Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr.-Ing. Dr. h.c. Theo Härder

Möglichkeiten der Identifikation und Notifikation von Änderungen bei Web-basierten Dokumenten durch Hashcodes

Realisierung einer Aktualisierungskomponente im Projekt META-AKAD

Projektarbeit von
Karsten Krennrich
k_krennr@informatik.uni-kl.de

Betreuer:
Dipl.-Inform. Marcus Flehmig
(AG Datenbanken und Informationssysteme)

September 2003

Inhalt

1 Einleitung

1.1 Zum Projekt META-AKAD

Weltweit stellen heute Hochschulen, Bildungseinrichtungen und Verlage Lehr- und Lernmaterialien elektronisch über das Internet zur Verfügung. Auf diese Materialien können sowohl Lehrende als auch Studierende zugreifen, um sie zu ihrem eigenen Gebrauch zu nutzen. Die Suche nach diesen Dokumenten stellt jedoch ein Problem dar. Meist liefern Suchmaschinen nur unvollständige Ergebnisse mit einer geringen Qualität. Das Projekt *Metadatenzugang für akademisches Lehr- und Lernmaterial* (META-AKAD[1]) soll eine komfortable, d. h. eine schnelle und gezielte Suche nach den im Internet verfügbaren Materialien erlauben.

Mit Hilfe von META-AKAD sollen die elektronisch verfügbaren Materialien gesammelt und mit Hilfe eines Metadatensatzes erschlossen werden. Weiterhin sollen die Lehr- und Lernmaterialien nach inhaltlichen und didaktischen Merkmalen beurteilt werden und durch eine Benutzeroberfläche den Lehrenden und Lernenden zur Verfügung gestellt werden. Die Benutzeroberfläche bietet dabei die Möglichkeit eine gezielte Suche, z. B. durch die Angabe von Titel, Autor oder Schlagwörtern, auf den gesammelten Materialien durchzuführen [FH02].

1.2 Einordnung der Arbeit in das Projekt META-AKAD

Das Internet ist kein statisches Objekt, das keinen Veränderungen ausgesetzt ist. Ständig werden neue Informationsmaterialien hinzugefügt, bestehende Inhalte verändert oder ganz entfernt. Das Verändern und Entfernen von Materialien ist ein Problem, das von META-AKAD nicht außer Acht gelassen werden darf. Es ist notwendig die erschlossenen Lehr- und Lernmaterialien ständig auf Veränderung und auf ihre Verfügbarkeit hin zu prüfen.

Im Rahmen dieser Projektarbeit soll eine Aktualisierungskomponente entwickelt werden, die es ermöglicht die gesammelten und bereits erschlossenen Lehr- und Lernmaterialien in festgelegten Zyklen automatisch auf ihre Aktualität und Verfügbarkeit hin zu überprüfen. Dies erleichtert es, die gesammelten Materialien zu kontrollieren und zu verwalten. Weiterhin soll die Aktualisierungskomponente dabei helfen nicht mehr verfügbare oder veraltete Dokumente aus der Menge der erfassten Materialien ausfindig zu machen, um diese eventuell entfernen zu können.

1.3 Motivation

Da sich die Anzahl der über das Internet zur Verfügung gestellten Lehr- und Lernmaterialen ständig vergrößert, wird sich auch die Anzahl der in META-AKAD erschlossenen Dokumente im Laufe der Zeit erhöhen. Aus diesem Grund ist eine manuelle Überprüfung der gesamten Materialien auf deren Aktualität und Verfügbarkeit hin nur mit einem erheblichen Zeit- und Kostenaufwand möglich. Um eine schnelle und vor allem auch zuverlässige Aktualitätskontrolle zu ermöglichen ist eine Software-Komponente, die diesen Dienst automatisiert durchführt, vorteilhaft.

Diese Projektarbeit liefert mit der Entwicklung einer Aktualisierungskomponente einen Beitrag zur Lösung des Problems, die wachsende Menge von erschlossenen Dokumenten auf ihre Aktualität hin zu überprüfen.

[1] http://rzblx1.uni-regensburg.de/metaakad/homepage

1.4 Gliederung

Im zweiten Kapitel werden zunächst die eigentlichen Probleme, die eine Entwicklung einer Aktualisierungskomponente notwendig machen, und die Anforderungen an die entwickelte Komponente beschrieben. Im Anschluss werden in Kapitel drei die Begriffe Hash-Wert, Hash-Funktion und Hash-Verfahren voneinander abgegrenzt und einige Möglichkeiten zur Generierung eines Hash-Wertes vorgestellt. Danach werden in dem gleichen Kapitel die vorgestellten Hash-Verfahren miteinander verglichen, bevor im Anschluss die in diesem Projekt verwendete Methode zur Generierung eines Hash-Wertes ausführlich beschrieben wird. In Kapitel vier wird im Detail auf die Struktur und Implementierung der entwickelten Softwarekomponente eingegangen. Dabei werden zunächst Haupteigenschaften der in diesem Projekt verwendeten Softwarearchitektur J2EE erläutert und deren Anwendung in der Aktualisierungskomponente aufgezeigt.

2 Problembeschreibung und Anforderungen

Das Hauptproblem, das eine Aktualisierungskomponente für META-AKAD notwendig macht, ist die Tatsache, dass im Internet verfügbare Materialien von den jeweiligen Anbietern geändert oder ganz aus dem Internet entfernt werden können. Die Aktualisierungskomponente muss aus diesem Grund in der Lage sein Dokumente zu kontrollieren, um Veränderungen in den bereits erschlossenen Lehr- und Lernmaterialien zu erkennen. Weiterhin müssen diese Materialien kenntlich gemacht und aus der, dem Benutzer zur Verfügung stehenden, Menge an Lehr- und Lernmaterialien entfernt werden. Ebenso muss auch ein nicht mehr verfügbares Material erkannt werden, um es in gleicher Weise, wie das veränderte Dokument, kenntlich machen zu können.

Um eine Veränderung in einem Dokument feststellen zu können, muss das aktuelle Dokument mit einer älteren Version des Dokuments verglichen werden. Eine Möglichkeit dies durchzuführen besteht darin das komplette Lehr- oder Lernmaterial abzuspeichern, um es später bei einer Kontrolle mit der aktuellen Version zu vergleichen. Diese Methode würde aber viel Speicherplatz in Anspruch nehmen und ist aus diesem Grund für das Projekt nicht geeignet. Eine weitere Möglichkeit, die weniger Speicherplatz benötigt und in der Aktualisierungskomponente angewendet wurde, ist die Verwendung von Hash-Verfahren und Hash-Werten, die in Kapitel 3 näher beschrieben werden. Hierbei werden mit Hilfe der Hash-Verfahren Hash-Werte von den jeweiligen Dokumenten generiert, die dann später zum Vergleich herangezogen werden. Diese Hash-Werte sind „digitale Fingerabdrücke“ des jeweiligen Dokuments und signifikant kleiner als das Ursprungsdokument, was zur Folge hat, das sie zum einen weniger Speicherplatz benötigen, und zum anderen einen effizienteren Vergleich möglich machen.

Im Zusammenhang mit Veränderungen in erschlossenen Lehr- oder Lernmaterialien ist jedoch Folgendes zu beachten. Es gibt Änderungen, die den eigentlichen Inhalt des Materials nicht verändern. Zu diesen gehören z. B. Zählerstände von Besucherzählern. Solche Unterschiede sollen die Aktualisierungskomponente nicht dazu veranlassen, das Dokument als verändert zu markieren und aus der verfügbaren Menge von Materialien zu streichen. Die Aktualisierungskomponente muss deshalb so beschaffen sein, dass sie Teile von Dokumenten, die mit dem eigentlichen Inhalt nicht in Verbindung stehen, bei der Kontrolle nicht beachtet. Eine Möglichkeit dies zu gewährleisten, ist bei dem Vergleich nur den reinen textuellen Inhalt des Dokuments zu berücksichtigen.

Die Tatsache, dass ein Lehr- oder Lernmaterial nicht verfügbar ist, stellt uns vor ein weiteres Problem. Es kann sein, dass dieses Material nur temporär, aufgrund eines technischen Problems, nicht erreichbar ist. Ist dies der Fall, so sollte dieses Dokument nicht markiert und auch nicht aus der zur Verfügung stehenden Menge von Materialien entfernt werden. Eine Möglichkeit dieses Problem zu lösen wäre einen Zähler einzubauen. Dieser wird bei jedem Kontrolldurchlauf genau dann erhöht, wenn ein Dokument nicht erreicht werden konnte und wieder zurückgesetzt, sobald das Dokument wieder verfügbar ist. Erreicht der Zähler allerdings einen bestimmten Schwellwert, so kann davon ausgegangen werden, dass das betreffende Lehr- oder Lernmaterial tatsächlich nicht mehr zur Verfügung steht. Das Dokument wird dann als nicht mehr verfügbar markiert und aus der Menge der verfügbaren Dokumente gestrichen.

Eine weitere Anforderung, der die Aktualisierungskomponente genügen soll, ist eine Kontrolle automatisch in einem vorgegebenen Intervall durchführen zu können. Das Intervall soll dabei auf einer graphischen Benutzeroberfläche festgelegt werden können. Weiterhin soll auf der Benutzeroberfläche auch die Möglichkeit bestehen, manuell einen Kontrolldurchlauf zu

starten. Bei den Kontrolldurchläufen soll auch die Option bestehen, jeweils nur eine zufällige Teilmenge der gesammelten Dokumente zu überprüfen. Den Umfang dieser Teilmenge sollte man ebenso wie das Intervall auf der Benutzeroberfläche festlegen.

