IPLUS
Konzeption eines leistungsfähigen Indexierungsprogrammes für das CSLIB 2000-System

Björn Voigt – Gruppe DELTA

6. Juli 1996

Inhaltsverzeichnis

1 Systemabgrenzung

Eine der wichtigsten Komponenten des künftigen Prototypen eines Bibliothekssystems CSLIB–2000 wird die Suchmaschine sein. Die Suchmaschine braucht schon zur Bearbeitung elementarer Anfragen eine leistungsfähige Indexierung.

Die Suchmaschine kann in den meisten Fällen nicht auf den formalen Daten der CSLIB 2000-Datenbank (im folgenden mit FDB abgekürzt) arbeiten, da diese zu umfangreich und zu weit strukturiert sind. Außerdem sind die Relationen meist nicht entsprechend den Anforderungen der Suchmaschine sortiert, so daß ohne Indexierung meist eine lineare Suche in den umfangreichen Datenbeständen notwendig würde.

Deshalb muß man für die Suchmaschine eine Art Cache (im folgenden Deskriptorspeicher DS genannt) anlegen. Der Aufbau des DS wird in Abschnitt 5 beschrieben.

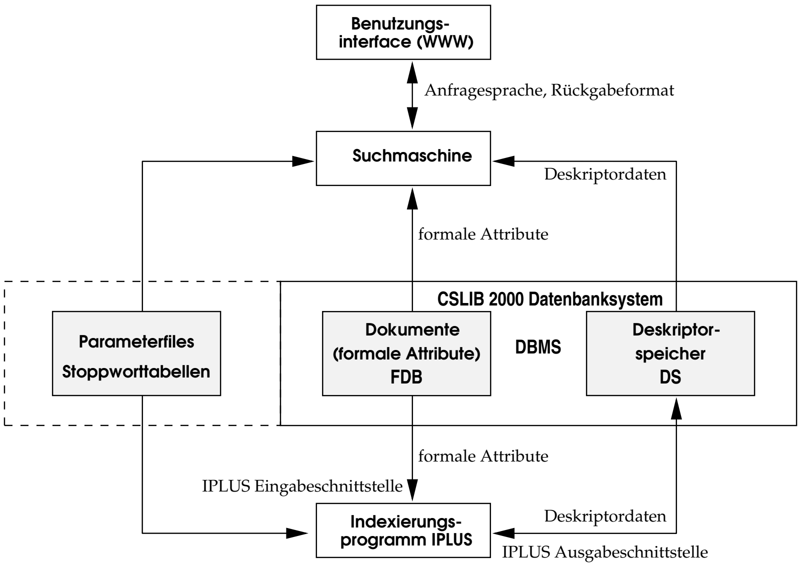

Abbildung 1: Schnittstellen von IPLUS

In Abbildung 1 werden die Datenflüsse zwischen den Komponenten Benutzungsinterface, Suchmaschine, Datenbanksystem und Indexierungsprogramm dargestellt. Schon die Tatsache, daß Suchmaschine und Indexierungsprogramm mit den gleichen Daten arbeiten, zeigt den engen Zusammenhang zwischen beiden Komponenten. Das Indexierungsprogramm hat jedoch keine Verbindung zum Benutzungsinterface.

Bei der Entwicklung des Indexierungsprogrammes sollte man sich stets an den Forderungen der Suchmaschine orientieren. Um die Entwicklung der Suchmaschine nicht unnötig zu belasten, verzichte ich auf einige Features wie die Indexierung nach der Additionsmethode. Für die phonetische Suche wird zunächst nur der SOUNDEX–Algorithmus vorgeschlagen.

2 Aufgaben des Indexierungsprogramms

Der Deskriptorspeicher DS ist immer direkt von den formalen Daten der CS-LIB 2000-Datenbank FDB abhängig. Jedesmal, wenn Daten der FDB hinzugefügt, geändert oder gelöscht werden, muß der DS entsprechend aktualisiert werden.

Aktionen in der FDB	**Folgeaktion im DS**
Hinzufügen eines Datensatzes D	Indexierung des Datensatzes, Erweitern des DS
Löschen eines Datensatzes D	Löschen aller Daten im DS, die sich nur auf D beziehen
Ändern eines Datensatzes D	Nacheinanderausführen der Aktionen für das Löschen und das Hinzufügen eines Datensatzes D

Tabelle 1: Aktualisierung des Deskriptorspeichers

Die Initialisierung und Aktualisierung des DS ist primäre Aufgabe des Indexierungsprogrammes.

