters**
    - `s1` - String zum umwandeln
  - **Returns** -
    Unescapten String

- *parseReply*

  `public abstract HashMap` **parseReply**`( java.io.BufferedReader` **buf** `)`

  - **Usage**
    - Von den Subklassen zu implementierender Parser

- *xmlEscape*

  `public static String` **xmlEscape**`( java.lang.String` **s1** `)`

  - **Usage**
    - Sorgt dafür dass die Eingabe so escaped wird, dass sie XML-Dokumente nicht mehr non-wellformed machen kann
  - **Parameters**
    - `sl` - Zu escapender String
  - **Returns** -
    Escpater String

7.2.0.6 CLASS FTCacheException

Exception für die Volltextsuche

Declaration

public class FTCacheException **extends** java.lang.Exception

Constructors

- *FTCacheException*

  `public` **FTCacheException**`( )`

  - **Usage**
    - Creates a new instance of `FTCacheException` without detail message.

- *FTCacheException*

  `public` **FTCacheException**`( java.lang.String` **msg** `)`

  - **Usage**
    - Constructs an instance of `FTCacheException` with the specified detail message.
  - **Parameters**
    - `msg` - the detail message.

- *FTCacheException*

  `public` **FTCacheException**`( java.lang.String` **msg**`,`
  `java.lang.Throwable` **cause** `)`

  - **Usage**
    - Constructs an instance of `FTCacheException` with the specified name and cause
  - **Parameters**
    - `name` - the name
    - `cause` - the cause

- *FTCacheException*

  `public` **FTCacheException**`( java.lang.Throwable` **cause** `)`

  - **Usage**
    - Constructs an instance of `FTCacheException` with the specified throwable
  - **Parameters**
    - `cause` - The throwable that was causing the exception

METHODS INHERITED FROM CLASS `java.lang.Exception`

(in 7.3.0.12, page 52)

METHODS INHERITED FROM CLASS `java.lang.Throwable`

(in 7.3.0.12, page 52)

- *fillInStackTrace*

  `public synchronized native Throwable` **fillInStackTrace**`( )`
- *getCause*

  `public Throwable` **getCause**`( )`
- *getLocalizedMessage*

  `public String` **getLocalizedMessage**`( )`
- *getMessage*

  `public String` **getMessage**`( )`
- *getStackTrace*

  `public StackTraceElement` **getStackTrace**`( )`
- *initCause*

  `public synchronized Throwable` **initCause**`( java.lang.Throwable )`
- *printStackTrace*

  `public void` **printStackTrace**`( )`
- *printStackTrace*

  `public void` **printStackTrace**`( java.io.PrintStream )`
- *printStackTrace*

  `public void` **printStackTrace**`( java.io.PrintWriter )`
- *setStackTrace*

  `public void` **setStackTrace**`( java.lang.StackTraceElement [] )`
- *toString*

  `public String` **toString**`( )`

7.2.0.7 CLASS **FTDBCacheOperator**

Hilfsklasse zur Verwaltung des Caches in der DB

DECLARATION

public class FTDBCacheOperator **extends** java.lang.Object

CONSTRUCTORS

- *FTDBCacheOperator*

  `public` **FTDBCacheOperator**`( )`

  - **Usage**
    - Creates a new instance of FTDBCacheOperator

METHODS

- *clearIfTooManyTables*

  `public void` **clearIfTooManyTables**`( int` **maxNumberOfCacheTable**`,`
  `int` **amountBelowThreshold** `)`

  - **Usage**
    - Löscht überzählige Tabellen aus FTCache falls mehr als maximal viele angegeben sind
  - **Parameters**
    - `maxNumberOfCacheTable` - Maximalzahl der zulässigen Cache-Tabellen
    - `amountBelowThreshold` - Determiniert wie weit die Anzahl der Tabellen unter das Limit gedrückt werden soll falls überschritten

---

- *clearTimedOutTables*

  `public void` **clearTimedOutTables**`( int` **tableUseTimeout** `)`

  - **Usage**
    - Löscht Einträge aus FTCache, welche länger als angegeben nicht mehr genutzt wurden
  - **Parameters**
    - `tableUseTimeout` - Timeout für die Tabellen-Nutzung