Wie bereits oben beschrieben besteht die Aufgabe der Aktualisierungskomponente darin, jeweils zwei Versionen eines Dokuments miteinander zu vergleichen. Eine vorteilhafte Möglichkeit dafür, die auch in diesem Projekt Verwendung fand, lässt sich mit Hilfe von Hash-Verfahren und Hash-Werten realisieren, auf die im folgenden Kapitel näher eingegangen wird.

3 Hash-Verfahren

3.1 Begriffsdefinition

Ein Hash-Wert ist ein Wert, der mit Hilfe eines Hash-Verfahrens aus einem beliebigen Eingabedokument generiert wird. Das Hash-Verfahren beinhaltet wiederum eine Hash-Funktion, die aus Teilen des Eingabedokuments Zwischenwerte berechnet. Ein Beispiel für eine solche Hash-Funktion ist die *modulo*-Operation. So wäre der Hash-Wert für die Zahl 32, der mit Hilfe der Hash-Funktion „mod 10“ generiert wird 2. Das Hash-Verfahren setzt dann diese Zwischenwerte mit Hilfe weiterer Operationen, z. B. der Addition, zu einem einzigen Wert zusammen. Das Beispiel zeigt auch, dass der Hash-Wert kürzer als das Eingabedokument ist.
Der generierte Hash-Wert wird auch als digitaler Fingerabdruck des Dokuments bezeichnet. Die Hash-Funktion eines Hash-Verfahrens, die den wichtigsten Teil bei der Berechnung des „Fingerabdrucks“ durchführt, sollte nach Möglichkeit so beschaffen sein, dass das Hash-Verfahren für unterschiedliche Dokumente nicht den gleichen Hash-Wert als Ergebnis liefert. Ist dies gewährleistet so heißt die Funktion „kollisionsfrei“. Weiterhin soll es auch nicht möglich sein aus einem generierten Hash-Wert auf den ursprünglichen Klartext des Dokuments zu schließen. Hash-Verfahren, die diese Eigenschaft erfüllen werden „Einweg-Hash-Verfahren“ genannt. Drei Verfahren, die diesen Eigenschaften genügen, werden in Abschnitt 3.2 beschrieben.

Hash-Werte werden hauptsächlich für digitale Signaturen und zur Integritätsprüfung von Dokumenten verwendet, d. h. man will sicherstellen, dass ein empfangener Text auf dem Weg vom Sender zum Empfänger nicht verändert wurde. Dazu generiert der Sender den Hash-Wert des Klartextes, den er versenden möchte. Dieser Wert wird ebenfalls dem Empfänger zugesendet. Der Empfänger wiederum generiert von dem empfangenen Text den Hash-Wert und vergleicht diesen mit dem Hash-Wert des Senders. Stimmen die beiden Werte überein, so kann der Empfänger mit hoher Wahrscheinlichkeit davon ausgehen, dass der Text auf dem Weg vom Sender zum Empfänger nicht verändert wurde [JL98].

3.2 Hash-Verfahren

3.2.1 MD-5 (Message–Digest)

Das Hash-Verfahren MD-5 wurde von Professor Ronald L. Rivest entwickelt. Es zählt zu den so genannten „Einweg-Hash-Verfahren“. Der Algorithmus berechnet einen 128-Bit Hash-Wert aus einer Eingabedatei, z. B. einer Textdatei [DT02].

3.2.2 RIPEMD-160

RIPEMD-160 wurde von Hans Dobbertin, Anton Bosselaers und Bart Preneel entwickelt. Das Verfahren gehört ebenso wie MD-5 zu den „Einweg-Hash-Verfahren“. Der Algorithmus berechnet jedoch im Gegensatz zum MD-5 Hash-Verfahren einen 160-Bit Hash-Wert aus einer beliebig langen Eingabe. Durch den längeren Hash-Wert benötigt man mit einer Brute-Force-Methode 2^{160} Operationen um einen Hashwert aus einer Nachricht zu erzeugen. Im Vergleich dazu benötigt man bei einem 128-Bit Hash-Wert 2^{128} Operationen. Aus diesem Grund bietet das Verfahren eine erhöhte Sicherheit und man beabsichtigt das MD-5 Hash-Verfahren durch das Verfahren RIPEMD-160 zu ersetzen. Ein weiterer Aspekt, der für das Hash-Verfahren RIPEMD-160 spricht, ist, dass man bei der Berechnung von Hash-Werten mit Hilfe von MD-

5 auf Kollisionen gestoßen ist, d. h. das Verfahren lieferte für unterschiedliche Eingabedokumente als Ergebnis den gleichen Hash-Wert [AB99, JL98].

3.2.3 SHA-1

Das Hash-Verfahren SHA-1 wurde von der National Security Agency (NSA) entwickelt. Dieses Verfahren zählt wie schon die beiden zuvor beschriebenen Verfahren zu den „Einweg-Hash-Verfahren". Der Algorithmus berechnet einen 160-Bit Hash-Wert aus einer Eingabe, deren Länge auf 2^{64} Bits beschränkt ist. Dieses Verfahren gilt, wie auch das zuvor beschriebene Verfahren RIPEMD-160, als sicher [DT02, PDA97].

3.3 Verwendete Methode MD-5

Für die Aktualisierungskomponente, die im Rahmen dieser Projektarbeit entwickelt wurde, wird das MD-5 Hashverfahren verwendet. Die Gründe dafür sind folgende:

- Die Leistung der Aktualisierungskomponente ist aufgrund der großen Anzahl der zu prüfenden Dokumente ein wichtiger Gesichtspunkt, der nicht außer Acht gelassen werden darf. Ein Leistungstest zeigt, dass der Algorithmus des RIPEMD-160 Hash-Verfahrens auf einem Pentium Prozessor mit 90 MHz 45,5 MBits/sec an Eingabedaten verarbeitet. Der Algorithmus des MD-5 Hash-Verfahrens konnte beim gleichen Test auf dem gleichen Prozessor 136,7 MBits/sec verarbeiten, d. h. das MD-5 Hash-Verfahren ist bei diesem Test dreimal schneller als das RIPEMD-160 Hash-Verfahren. Diese Leistungsgründe sind ausschlaggebend, dass RIPEMD-160 in diesem Projekt nicht verwendet wird [AB99].
- Das Hash-Verfahren SHA-1 bietet zwar durch seinen 160-Bit langen Hash-Wert eine erhöhte Sicherheit, in dem Sinne, dass die Wahrscheinlichkeit der Generierung zweier identischer Hash-Werte für zwei verschiedene Dokumente gering ist und eine Rücktransformation eines Klartextes aus einem Hash-Wert 2^{160} Operationen benötigt, jedoch übersteigt die Komplexität des Algorithmus des SHA-1 Hash-Verfahrens die des MD-5 Hash-Verfahrens und würde dadurch zu einer geringeren Leistung führen. Weiterhin handelt es sich bei den Dokumenten, die mit Hilfe des Hash-Verfahrens in der Aktualisierungskomponente „komprimiert" werden sollen, auch nicht um geheime, sondern um öffentlich zugängliche Klartexte. Die Hash-Werte müssen also nicht vor Angreifern geschützt werden. Aus diesen Gründen wird auch das Hash-Verfahren SHA-1 in dieser Projektarbeit nicht angewendet.

Die genannten Gründe verdeutlichen weshalb in diesem Projekt zur Entwicklung einer Aktualisierungskomponente das MD-5 Hashverfahren und nicht eines der anderen beiden Verfahren eingesetzt wird. Im Folgenden wird jetzt auf das verwendete Verfahren detailliert eingegangen.

3.4 Vorgehensweise von MD-5

Die Abbildung 3.1 verdeutlicht die Struktur des Hash-Verfahrens MD-5. Die genaue Vorgehensweise des Algorithmus wird im Folgenden ausführlich beschrieben.