Da das verwendete Datenbank Management System (DBMS) keine leistungsfähige Indexierung vorsieht, wird es sich kaum vermeiden lassen, ein solches Programm selbst zu entwickeln.

Diese Ausarbeitung ist also keine Dokumentation zu einer fertigen Software. Sie soll nur Anregungen und Tips für Projektteilnehmer geben, die ein Indexierungsprogramm für das CSLIB–2000 System entwickeln wollen. Das noch zu schreibende Programm bezeichne ich mit dem Codenamen **IPLUS**. **IP** steht dabei für „Indexierungsprogramm" und ist gleichzeitig der Name für ein „Testprogramm zur automatischen Indexierung von Dokumenten" [7], daß von der Gruppe DELTA [2] geschrieben wurde. Das „PLUS" von „IPLUS" soll andeuten, daß das künftige Indexierungsprogramm viel besser sein soll als IP. Die Spezifikationen von IP und IPLUS sind jedoch ähnlich. Auch IPLUS soll beliebige Relationen der FDB automatisch nach der Extraktionsmethode [1] indexieren.

3 Beziehungen zwischen DBMS und IPLUS

Aus der Abbildung 1 ist zu entnehmen, daß das Indexierungsprogramm eine Eingabe- und eine Ausgabeschnittstelle zum DBMS besitzt. Schnittstellen sind meist recht problematisch. Der Aufwand zur Schnittstellenimplementierung verringert sich erheblich, wenn das Indexierungsprogramm nah am DBMS angesiedelt wird. Im folgenden werden zwei Varianten diskutiert.

3.1 Indexierungsprogramm als Teil des DBMS

Das Indexierungsprogramm wird hier als Trigger realisiert. Immer, wenn sich der Inhalt der Datenbank ändert, wird anschließend das Indexierungsprogramm gestartet. Das DBMS würde z.B. beim Einfügen eines neuen Dokuments über eine SQL-INSERT-Anweisung IPLUS starten, welches die neuen Daten indexiert und dem DS hinzufügt.

Das Indexierungsprogramm muß bei dieser Lösung in der jeweiligen Datenbanksprache implementiert werden. Es muß geprüft werden, ob die Datenbanksprache über genügend Ausdrucksmittel zur Implementierung eines Indexierungsprogrammes verfügt und ob es Experten für diese Sprache in der Projektgruppe gibt.

3.2 Indexierung außerhalb des DBMS

Wenn man das Indexierungsprogramm außerhalb des DBMS implementiert, hat man den Vorteil, daß man bei der Auswahl einer Programmiersprache und eines Datenformats freier ist. Zusätzlich muß jedoch eine Ein- und Ausgabeschnittstelle implementiert werden. Bei dieser Lösung ist es deutlich schwieriger, die Konsistenz zwischen FDB und DS zu wahren. Das Auftreten kurzzeitiger (weniger als ein Tag) Inkonsistenzen ist zwar aus Sicht der Informatik unbefriedigend, andererseits hat sich der Bibliotheksnutzer genauso wie der Internet–Surfer längst an solche Schwächen gewöhnt.

Mit etwas Aufwand lassen sich diese kurzzeitgen Inkonsistenzen, die durch die Abkopplung des Indexierungsprogramm vom DBMS entstehen können, minimieren (siehe Abschnitt 7). Im folgenden wird nur noch diese Lösung erläutert.

4 Eingabeschnittstelle

Die Relationen der FDB werden einzeln indexiert. Falls man komplexere Relationen braucht, müssen diese über SQL-VIEWS erzeugt werden.

Neben dem Inhalt einer Relation braucht das Indexierungsprogramm eine Formatbeschreibung, in der steht, wie die Relation aufgebaut ist. Je nach Anspruch des Programmierers kann diese Formatbeschreibung „fest verdrahtet" oder aber über Parameterfiles definiert werden.

In einer Formatbeschreibung muß folgendes definiert sein:

- Name der Tabelle und Namen der Spalten (evtl. kombiniert)
- Angabe, welche Spalten indexiert werden sollen
- Angabe, welche Spalte(n) den Primärschlüssel bilden

Angenommen Tabelle 2 liegt als ISO-8859-1 kodiertes File vor. Die Titel sind teilweise in deutscher und teilweise in englischer Sprache. Deshalb soll zur Indexierung die Standard-Stoppworttabelle für „westliche" Sprachen verwendet werden. Als Spaltentrenner wird das Zeichen „|" verwendet.