- *deleteFromCache*

  `public void` **deleteFromCache**`( java.lang.String` **constraint** `)`

  - **Usage**
    - Löscht Einträge aus dem Cache
  - **Parameters**
    - `constraint` - Einschränkung des SQL-Statements welches die zu löschenden
      Cache-Einträge selektiert

- *getFromCache*

  `public String` **getFromCache**`( java.lang.String` **searchstring** `)`

  - **Usage**
    - Schaut ob searchstring bereits im Cache vorhanden ist
  - **Parameters**
    - `searchstring` - Suchstring
  - **Returns** -
    Name der Tabelle, welche die Ergebnisse der Volltextsuche enthält

- *insertIntoCache*

  `public String` **insertIntoCache**`( java.lang.String` **searchstring**`,`
  `java.lang.String` **tableName** `)`

  - **Usage**
    - Fügt eine Cache-Tabelle in die Verwaltungs-DB ein
  - **Parameters**
    - `searchstring` - Suchstring
    - `tableName` - Name der Cachetabelle
  - **Returns** -
    Name der Tabelle, welche die Ergebnisse der Volltextsuche enthält

7.2.0.8 Class **FTSearchEngine**

Beheimatet alle die Suche steuernden Methoden. Diese werden ausgehend von executeFullTextSearch aufgerufen.

Declaration

public class FTSearchEngine **extends** java.lang.Object

Constructors

- *FTSearchEngine*

  `public` **FTSearchEngine**`( )`

  - **Usage**
    - Constructor

METHODS

- *executeFullTextSearch*

  `public String` **executeFullTextSearch**`( java.lang.String` **searchstring**`,`
  `int` **maxNumberOfTuples** `)`

  - **Usage**
    - Führt eine Volltextsuche durch und legt das Ergebnis in einer Tabelle ab
  - **Parameters**
    - `searchstring` - Suchstring
    - `maxNumberOfTuples` - Maximale Anzahl der Suchresultate, die in die Datenbank geschrieben werden sollen
  - **Returns** -
    Name der Tabelle, welche die Ergebnisse der Volltextsuche enthält

7.2.0.9 CLASS FTSearchResult

Repräsentiert ein einzelnes Suchergebnis der Volltextsuche

DECLARATION

public class FTSearchResult **extends** java.lang.Object

CONSTRUCTORS

- *FTSearchResult*

  `public` **FTSearchResult**`( )`

- *FTSearchResult*

  `public` **FTSearchResult**`( int` **id**`,`
  `java.lang.String` **str**`,`
  `int` **ran**`,`
  `boolean` **hmore** `)`

METHODS

- *getFulltext*

  `public String` **getFulltext**`( )`

- *getHasMore*

  `public boolean` **getHasMore**`( )`

- *getLrIdRef*

  `public int` **getLrIdRef**`( )`

- *getRanking*

  `public int` **getRanking**`( )`

- *setFulltext*

  `public void` **setFulltext**`( java.lang.String` **str** `)`

- *setHasMore*

  `public void` **setHasMore**`( boolean` **more** `)`

- *setLrIdRef*

  `public void` **setLrIdRef**`( int` **id** `)`

- *setRanking*

  `public void` **setRanking**`( int` **ran** `)`

7.2.0.10 CLASS IdGenerator

Generiert unique IDs für die Cache-Tabellen der FT-Suche

DECLARATION

public class IdGenerator **extends** java.lang.Object

CONSTRUCTORS

- *IdGenerator*

  `public` **IdGenerator**`( )`

  - **Usage**
    - Creates a new instance of IdGenerator

METHODS

- *getId*

  `public int` **getId**`( )`

  - **Usage**
    - Holt die nächste ID
  - **Returns** -
    Eine einmalige ID

7.2.0.11 CLASS InvalidXMLCharFilter

Filtert XML-feindliche Zeichen (ASCII <= 32 und != 10) aus dem Eingabestream heraus und verhindert damit eine eventuelle Exception vom XML-Parser.