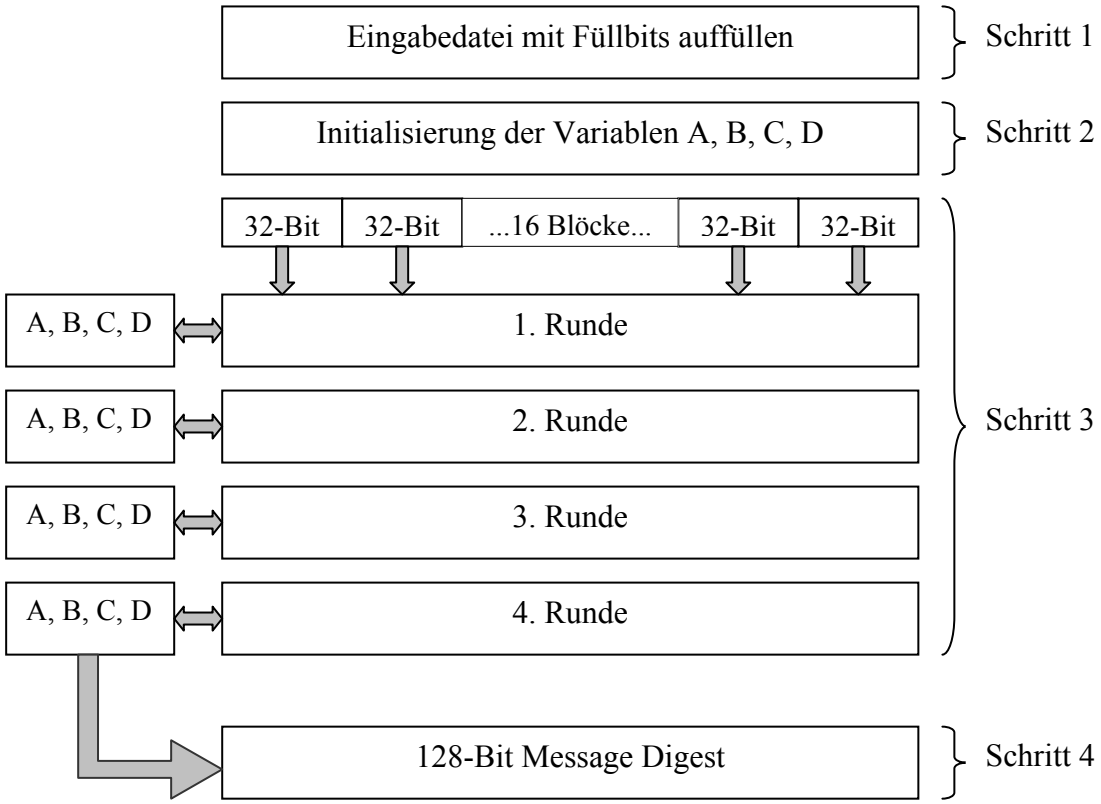


Abbildung 3.1: Struktur von MD-5

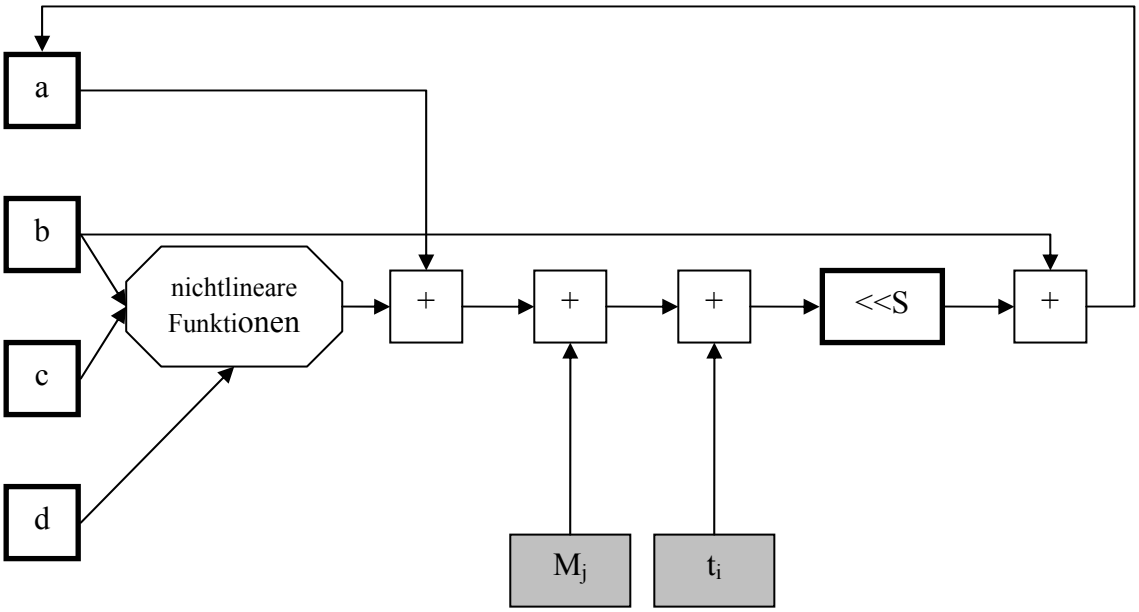
Wie die Abbildung 3.1 deutlich macht, lässt sich der Ablauf des Algorithmus in vier Teilschritte unterteilen, die nun näher erläutert werden.

1. Schritt:
Zu Beginn wird die Eingabedatei zunächst mit Füll-Bits aufgefüllt, so dass die Länge der Datei ein Vielfaches von 512 Bit beträgt. Diese Füll-Bits haben alle den Wert 0. Das erste Füll-Bit wird jedoch durch eine 1 ersetzt. Die Datei, die daraus resultiert beginnt jetzt mit „1000000...“. Danach werden die letzten 64 Füllbits durch eine 64-Bit Binärzahl ersetzt. Diese Binärzahl hat als Wert die Länge der Eingabedatei. Das Resultat dieses Ersetzungsvorgangs ist dann wieder eine Eingabedatei deren Länge ein Vielfaches von 512 Bit ist.

2. Schritt:
In diesem Schritt werden die so genannten Verkettungsvariablen A, B, C und D mit Hexadezimalzahlen initialisiert. Die Werte der Zahlen sind nicht festgelegt und können willkürlich gewählt werden. Ein Beispiel für eine Initialisierung wäre z. B.:
A = 0x01234567
B = 0x89abcdef
C = 0xfedcba98
D = 0x76543210

Diese Hexadezimalzahlen sind später die ersten Eingabeparameter für die Hash-Funktionen. Bei MD-5 sind dies vier nichtlineare Funktionen. Sie werden im 3. Schritt näher beschrieben.

3. Schritt:
In Schritt Nummer drei findet die eigentliche Berechnung des Hash-Wertes mit Hilfe der Hash-Funktionen statt. Es wird zunächst ein 512 Bit langer Teileingabeblock aus der gesamten Eingabedatei „entnommen“. Bei diesem Schritt wird deutlich weshalb die Länge der Datei am Ende von Schritt eins ein Vielfaches von 512 ist. Es ist damit gewährleistet, dass kein Teileingabeblock kürzer als 512 Bit sein kann. Danach wird dieser 512 Bit lange Teileingabeblock in 16 jeweils 32 Bit große Blöcke unterteilt. Dieser unterteilte Eingabeblock durchläuft im Anschluss vier Runden (siehe Abb.3.1), die alle nach folgendem Schema ablaufen:



nichtlineare Funktionen:

Runde 1: $F(X,Y,Z) = (X \wedge Y) \vee (\overline{X} \wedge Z)$

Runde 2: $G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \overline{Z})$

Runde 3: $H(X,Y,Z) = (X \oplus Y \oplus Z)$

Runde 4: $I(X,Y,Z) = Y \oplus (X \vee \overline{Z})$

Abbildung 3.2: Rundendurchlauf des MD-5 Algorithmus

Für die Berechnung des Hash-Werts werden noch die zusätzlichen Variablen M_j, t_i und eine Anzahl von Bits, um die die jeweils berechnete Summe nach links rotiert wird benötigt. M_j (j = 0, ..., 15) ist jeweils einer der 16 32-Bit großen Eingabeblöcke, in die der 512-Bit große Teileingabeblock unterteilt wurde. t_i ist eine additive Konstante, die sich in den einzelnen Runden folgendermaßen berechnet:

$t_i = 2^{32} * |\sin(i)|$ (Runde 1: i = 1, ..., 16
Runde 2: i = 17, ..., 32
Runde 3: i = 33, ..., 48
Runde 4: i = 49, ..., 64)

i nimmt in einer Runde genau 16 Werte an, weil eine Runde aus 16 Schritten besteht (16 32-Bit Blöcke).
Die Anzahl der Bits, um die die Summe jeweils nach links rotiert wird, ist in der folgenden Tabelle aufgelistet.

	16 Schritte je Runde															
Runde	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	7	12	17	22	7	12	17	22	7	12	17	22	7	12	17	22
2	5	9	14	20	5	9	14	20	5	9	14	20	5	9	14	20
3	4	11	16	23	4	11	16	23	4	11	16	23	4	11	16	23
4	6	10	15	21	6	10	15	21	6	10	15	21	6	10	15	21

Tabelle 3.1: Anzahl der Bits um die rotiert wird

Zur Einfachheit wird im Folgenden nur die Runde 1 in Pseudocode beschrieben. Runde 2, 3 und 4 laufen analog ab, mit dem einzigen Unterschied, dass in jeder der vier Runden unterschiedliche Hash-Funktionen verwendet werden. In Runde eins wird Funktion F, in Runde zwei G, in Runde drei H und in Runde vier I verwendet (siehe Abb. 3.2).

Ablauf der Runde 1:
Zunächst speichere die ersten Verkettungsvariablen A, B, C und D in a, b, c und d:

a = A
b = B
c = C
d = D

Der weitere Verlauf einer Runde geschieht dann in insgesamt 16 Schritten:

1. $a = a + F(b, c, d) + M_0 + t_1$;
   rotiere a um 7 Stellen nach links (siehe Tabelle 3.1);
   a = a + b;
   (Dieser erste Schritt wird durch Abbildung 3.2 wiedergespiegelt.)

2. $d = d + F(a, b, c) + M_1 + t_2$;
   rotiere d um 12 Stellen nach links;
   d = d + a;

3. $c = c + F(d, a, b) + M_2 + t_3$;
   rotiere c um 17 Stellen nach links;
   c = c + d ;

4. $b = b + F(c, d, a) + M_3 + t_4$;
   rotiere b um 22 Stellen nach links;
   b = b + c;

Die restlichen 12 Schritte der Runde 1 laufen analog zu den beschriebenen ersten vier Schritten ab. Der einzige Unterschied besteht darin, dass die Variablen M und t in den verbleibenden Schritten immer neue Werte annehmen.

Die jeweils resultierenden Werte von a, b, c und d nach Beendigung einer Runde werden als Eingabeparameter für die darauf folgende Runde verwendet.