<u>Dok-Id</u>	Autor	Titel	Erscheinungsjahr	Zeitstempel
1	Wirth, Niklaus	Programming in Modula–2	1985	1.2.96
2	Gehrke, Wilhelm	FORTRAN 77 Handbuch	1987	12.3.96

Tabelle 2: Beispielrelation Dokument

```
1 | Wirth, Niklaus  | Programming in Modula-2 | 1985 |  1.2.96 |
2 | Gehrke, Wilhelm | FORTRAN 77 Handbuch     | 1987 | 12.3.96 |
```

Die Formatbeschreibung könnte somit wie folgt aussehen.

```
Relationsname=Dokument;
Kodierung=ISO-8859-1;
Sprache=westlich;
Spaltenanzahl=5;
Spalte[1]=(s,-)Dok-Id;
Spalte[2]=(-,+)Autor;
Spalte[3]=(-,+)Titel;
Spalte[4]=(-,-)Erscheinungsjahr;
Spalte[5]=(d,-)Zeitstempel;
```

Für jede Spalte werden Optionen (x,y) spezifiziert.

Option	Diese Spalte
x=s	... ist Teil des Primärschlüssels.
x=d	... enthält die Information, wann die entsprechende Zeile zuletzt geändert wurde (Zeitstempel).
x=−	... hat keine Sonderbedeutung.
y=+	... soll indexiert werden.
y=−	... soll nicht indexiert werden.

Tabelle 3: Bedeutung der (x,y)-Optionen für jede Spalte

Die Liste der Optionen für das Indexierungsprogramm kann noch erweitert werden. Zusätzliche Optionen werden z.B. für die Indexierung von e-mail-Adressen, URLs und Signaturen benötigt.

5 Ausgabeschnittstelle

Der Deskriptorspeicher besteht minimal aus den beiden Relationen DOK_DESKRIPTOR und DOK_DESKPOS.

In der Relation DOK_DESKRIPTOR ist jeder Deskriptor genau einmal verzeichnet. Jeder Deskriptor wird dort in Textform zusammen mit einer einheitlichen DESK_ID, die den Primärschlüssel dieser Relation bildet, abgelegt. In der Relation können noch weitere Attribute, die einen Deskriptor beschreiben, aufgenommen werden (z.B. SOUNDEX-Code).

DESK_ID	TEXT	SOUNDEX–Code und weitere Attribute
1	Wirth	...
2	Niklaus	...

Tabelle 4: Relation DOK_DESKRIPTOR

Die Relation DOK_DESKRIPTOR wird nicht bei jedem Indexierungslauf neu erzeugt, sondern nur erweitert. Deskriptoren, die nicht mehr referenziert werden, können vom DBMS gelöscht werden (Garbage Collection).

Die Relation DOK_DESKPOS enthält die Attribute DESK_ID, QUELLE_ID und QUELLE_POS. Die Quellenidentifikation QUELLE_ID ist eine eindeutige Bezeichnung einer Spalte einer Relation der FDB. QUELLE_POS ist der Primärschlüssel der Relation, auf die sich QUELLE_ID bezieht. Der Primärschlüssel der Relation DOK_DESKPOS ist zusammengesetzt aus den drei Attributen DESK_ID, QUELLE_ID und QUELLE_POS.

Mit den zwei Dimensionen QUELLE_ID (Spalte und Tabelle) und QUELLE_POS (Zeile) läßt sich jedes Datenfeld exakt lokalisieren.

Die Deskriptoren *1 (=Wirth)* und *2 (=Niklaus)* kommen in der Relation *Dokument*, Spalte *Autor* vor und haben dort den Primärschlüssel *1*.