DECLARATION

public class InvalidXMLCharFilter **extends** java.io.FilterInputStream

CONSTRUCTORS

- *InvalidXMLCharFilter*

  `public` **InvalidXMLCharFilter**`( java.io.InputStream` **in** `)`

METHODS

- *read*

  `public int` **read**`( )`

  - **Usage**
    - Reads the next byte of data from this input stream.

- *read*

  ```
  public int read( byte [] b )
  ```

  - **Usage**
    - Reads up to byte.length bytes of data from this input stream into an array of bytes.

- *read*

  ```
  public int read( byte [] b,
  int  off,
  int  len )
  ```

  - **Usage**
    - Reads up to len bytes of data from this input stream into an array of bytes.

METHODS INHERITED FROM CLASS `java.io.FilterInputStream`

- *available*

  ```
  public int available( )
  ```

- *close*

  ```
  public void close( )
  ```

- *mark*

  ```
  public synchronized void mark( int   )
  ```

- *markSupported*

  ```
  public boolean markSupported( )
  ```

- *read*

  ```
  public int read( )
  ```

- *read*

  ```
  public int read( byte []  )
  ```

- *read*

  ```
  public int read( byte [] ,
  int  ,
  int   )
  ```

- *reset*

  ```
  public synchronized void reset( )
  ```

- *skip*

  ```
  public long skip( long   )
  ```

METHODS INHERITED FROM CLASS `java.io.InputStream`

- *available*

  `public int` **available**`( )`
- *close*

  `public void` **close**`( )`
- *mark*

  `public synchronized void` **mark**`( int )`
- *markSupported*

  `public boolean` **markSupported**`( )`
- *read*

  `public abstract int` **read**`( )`
- *read*

  `public int` **read**`( byte [] )`
- *read*

  `public int` **read**`( byte [] , int , int )`
- *reset*

  `public synchronized void` **reset**`( )`
- *skip*

  `public long` **skip**`( long )`

7.3 Package maintain.ftcache

Package Contents *Page*

Classes

Zeigt alle URLls aus den Learningresources mit ihrer dazugehörigen lrid in der URL kodiert an.

7.3.0.12 Class **FTCrawlPageGenerator**

Zeigt alle URLs aus den Learningresources mit ihrer dazugehörigen lrid in der URL kodiert an.

Declaration

public class FTCrawlPageGenerator **extends** HttpServlet

Constructors

- *FTCrawlPageGenerator*

  `public` **FTCrawlPageGenerator**`( )`

Methods

- *destroy*

  `public void` **destroy**`( )`

  - **Usage**
    - Destroys the servlet.

- *doGet*

  `protected void` **doGet**`( HttpServletRequest` **request**`, HttpServletResponse` **response** `)`

  - **Usage**
    - Handles the HTTP `GET` method.
  - **Parameters**
    - `request` - servlet request
    - `response` - servlet response

- *doPost*

  `protected void` **doPost**`( HttpServletRequest` **request**`, HttpServletResponse` **response** `)`

  - **Usage**
    - Handles the HTTP `POST` method.
  - **Parameters**

* `request` - servlet request
* `response` - servlet response

- *getServletInfo*

  `public String` **getServletInfo**`( )`

  - **Usage**
    * Returns a short description of the servlet.

- *init*

  `public void` **init**`( ServletConfig` **config** `)`

  - **Usage**
    * Initializes the servlet.

- *processRequest*

  `protected void` **processRequest**`( HttpServletRequest` **request**`,`
  `HttpServletResponse` **response** `)`

  - **Usage**
    * Processes requests for both HTTP `GET` and `POST` methods.
  - **Parameters**
    * `request` - servlet request
    * `response` - servlet response

METHODS INHERITED FROM CLASS `HttpServlet`

(in 7.2.0.6, page 42)

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Karzauninkat, Stefan: “Die Suchfibel”. Ernst-Klett Verlag Leipzig, April 2002, dritte Auflage.

[2] Aspseek Homepage: http://www.aspseek.org

[3] Aspseek Manpages: http://www.aspseek.org/man

[4] Wagner, Björn: “Entwurf und Implementierung der Anfrageverarbeitung von Meta-Akad”. Projektarbeit, Universität Kaiserslautern, Januar 2003