Nachdem der erste 512-Bit lange Teileingabeblock die vier Runden durchlaufen hat, wird der nächste 512-Bit lange Teileingabeblock aus der Eingabedatei entnommen. Dieser durchläuft dann in gleicher Weise die vier Runden, wobei die Eingabeparameter a, b, c und d die Resultate der Runde 4 des vorherigen Teileingabeblocks sind. Die-

se Prozedur wird solange wiederholt bis alle Teileingabeblöcke die vier o. g. Runden durchlaufen haben. Die Gefahr, dass der letzte Teileingabeblock kürzer als 512-Bit ist, besteht nicht, da zu Anfang die Eingabedatei aufgefüllt wurde, so dass sie ein Vielfaches von 512-Bit groß ist.

4. Schritt:
Die Variablen a, b, c und d, die aus der vierten Runde des letzten 512-Bit langen Teileingabeblock resultieren bilden den 128-Bit langen Hash-Wert, den es zu berechnen galt [JL98].

Nachdem die, in der entwickelten Aktualisierungskomponente verwendete, Technik zur Berechnung der Hash-Werte vorgestellt wurde, wird im folgenden Abschnitt auf die Softwarearchitektur, die in diesem Projekt Verwendung fand, und auf die konkrete Implementierung der Aktualisierungskomponente eingegangen.

4 Implementierung

4.1 Verwendete Softwarearchitektur Java 2 Enterprise Edition

Das Projekt META-AKAD wurde auf der Basis der Softwarearchitektur Java 2 Enterprise Edition implementiert. Die Aktualisierungskomponente wurde auch mit Hilfe der gleichen Softwarearchitektur implementiert. Aus diesem Grund werden im Folgenden die Struktur und die wichtigsten Technologien der Architektur beschrieben.

4.1.1 Struktur der Softwarearchitektur

Die Java 2 Enterprise Edition[2] (J2EE) bietet ein Modell zur Entwicklung von verteilten Anwendungen. Durch die wohl definierten Komponenten, auf denen diese Architektur beruht, macht sie eine standardisierte und vereinfachte Entwicklung von verteilten Anwendungen, die den heutigen Anforderungen des Internets und der Wettbewerbssituation genügen, möglich. Der komponentenbasierte Aufbau der Softwarearchitektur J2EE, die in dem Projekt META-AKAD Verwendung findet, ist in der folgenden Abbildung 4.1 beschrieben.

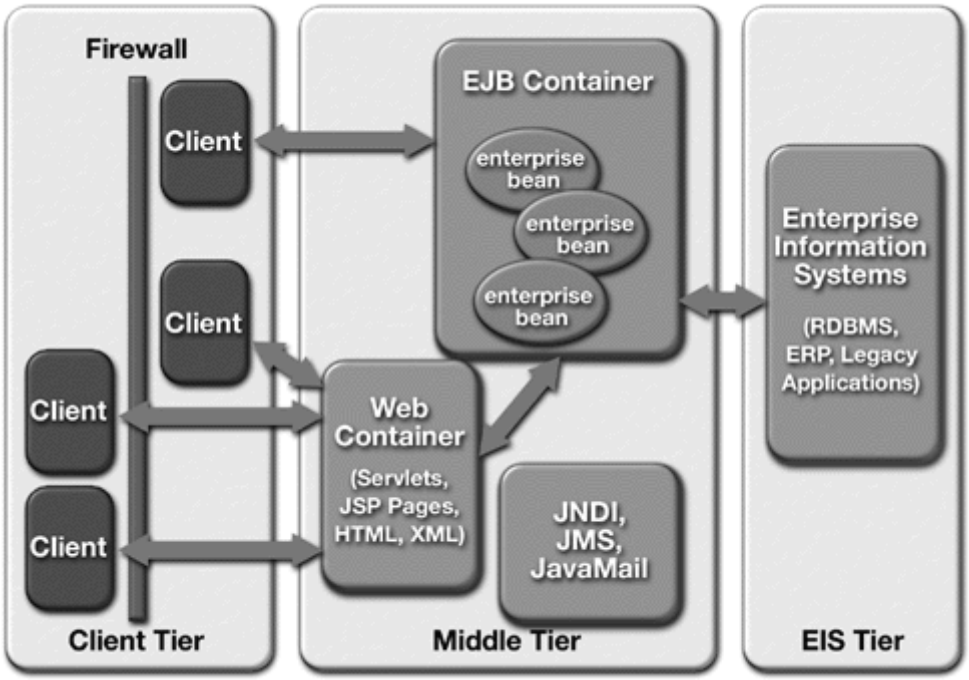


Abbildung 4.1: J2EE Architektur

Wie die Abbildung zeigt besteht die Architektur aus einem dreischichtigen Modell für verteilte Anwendungen. Durch diese dreischichtige Architektur ist es möglich die Präsentation, die Geschäftslogik und die Datenhaltung voneinander abzugrenzen. Diese Möglichkeit ist insbesondere bei einer Systemerweiterung vorteilhaft. Man kann einzelne Schichten unabhängig von den anderen Schichten erweitern. Die drei Schichten, die die Architektur spezifiziert sind *Client Tier*, *Middle Tier* und *EIS Tier* (Enterprise Information System):

- Die *Client Tier* unterstützt eine Vielzahl verschiedener sowohl innerhalb als auch außerhalb einer Firewall gelegene Clients.

[2] http://java.sun.com/j2ee/

- Die *Middle Tier* stellt den Clients Dienste durch den Web Container und den EJB (Enterprise Java Bean) Container zur Verfügung. Container sind Laufzeitumgebungen und stellen den jeweiligen Komponenten, die sich in den Containern befinden, standardisierte Dienste bereit. Der Web Container dient dabei als Laufzeitumgebung für die Servlets und die Java Server Pages (JSP). Sie realisieren die Präsentation der Daten. Der EJB Container stellt die Laufzeitumgebung für die Enterprise Java Beans dar. In ihnen ist die eigentliche Geschäftslogik der Dienste, die einem Client zur Verfügung gestellt werden können, implementiert.
- Die dritte Schicht, das so genannte Enterprise Information System (EIS), ist für die Verwaltung der Daten, z. B. in relationalen Datenbanken, zuständig. Auf diese Daten können EJBs über standardisierte APIs zugreifen.

4.1.2 Technologien der Architektur

Die plattformspezifischen Technologien von J2EE lassen sich in die drei Kategorien Komponenten, Dienste und Kommunikation unterteilen, die im Folgenden genauer erläutert werden.

4.1.2.1 Komponenten

Web-Komponenten

- Servlets:
  Ein Servlet ist ein Server-seitiges Programm, das durch eine HTTP-Anfrage aktiviert wird. Das Programm generiert dann, eventuell mit Hilfe von Daten aus der Datenbank, ein HTML-Dokument, das es an den Client als zurücksendet.
- Java Server Pages (JSP):
  Java Server Pages sind HTML-Dokumente mit eingebettetem auf Java basierendem Code, mit dessen Hilfe der Inhalt der Seiten dynamisch generiert wird. Java Server Pages bieten im Vergleich zu Servlets den Vorteil, dass sie eine bessere Trennung zwischen Präsentation und Geschäftslogik ermöglichen.

EJB (Enterprise Java Bean)-Komponenten

- Session EJB:
  Ein Session EJB wird benötigt, wenn ein Client die Dienste eines EJB nutzen möchte. Es existieren zwei Arten von Session EJBs:
  - *Stateful Session EJBs* halten ihren Zustand, der durch Instanzvariablen repräsentiert wird, solange bis die Session beendet ist, d. h. der Zustand kann auch über mehrere Methodenaufrufe hinweg existieren.
  - *Stateless Session EJBs* halten die Instanzvariablen und damit ihren Zustand nur solange, wie ein Methodenaufruf dauert.
- Entity EJBs:
  Ein Entity EJB ist ein Objekt, das persistente Daten der EIS-Ebene repräsentiert. Sie werden als Geschäftsobjekte bezeichnet.
- Message-Driven EJBs:
  Ein Message-Driven EJB erlaubt einem Client asynchron die Dienste, die durch die EJBs zur Verfügung gestellt werden, aufzurufen. Ein Client kann jedoch nicht direkt auf ein Message-Driven EJB zugreifen. Das Message-Driven EJB kann nur mittels einer asynchronen Nachricht aus einer Java Message Service (JMS) Queue aktiviert werden. JMS ist eine Java-API zum Senden und Empfangen von Nachrichten.

4.1.2.2 Dienste

- Java Data Base Connectivity API:
  Die Java Data Base Connectivity (JDBC) API ermöglicht den Zugriff auf Tabellen der Datenquelle. Die API ist von der Datenbank unabhängig.

- Java Transaction API:
  Die Java Transaction API (JTA) erlaubt der Anwendung Transaktionen unabhängig von ihrer spezifischen Implementierung zu kontrollieren.

- Java Naming und Directory Interface:
  Das Java Naming und Directory Interface (JNDI) gibt der Anwendung die Möglichkeit Objekte zu speichern und zu suchen.

- J2EE Connector Architectur:
  Die J2EE Connector Architectur dient dazu die J2EE Plattform mit einem Enterprise Information System zu integrieren.

- Java API für XML Processing:
  Die Java API für XML Processing (JAXP) unterstützt die Verwaltung und Verarbeitung von XML Dokumenten.

4.1.2.3 Kommunikationstechnologien

Die Kommunikationstechnologien machen es möglich, dass der Client und der Server einer verteilten Anwendung miteinander kommunizieren können.

- Hyper Text Transfer Protocol:
  Das Hyper Text Transfer Protocol (HTTP) wird dazu verwendet, Anfragen vom Client zum Server und Antworten in umgekehrter Richtung zu übermitteln.

- Remote Method Invocation:
  Remote Method Invocation ist eine Menge von APIs mit deren Hilfe es möglich ist verteilte Anwendungen zu entwickeln. Durch die Schnittstellen können Objekte, die auf verschiedenen Maschinen existieren, miteinander kommunizieren.

- Java Message Service API:
  Die Java Message Service (JMS) API dient dazu, auf Enterprise Messaging Systeme zuzugreifen und um nachrichtenbasierte J2EE-Anwendungen (Message-Driven Beans) zu einsetzen zu können.

- Java Mail API:
  Die Java Mail API enthält Klassen und Schnittstellen mit deren Hilfe man E-Mails verschicken und empfangen kann [J2EE02].

4.2 Beschreibung der entwickelten Aktualisierungskomponente

Nachdem in Kapitel 4.1 die Softwarearchitektur beschrieben wurde mit deren Hilfe die Aktualisierungskomponente in diesem Projekt implementiert wurde, wird im Folgenden konkret die Implementierung und die genaue Arbeitsweise der im Rahmen dieser Projektarbeit entwickelten Komponente zur Identifikation und Notifikation von Änderungen bei Web-basierten Dokumenten näher beschrieben. Weiterhin werden auch die Realisierung der automatischen Kontrolle und die Toleranz der Aktualisierungskomponente erläutert.

4.2.1 Struktur der Aktualisierungskomponente

Bevor die genaue Vorgehensweise der Aktualisierungskomponente erläutert wird, wird zunächst ein Überblick über den strukturellen Aufbau der Komponente in Abbildung 4.2 gegeben.

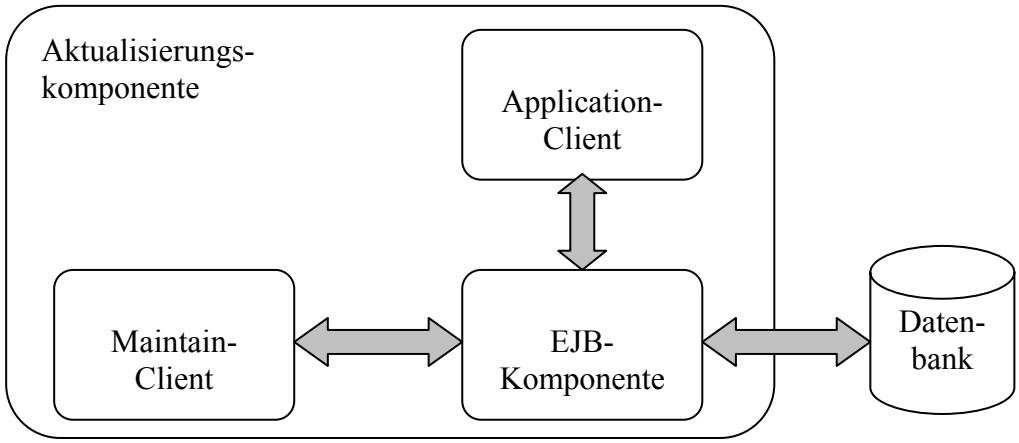


Abbildung 4.2: Struktureller Aufbau der Aktualisierungskomponente

Die Abbildung zeigt, dass die Aktualisierungskomponente im Wesentlichen aus drei Teilkomponenten aufgebaut ist: Dem Maintain-Client, der EJB-Komponente und dem Application-Client.

Die Komponente Maintain-Client stellt die graphische Schnittstelle für den Benutzer zur Verfügung. Mit Hilfe dieser können, wie in den Anforderungen in Kapitel zwei beschrieben, das Intervall in dem eine Überprüfung automatisch durchgeführt werden soll und der prozentuale Anteil der zu prüfenden Dokumente festgelegt werden.

Die EJB-Komponente beinhaltet die Geschäftslogik der Aktualisierungskomponente. In ihr findet die eigentliche Aktualitätskontrolle statt. Sie steht in Verbindung mit der Datenbank. Dadurch kann sie dem Maintain-Client die aktuellen Werte des prozentualen Anteils von Dokumenten, die automatisch überprüft werden sollen und die Daten über das Intervall zur Verfügung stellen, um sie dem Benutzer anzeigen zu können. Weiterhin benötigt die EJB-Komponente auch die gespeicherten Hash-Werte der erfassten Dokumente aus der Datenbank, um diese mit den neu generierten Hash-Werten vergleichen zu können. Die Art und Weise wie die EJB-Komponente die Aktualitätsprüfung durchführt und welche EJBs bzw. Java Beans welche Aufgaben übernehmen wird in Abschnitt 4.2.2 erläutert.

Für die automatisierte Überprüfung der gesammelten und erschlossenen Lehr- und Lernmaterialien in festgelegten Zykle, ist die dritte Komponente, der Application-Client, zuständig. Das automatische Starten der Aktualitätskontrolle wurde dabei mit Hilfe von Threads realisiert. Es wäre auch denkbar, diesen Mechanismus direkt in der EJB-Komponente zu realisie-

ren und somit den Application-Client einzusparen. Dies widerspricht jedoch der *Enterprise Java Beans Spezifikation*, in der ausdrücklich verboten ist Threads durch EJBs zu steuern oder zu kontrollieren. Auch wäre es eventuell möglich gewesen zu diesem Zweck, die in Kapitel 4.1.2.1 beschriebenen, Message-Driven EJBs zu verwenden. Diese erlauben es asynchron die in den EJBs implementierten Methoden aufzurufen. Das Problem bei Message-Driven EJBs besteht jedoch darin, das es in der Version EJB 2.0 nicht möglich ist einen solchen Methodenaufruf zu einer festgelegten Zeit durchzuführen. Genau diese Möglichkeit wäre für das automatische Starten der Kontrolle nötig gewesen. Aus diesem Grund wurde für die Entwicklung der Aktualisierungskomponente auf Message-Driven EJBs verzichtet und Threads eingesetzt. Der Mechanismus für das automatische Starten der Kontrolle ist in der Klasse IntervalThread implementiert. Dabei werden aus den Daten, die in der Benutzeroberfläche des Maintainclient eingegeben werden können (siehe 7.1), die Zeitspannen berechnet, die der Thread jeweils warten muss bevor dieser eine neue Kontrolle aktiviert. Die Daten, die für die Berechnung relevant sind, sind der angegebene Tagesrhythmus und die Uhrzeit zu der jeweils eine Kontrolle stattfinden soll [EJBS 01].

4.2.2 Vorgehensweise der entwickelten Aktualisierungskomponente

Nachdem die Struktur der entwickelten Aktualisierungskomponente erläutert wurde, wird nun auf die genaue Vorgehensweise eingegangen. Der genaue Ablauf eines Kontrolldurchlaufs der Aktualisierungskomponente, in dem die in der Datenbank erfassten Dokumente auf ihre Aktualität hin überprüft werden, wird in Abbildung 4.3 dargestellt.

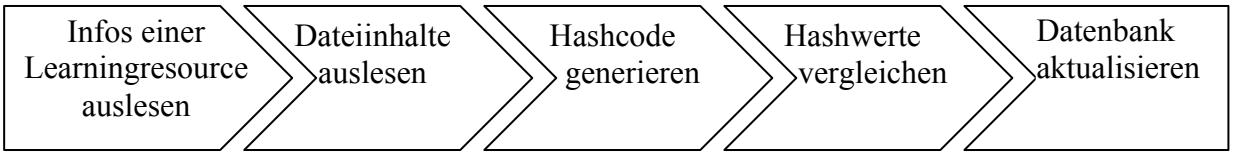


Abbildung 4.3: Vorgehensweise der Aktualisierungskomponente

Aus der Abbildung 4.3 geht hervor, dass ein Kontrolldurchlauf insgesamt fünf Hauptschritte umfasst, die in den nächsten Abschnitten erläutert werden:

1. Schritt:
Zu Beginn einer Überprüfung durch die Aktualisierungskomponente müssen zunächst eine URL, eine ID und ein zugehöriger Hash-Wert der erfassten Dokumente (Learningresource) aus der Datenbank ausgelesen werden. Die URL gibt die Adresse an, unter der das jeweilige zu überprüfende Dokument im Internet zu finden ist. Die ID ist eine eindeutige Kennung, die jedes Dokument besitzt, um es identifizieren zu können. Sie ist später bei der Aktualisierung der Daten in der Datenbank nötig (siehe 4.2.1.5). Der zugehörige Hash-Wert wird zum späteren Vergleich mit dem neu generierten Hash-Wert des aktuellen Materials benötigt (siehe 4.2.1.4). Die im ersten Schritt beschriebenen Aktionen werden von dem Session EJB *ETControllerEJB* ausgeführt. Das genannte Session EJB ist Teil der Geschäftslogik und befindet sich in der EJB-Komponente (siehe Abbildung 4.2).

2. Schritt:
Um einen Hashcode generieren zu können werden in diesem Schritt die Dateien, die sich hinter den URLs befinden ausgelesen. Hier kommt die Option, nur einen zufälligen Teil der Dokumentenmenge zu überprüfen, zum Tragen. Aus dem angegeben prozentualen Anteil wird ein Wert berechnet, der festlegt welche URLs und damit auch welche Dokumente der Überprüfung unterzogen werden. Beim Auslesen der jeweiligen Dateien werden die empfangen Daten in eine Zeichenkette umgewandelt und diese dann als Eingabeparameter dem MD-5 Algorithmus übergeben. Die Berechnung des o. g. Werts, das Auslesen der Datei und die

Umwandlung der Daten wird in der Klasse *ChangeController* durchgeführt, die auch einen Teil der Geschäftslogik darstellt.

3. Schritt:
Die Generierung des Hashwerts wird mit Hilfe des Algorithmus des MD-5 Hash-Verfahrens durchgeführt. Hierbei wird aus der vom *ChangeController* generierten Zeichenkette ein 128 Bit langer Hash-Wert errechnet. Der genaue Ablauf des Algorithmus ist in Kapitel 3.2 beschrieben.

4. Schritt:
Nachdem der aktuelle Hash-Wert berechnet wurde, wird dieser nun mit dem alten Hash-Wert des erschlossenen Lehr- oder Lernmaterials aus der Datenbank verglichen. Wird bei dem Vergleich eine Abweichung festgestellt, so werden die ID und der aktuelle Hash-Wert des gerade geprüften Dokuments zwischengespeichert. Diese beiden Daten sind notwendig, um später die Datenbank aktualisieren zu können und sie damit konsistent zu halten. Den Vergleich des neuen mit dem alten Hash-Wert wird auch von der Klasse *ChangeController* übernommen. Die Schritte 2 bis 4 werden anschließend für alle zu überprüfenden Dokumente wiederholt.

5. Schritt:
Der letzte Schritt, den die Aktualisierungskomponente während eines Kontrolldurchlaufs durchführt ist die Aktualisierung der Datenbank. Mit Hilfe der IDs und der neu generierten Hash-Werte, die beim Vergleich in Schritt vier zwischengespeichert wurden, wird die Datenbank aktualisiert. Es wird vermerkt, dass sich die Dokumente mit den jeweiligen IDs verändert haben. Dadurch werden sie vorläufig aus der Dokumentenmenge entnommen, die bei einer Suchanfrage durch einen Benutzer durchsucht werden. Zudem wird noch der alte Hashwert des jeweiligen Dokuments durch den gerade neu generierten Hashwert ersetzt, sodass die Datenbank wieder konsistent ist.

4.2.3 Toleranz bei der Kontrolle der erfassten Materialien

Nachdem der Ablauf eines Kontrolldurchlaufs erklärt wurde wird jetzt auf das Problem, dass nicht jede Veränderung von der Aktualisierungskomponente auch wirklich als Veränderung angesehen werden soll, eingegangen und die in diesem Projekt verwendete Lösung aufgezeigt.

Das Problem besteht darin, dass nicht jede Abweichung des aktuellen Dokuments zum Ursprungsdokument auch eine Veränderung darstellt, die einen Ausschluss des Dokuments aus der Menge, der dem Benutzer zur Verfügung stehenden Materialien, rechtfertigt. Deshalb muss auch die Aktualisierungskomponente gewisse Veränderungen tolerieren können.

Im Gegensatz zu statischen Dokumenten, wie PDF- oder PS-Dateien, können auf dynamischen Webseiten (HTML-Dateien) Änderungen entstehen, die durch Java-Skripte oder durch sich ständig ändernde dynamische Elemente, wie z. B. Zähler, die die Anzahl der Besucher der Seite anzeigen, hervorgerufen werden. Solche Veränderungen, die den eigentlichen thematischen Inhalt oder die Verfügbarkeit des Dokuments nicht beeinflussen, sollen auch von der Aktualisierungskomponente nicht erfasst werden. Aus diesem Grund werden HTML-Dateien vor der eigentlichen Aktualitätskontrolle durch einen Parser bearbeitet. Der Parser entfernt aus der jeweiligen Datei sämtliche Tags und eingebetteten Code-Elemente wie z. B. Java Skript oder PHP. Als Ausgabe wird nur der reine textuelle Inhalt des Dokuments geliefert. Dieser wird dann, wie im o. g. Schritt drei eines Kontrolldurchlaufs beschrieben, dem MD-5 Hash-Algorithmus übergeben. Damit wird gewährleistet, dass z. B. aufgrund eines veränderten Zählerwertes eines Besucherzählers auf der Seite keine Veränderung durch die Aktualisierungskomponente realisiert wird. Das Dokument bleibt damit in der Menge der Lehr- und Lernmaterialien, die bei einer Suchanfrage durch den Benutzer durchsucht werden kann.

5 Zusammenfassung

In dieser Projektarbeit, deren Ziel die Entwicklung einer Aktualisierungskomponente für das Projekt META-AKAD war, wurden zunächst die Probleme, die eine Aktualisierungskomponente notwendig machen und die Anforderungen, die sich dadurch ergeben, aufgezeigt. Nachdem der Einsatz von Hash-Werten in diesem Projekt als sinnvoll erachtet wurde, wurden die drei Verfahren MD-5, RIPEMD-160 und SHA-1 zur Generierung von Hash-Werten vorgestellt. Anschließend wurden die Gründe aufgeführt, die dafür verantwortlich sind, dass für die Aktualisierungskomponente das MD-5 Hash-Verfahren zum Einsatz kam. Danach wurde die Arbeitsweise des Algorithmus des verwendeten Verfahrens genau beschrieben.

Der eigentliche Grund für die Entwicklung von Funktionen zur Generierung von Hash-Werten wurde in Kapitel 3.1 deutlich gemacht. Mit Hilfe der Hash-Werte will man die Sicherheit bei der Übertragung von Daten erhöhen. Durch den Einsatz von Hash-Verfahren lässt sich eine Integritätssicherung insofern realisieren, dass man anhand eines Vergleichs der beiden Hash-Werte, die Sender und Empfänger unabhängig voneinander berechnen, erkennen kann, ob ein Dokument auf dem Weg vom Sender zum Empfänger verändert wurde.

Die Hauptaufgabe der Aktualisierungskomponente besteht darin, zwei Dokumente miteinander zu vergleichen und existierende Unterschiede oder Veränderungen zu erkennen. Eine Möglichkeit einen solchen Vergleich durchzuführen wäre, die beiden Dateien vollständig auf Unterschiede hin zu untersuchen. Da die Dateien unter Umständen aber sehr groß sein können, kann ein Vergleich auf diese Weise mit einem erheblichen Aufwand verbunden sein. Weiterhin müssen auch beide Dateien lokal verwaltet werden, um einen solchen Vergleich durchführen zu können.

In dieser Projektarbeit werden Hash-Werte nicht aus Sicherheitsaspekten angewendet. Sie dienen hier als digitaler Fingerabdruck der erschlossenen Lehr- und Lernmaterialien. Er wird zum Vergleich von Dateien und zum Erkennen von Änderungen in den jeweiligen Dokumenten herangezogen. Mit dem Einsatz von Hash-Werten ist ein Vergleich zweier Dokumente viel effizienter durchführbar. Der Vorteil der Anwendung von Hash-Werten in der Aktualisierungskomponente besteht darin, dass man z. B. bei der Kontrolle der erfassten Lehr- und Lernmaterialien nicht die gesamten Dateien der einzelnen Dokumente zunächst abspeichern und diese dann mit großem Aufwand Schritt für Schritt miteinander vergleichen muss. Man muss lediglich die Hash-Werte der jeweiligen Dokumente generieren und diese dann vergleichen. Dies erfordert viel weniger Aufwand, da die Hash-Werte einfach und schnell zu berechnen sind. Weiterhin nimmt der Vergleich auf Grund der Kürze der Hash-Werte nur einen vergleichsweise geringen Aufwand in Anspruch. Insgesamt erreicht man also durch den Einsatz von Hash-Werten eine Effizienzsteigerung beim Kontrollieren der erfassten Materialien.

Die in dieser Projektarbeit entwickelte Komponente ist in der Lage die gesammelten Dokumente auf ihre Aktualität hin zu überprüfen. Beim Vergleich werden Hashwerte, die aus den jeweiligen Dokumenten generiert werden, verwendet, wodurch eine effiziente und zuverlässige Kontrolle möglich ist.

6 Ausblick

Bisher ist ein verändertes oder nicht mehr erreichbares Dokument nur an einem bestimmten Eintrag in der Datenbank erkennbar. Um der Redaktion von META-AKAD mehr Komfort beim Auffinden der nicht mehr aktuellen Lehr- und Lernmaterialien zu geben, könnte die entwickelte Komponente um eine graphische Benutzeroberfläche erweitert werden. Auf dieser Oberfläche werden die veränderten oder nicht mehr verfügbaren Dokumente aufgelistet. Aus dieser Liste wiederum kann der Redakteur die Lehr- und Lernmaterialien auswählen, die aus der, von META-AKAD, erfassten Menge von Materialien entfernt werden sollen, um die Aktualität der Daten zu erhalten. Der Redakteur muss somit keine Anfragen an die Datenbank mehr stellen um die jeweiligen Materialien zu finden und zu entfernen. Er benötigt auch kein Wissen über Anfragesprachen (z. B. SQL) um die gewünschten Daten zu erhalten.

Die Lösungsmöglichkeit aus Kapitel 2, die das Problem der Nichtverfügbarkeit eines erschlossenen Lehr- oder Lernmaterials auf robuste Weise lösen soll konnte aus zeitlichen Gründen in dieser Projektarbeit nicht mehr berücksichtigt werden. Ein Dokument, das aus technischen Gründen nur kurzzeitig nicht erreichbar ist, wird von der entwickelten Aktualisierungskomponente als nicht mehr verfügbar markiert und aus der Menge der Materialien, die für den Benutzer zugänglich sind, gestrichen. Eine robustere Lösung wäre für ein Lehr- oder Lernmaterial, das bei einem Kontrolldurchlauf nicht verfügbar ist, zunächst einen Zähler zu erhöhen, dieses zudem besonders kenntlich machen und zunächst nicht aus der Materialmenge zu streichen. Bei nachfolgenden Kontrolldurchläufen wird dieses Dokument explizit nochmals auf seine Verfügbarkeit kontrolliert. Ist es wiederholt nicht verfügbar, so wird der Zähler entsprechend erhöht, ansonsten wieder zurückgesetzt. Falls für das jeweilige Dokument jedoch der Zähler einen bestimmte Schwellwert erreicht, so wird das Dokument als nicht mehr verfügbar markiert. Diese beschriebene Lösungsmöglichkeit würde somit gewährleisten, das Dokumente, die nur für eine kurze Zeit aus technischen Gründen nicht zur Verfügung stehen, nicht sofort aus der Menge der Lehr- und Lernmaterialien, die dem Benutzer zur Verfügung stehen, gestrichen werden.

In Kapitel 4.2.1 wurden Gründe genannt, weshalb Message-Driven EJBs in diesem Projekt keine Anwendung fanden. In der Version EJB 2.0, die in dieser Projektarbeit verwendet wurde, ist es nicht möglich die Dienste eines Java-Beans zu einem ganz bestimmten Zeitpunkt zu aktivieren. Dass dies eine vorteilhafte Funktion ist, haben die Entwickler erkannt und in der Version EJB 2.1 ist der bisher fehlende Zeitaspekt durch den sogenannten *Timer Service* realisiert worden. Dieser Dienst wird durch den EJB-Container kontrolliert und erlaubt es Java-Beans so zu registrieren, dass diese zu einem bestimmten Zeitpunkt oder in festgelegten Intervallen aufgefordert werden ihre Dienste auszuführen. Das jeweilige Java-Bean, das den Timer Service verwendet, muss dazu die Schnittstelle javax.ejb.TimedObject implementieren. Der Timer Service kann für Stateless Session EJBs, Entity EJBs und Message-Driven EJBs verwendet werden. Es wäre jetzt also möglich in der Aktualisierungskomponente auf den Einsatz von Threads zu verzichten und das automatische Starten der Kontrolle mit Hilfe von Message-Driven EJBs in Verbindung mit dem Timer Service der Version EJB 2.1 zu realisieren [EJBS03].

7 Anhang

7.1 Bedienung des Maintainclient

Der Maintainclient verfügt über eine Benutzeroberfläche mit deren Hilfe man bestimmen kann zu welchem Zeitpunkt eine Überprüfung stattfinden soll. Dabei gibt es zwei Hauptoptionen:

- Zum einen kann man eine automatische Überprüfung konfigurieren, die dann in einem festgelegten Tagesrhythmus (volle Tage) zu einer ganz bestimmten Uhrzeit eine Überprüfung der Dokumente zyklisch vornimmt. Weiterhin kann man auch einen prozentualen Anteil der Dokumente angeben, der jeweils überprüft werden soll. Die Auswahl der Dokumente, die hierbei auf ihre Aktualität hin kontrolliert werden, ist bei jeder Prüfung zufällig.
- Zum anderen kann eine Überprüfung auch sofort manuell gestartet werden. Dabei besteht auch die Möglichkeit nur einen prozentualen Anteil der gesammelten Dokumente einer Aktualitätsprüfung zu unterziehen.
- Um zu erreichen, dass der Maintainclient bei einem Serverstart automatisch mitgestartet wird, muss zunächst ein erstes Deployment durchgeführt werden. Danach muss folgender Eintrag in der orion-application-deployments.xml-Datei erfolgen:

  ```
  <client-module path="maintainclient.jar" deployment-time="f5566560c0" auto-start="true" />
  ```

  Beim nächsten Start des Servers wird der Maintainclient automatisch mitaktiviert und die Überprüfung in den angegeben Intervallen automatisch ausgeführt.

7.2 Literatur

[AB99] Anton Bosselaers, The hash function RIPEMD-160. http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html

[EJBS01] Enterprise Java Beans Specification, Version: 2.0, Final Release, August 22, 2001

[EJBS03] Enterprise Java Beans Specification, Version: 2.1, Proposed Final Draft 2, June 3, 2003

[DT02] D-TRUST GmbH, Der digitale Fingerabdruck per Hashverfahren. http://www.d-trust.de/internet/content/signieren_hash.html

[FH02] Marcus Flehmig, Theo Härder. DB-orientierte Aufbereitung und Suchunterstützung für Lehr-/Lerndokumente. Fachbereich Informatik, Universität Kaiserslautern.

[IBM03] IBM, Integrating Message-Driven Beans into Enterprise Applications with WebSphere Studio -- Part 2: End-to-End Case Study http://www7b.software.ibm.com/wsdd/library/techarticles/0304_yu/yu2.html

[J2EE02] Designing Enterprise Applications with the J2EE™ Platform, Second Edition
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_02

[JL98] Jerry Luitwieler, Kryptographie, Oktober 1998
http://home.t-online.de/home/jerry.luitwieler/krypto5.htm

[PDA97] Philip A. DesAutels, SHA-1 Secure Hash Algorithm
http://www.w3.org/PICS/DSig/SHA1_1_0.html

Package
maintain.controller

maintain.controller

Class ChangeController

```
java.lang.Object
   |
   +-maintain.controller.ChangeController
```

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

Diese Klasse bestimmt und ueberprueft den Hashcode der Dokumente

Constructors

ChangeController

```
public ChangeController()
```

Methods

doControl

```
public java.util.Vector doControl(java.util.Vector urls,
                                  java.lang.String percent)
                           throws java.rmi.RemoteException
```

Diese Methode liest die Daten der jeweiligen url ein, berechnet den Hashcode und gibt einen Vector mit den veraenderten urls und Hashcodes zurueck

Parameters:

`urls` -
Die zu ueberpruefenden urls
`percent` -
Der prozentuale Anteil der urls, die ueberprueft werden sollen

Returns:

Vector mit der id und dem neuen Hashcode der Dokumente, die sich geaendert haben

maintain.controller

Class HTMLParser

```
java.lang.Object
   |
   +-maintain.controller.HTMLParser
```

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

Diese Klasse ist fuer das Parsen eines HTML-Dokuments verantwortlich

Constructors

HTMLParser

```
public HTMLParser()
```

Methods

parseHTML

```
public java.lang.String parseHTML(java.io.BufferedReader in)
                           throws java.io.IOException
```

Diese Methode liefert den reinen Text-content eines HTML-Dokuments

Parameters:

`in` -
buffered reader

Returns:

content des Dokuments als String

maintain.controller

Class InputController

```
java.lang.Object
   |
   +-maintain.controller.InputController
```

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

Diese Klasse validiert die Eingabedaten

Constructors

InputController

`public InputController()`

Methods

validatePercentInput

`public boolean validatePercentInput(java.lang.String input)`

Diese Methode validiert die Prozent-Angabe zum sofortigen Kontrollieren

Parameters:

`input` -
angegebener prozentualer Anteil

Returns:

boolean: richtig oder falsch angegeben

validateInput

`public boolean validateInput(java.lang.String[] input)`

Diese Methode validiert die Angaben fuer das zyklische Kontrollieren

Parameters:

`input` -
angegebene Daten

Returns:

boolean: richtig oder falsch angegeben

getInputResult

`public boolean[] getInputResult()`

Diese Methode liefert das Fehler-array

Returns:

(continued from last page)

array mit boolean-Werten

maintain.controller

Class RequestProcessor

```
java.lang.Object
   |
   +-HttpServlet
       |
       +-maintain.controller.RequestProcessor
```

public class **RequestProcessor**
extends HttpServlet

Diese Klasse ist fuer die Weitergabe eines Requests an die entsprechende Methode verantwortlich

Constructors

RequestProcessor

```
public RequestProcessor()
```

Methods

doGet

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
           throws ServletException,
                  java.io.IOException
```

Verarbeitet den HTTP GET request.

Parameters:

`request` -
object, das den request, den ein Client von einem Servlet gestellt hat, beinhaltet
`response` -
object, das den response, den ein Servlet zu einem Client sendet, beinhaltet

Exceptions:

`ServletException` -
wenn der request fuer get nicht verarbeitet werden kann
`java.io.IOException` -
wenn ein in- oder output - Fehler bei einem get-request aufgetreten ist

doPost

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
            throws ServletException,
                   java.io.IOException
```

Verarbeitet den HTTP POST request.

Parameters:

`request` -
object, das den request, den ein Client von einem Servlet gestellt hat, beinhaltet
`response` -
object, das den response, den ein Servlet zu einem Client sendet, beinhaltet

(continued from last page)

Exceptions:

`ServletException` -
wenn der request fuer post nicht verarbeitet werden kann

`java.io.IOException` -
wenn ein in- oder output - Fehler bei einem post-request aufgetreten ist

Package
maintainclient

maintainclient

Class IntervalThread

```
java.lang.Object
   |
   +-maintainclient.IntervalThread
```

All Implemented interfaces:
java.lang.Runnable

---

public class **IntervalThread**
extends java.lang.Object
implements java.lang.Runnable

Diese Klasse startet die Ueberpruefung in dem festgelegten Intervall

---

Fields

urls

`public java.util.Vector **urls**`

zu ueberpruefende urls

---

changedURL

`public java.util.Vector **changedURL**`

veraenderte Dokumente

---

interval

`public java.lang.String **interval**`

Daten, die vom Benutzer angegeben wurden

Constructors

IntervalThread

`public **IntervalThread**()`

Methods

main

`public static void **main**(java.lang.String[] args)`

Diese Methode startet den thread beim Serverstart

---

start

`public void **start**()`

Diese Methode startet thread

stop

```
public void stop()
```

Diese Methode stoppt den thread

run

```
public void run()
```

Diese Methode gibt an was der thread tun muss

Package
metabase.maintenance.beans

metabase.maintenance.beans

Class ErrorMessage

```
java.lang.Object
   |
   +-metabase.maintenance.beans.ErrorMessage
```

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

Diese Klasse ist fuer das Handling der Fehlermeldungen zustaendig

Constructors

ErrorMessage

`public ErrorMessage()`

Inititalisierung der Fehlermeldungen

Methods

getDayErrorMsg

`public java.lang.String getDayErrorMsg()`

Liefert die Tag-Fehlermeldungen

getTimeErrorMsg

`public java.lang.String getTimeErrorMsg()`

Liefert die Uhrzeit-Fehlermeldungen

getPercentErrorMsg

`public java.lang.String getPercentErrorMsg()`

Liefert die Prozent-Fehlermeldungen

getPercentErrorMsg1

`public java.lang.String getPercentErrorMsg1()`

Liefert die Prozent-Fehlermeldungen bei der sofortigen Kontrolle

setErrorMessages

`public void setErrorMessages(boolean[] error)`

Setzt das Fehlermeldungsattribut des ErrorMessage-Objekts

Parameters:

`error` -
Fehlermeldungen die angezeigt werden sollen

metabase.maintenance.beans

Class Interval

```
java.lang.Object
   |
   +-metabase.maintenance.beans.Interval
```

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

Diese Klasse ist fuer das Handling des Intervals zustaendig

Constructors

Interval

`public Interval()`

Methods

setInterval

`public void setInterval(java.lang.String[] interval)`

Diese Methode weist Daten Werte zu

getDay

`public java.lang.String getDay()`

Diese Methode liefert die Tage

Returns:

Tage

gethour

`public java.lang.String gethour()`

Diese Methode liefert die Stunden

Returns:

Stunden

getmin

`public java.lang.String getmin()`

Diese Methode liefert die Minuten

Returns:

Minuten

getpercent

`public java.lang.String` **`getpercent`**`()`

Diese Methode liefert den prozentualen Anteil

Returns:

prozentualer Anteil

Package
metabase.maintenance.ejb

metabase.maintenance.ejb

Class EJBUtil

```
java.lang.Object
   |
   +-metabase.maintenance.ejb.EJBUtil
```

public final class **EJBUtil**
extends java.lang.Object

Dies ist eine Hilfsklasse fuer das Bilden von ejb Referenzen

Constructors

EJBUtil

```
public EJBUtil()
```

Methods

getETControllerHome

```
public static ETControllerHome getETControllerHome()
                                                    throws javax.naming.NamingException
```

liefert Referenz fuer eTControllerHome

Returns:

eTControllerHome value

Exceptions:

`javax.naming.NamingException` -
wenn eine naming exception geworfen wird

getMetaModelObjectHome

```
public static metabase.metadata.ejb.metamodel.MetaModelObjectHome
getMetaModelObjectHome()

throws javax.naming.NamingException
```

liefert Referenz fuer metaModelObjectHome

Returns:

metaModelHome value

Exceptions:

`javax.naming.NamingException` -
wenn eine naming exception geworfen wird

(continued from last page)

getDependencyObjectHome

```
public static metabase.metadata.ejb.metamodel.DependencyObjectHome
getDependencyObjectHome()

throws javax.naming.NamingException
```

liefert Referenz fuer debendencyObjectHome

Returns:

dependencyObjectHome value

Exceptions:

`javax.naming.NamingException` -
wenn eine naming exception geworfen wird

---

getLREntryHome

```
public static metabase.xmlprocessor.ejb.learningresource.lrentry.LREntryHome
getLREntryHome()

throws javax.naming.NamingException
```

liefert Referenz fuer lrEntryHome

Returns:

lrEntryHome value

Exceptions:

`javax.naming.NamingException` -
wenn eine naming exception geworfen wird

---

getConfigurationStoreHome

```
public static metabase.metadata.ejb.configuration.ConfigurationStoreHome
getConfigurationStoreHome()

throws javax.naming.NamingException
```

liefert Referenz fuer configurationStoreHome

Returns:

configurationStoreHome value

Exceptions:

`javax.naming.NamingException` -
wenn eine naming exception geworfen wird

---

getBlacklistHome

```
public static metabase.metadata.ejb.blacklist.BlacklistHome getBlacklistHome()
                                                                    throws
javax.naming.NamingException
```

liefert Referenz fuer BlacklistHome

Returns:

BlacklistHome value

Exceptions:

(continued from last page)

`javax.naming.NamingException` -
wenn eine naming exception geworfen wird

metabase.maintenance.ejb

Interface ETController

public interface **ETController**

Remote interface des ETControllerEJB

Methods

changeInterval

```
public void changeInterval(java.lang.String[] input)
                    throws java.rmi.RemoteException
```

Diese Methode aendert die Zeitintervalangaben in der Datenbank

Parameters:

`input` -
Intervaldaten

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

getInterval

```
public java.lang.String[] getInterval()
                                throws java.rmi.RemoteException
```

Diese Methode liefert die Zeitintervalangaben aus der Datenbank

Returns:

Intervaldaten

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

getSleepTime

```
public long[] getSleepTime()
                    throws java.rmi.RemoteException
```

Diese Methode berechnet die Zeiten, die der thread schlafen muss

Returns:

Zeiten

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

getUrls

```
public java.util.Vector getUrls()
                         throws java.rmi.RemoteException
```

(continued from last page)

Diese Methode liefert die zu ueberpruefenden urls, die zugehoerigen ids und die checksum aus der Datenbank aus der Datenbank

Returns:

urls

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

getUris

```
public java.util.Vector getUris(java.util.Date date)
                         throws java.rmi.RemoteException
```

Diese Methode liefert urls fuer die Indexierung, die zur Volltextsuche notwendig ist

Returns:

urls

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

getBlacklistUris

```
public java.util.Vector getBlacklistUris(java.util.Date date)
                                  throws java.rmi.RemoteException
```

Diese Methode liefert urls geloeschter Inhalte. Fuer Volltextsuche notwendig

Returns:

urls

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

setFlags

```
public void setFlags(java.util.Vector changedURL)
              throws java.rmi.RemoteException
```

Diese Methode setzt resourcehaschanged und published in der Tabelle lrmaintain jeweils auf false und traegt die neue checksum ein

Parameters:

`changedURL` -
urls der geaenderten Dokumente

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

metabase.maintenance.ejb

Class ETControllerEJB

```
java.lang.Object
   |
   +-metabase.maintenance.ejb.ETControllerEJB
```

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

Diese Klasse repraesentiert den Enterprise Tier Controller

Constructors

ETControllerEJB

`public ETControllerEJB()`

Methods

ejbCreate

`public void ejbCreate()`

Ein container ruft diese Methode auf bevor ein session object erzeugt wird

ejbActivate

`public void ejbActivate()`

Die activate-Methode wird aufgerufen, wenn die Instanz aus dem passiven Zustand aktiviert wird

ejbPassivate

`public void ejbPassivate()`

Die passivate-Methode wird aufgerufen, bevor die Instanz in den passiven Zustand wechselt

ejbRemove

`public void ejbRemove()`

Ein container ruft diese Methode auf bevor ein session object zerstoert wird

setSessionContext

`public void setSessionContext(SessionContext ctx)`

Setzt das sessionContext Attribut

Parameters:

`ctx` -
der neue sessionContext value

(continued from last page)

changeInterval

`public void changeInterval(java.lang.String[] input)`

Diese Methode ändert die Zeitintervalangaben in der Datenbank

Parameters:

`input` -
Intervaldaten

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

getInterval

`public java.lang.String[] getInterval()`

Diese Methode liefert die Zeitintervalangaben aus der Datenbank

Returns:

Intervaldaten

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

getSleepTime

`public long[] getSleepTime()`

Diese Methode berechnet die Zeiten, die der thread schlafen muss

Returns:

Zeiten

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

getUrls

`public java.util.Vector getUrls()`

Diese Methode liefert die zu ueberpruefenden urls, die zugehoerigen ids und die checksum aus der Datenbank

Returns:

urls

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

getUris

`public java.util.Vector getUris(java.util.Date date)`

Diese Methode liefert urls fuer die Indexierung, die zur Volltextsuche notwendig sind

Returns:

(continued from last page)

urls

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

getBlacklistUris

```
public java.util.Vector getBlacklistUris(java.util.Date date)
```

Diese Methode liefert urls geloeschter Inhalte. Fuer Volltextsuche notwendig

Returns:

urls

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

---

setFlags

```
public void setFlags(java.util.Vector changedURL)
```

Diese Methode setzt resourcehaschanged und published in der Tabelle lrmaintain jeweils auf false und traegt die neue checksum ein

Parameters:

`changedURL` -
urls der geaenderten Dokumente

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

metabase.maintenance.ejb

Interface ETControllerHome

public interface **ETControllerHome**

Das Home Interface des ETController ejb

Methods

create

```
public ETController create()
                    throws java.rmi.RemoteException,
                           CreateException
```

Ein container ruft diese Methode auf bevor ein session object erzeugt wird

Returns:

ETControllerEJB

Exceptions:

`RemoteException` -
wird geworfen, wenn die Methode aufgrund eines Systemfehlers fehlschlaegt

`CreateException` -
wird geworfen, wenn create fehlgeschlagen ist