DESK ID	QUELLE ID	QUELLE POS
1	Dokument:Autor	1
2	Dokument:Autor	1

Tabelle 5: Relation DOK_DESKPOS

Problematisch scheint jedoch die Typisierung des Attributes QUELLE_POS. Dieses Attribut kann nämlich abhängig vom Wert QUELLE_ID ein atomarer Wert (INTEGER, STRING etc.) oder aber ein zusammengesetzter Wert sein. Deshalb schlage ich vor, das Attribut QUELLE_ID als ausreichend großen STRING zu bemessen. Atomare Werte müssen so dargestellt werden, daß sie sich leicht konvertieren lassen (z.B. mit string-to-int(str)) und zusammengesetzte Werte müssen so dargestellt werden, daß sie sich leicht separieren lassen. Zum Beispiel muß sich der zusammengesetzte Wert „IQ150:23" in die atomaren Werte STRING „IQ150" und INTEGER „23" konvertieren lassen.

Die Ausgabeschnittstelle sollte zunächst wie die Eingabeschnittstelle über ASCII-Files realisiert werden. Später kann man diese Schnittstelle eleganter gestalten (vergleiche dazu Abschnitt 7).

6 Der Indexierungsalgorithmus

6.1 Einlesen des Parameterfiles und der Deskriptortabelle

Zunächst wird das Parameterfile parsiert und interpretiert. Die Indexierungsoptionen für die einzelnen Spalten lassen sich z.B. in einer Liste ablegen. Danach wird die Deskriptortabelle DOK_DESKRIPTOR eingelesen. Diese enthält für jeden Deskriptor eine Deskriptor-Id. Da diese Tabelle sehr groß sein kann, empfehle ich eine Speicherung in Hashtabellen.

Im Betriebssystem Solaris 2.x gibt es dazu zwei Standardbibliotheken:

1. Die Library NDBM [11]. Diese Library dient zum Speichern von Datenbeständen auf Festplatte.

2. Die C++ Library Tools.h++ [8]. Darin gibt es eine Klasse RWTValHashDictionary. Mit dieser Klasse kann man Daten komfortabel über Hashtabellen im Speicher halten.

6.2 Die zeilenweise Indexierung

Die zu indexierende Relation wird zeilenweise eingelesen. Die einzelnen Felder werden in einer Liste gespeichert. Wird die erste Zeile der Beispieltabelle 2 eingelesen, ergibt sich folgende Liste:

```
Spalte[1]="1"
Spalte[2]="Wirth, Niklaus"
Spalte[3]="Programming in Modula-2"
Spalte[4]="1985"
Spalte[5]="1.2.96"
```

Mit Hilfe der Daten aus dem Parameterfile lassen sich diese Informationen klassifizieren. Der Primärschlüssel ist in diesem Fall atomar und trägt den Wert „1". Aus dem Namen der Relation und dem Spaltennamen läßt sich eine für die gesamte FDB eindeutige QUELLE_ID konstruieren.

Daten der Spalte 2 besitzen die QUELLE_ID *Dokument:Autor*, Daten der Spalte 3 die QUELLE_ID *Dokument:Titel* usw.

Im Parameterfile wird auch festgelegt, welche Spalten indexiert werden sollen. Im einfachsten Fall übergibt man den Inhalt eines Feldes (z.B. Inhalt des Feldes 2 der Zeile 1: „*Wirth, Niklaus*" an eine Funktion *indexiere_feld*, die diesen String indexiert.

Indexieren bedeutet hier das Aufspalten einer Zeile in eine Liste von Worten, wobei doppelte Wörter und Stoppwörter eliminiert werden. Dieser Vorgang wird in Abschnitt 6.5 näher erläutert.

Aus der Wortliste kann man nun unter Einbeziehung der QUELLE_ID, des Schlüssels und der Deskriptortabelle die Ausgabe zusammenstellen.

6.3 Zusammenstellung der Ausgabe

Für jedes Wort der Wortliste werden folgende Schritte durchlaufen:

1. Nachschlagen des Wortes in der gehashten Deskriptorliste.
2. Falls das Wort existiert, bekommt man durch Schritt 1 dessen DESK_ID geliefert. Ansonsten wird das Wort neu in die Hashtabelle eingefügt. Dabei muß eine neue DESK_ID erzeugt werden.
3. Es wird eine Zeile der Form

   ```
   DESK_ID | QUELLE_ID | QUELLE_POS |
   ```

   erzeugt.

Beispiel Die Wortliste „*Niklaus;Wirth*" sei Ausgabe der Funktion *indexiere_feld*. Das Wort „*Niklaus*" sei noch nicht in der Deskriptortabelle. Deshalb wird es eingefügt. Als DESK_ID wird der Wert *111* ausgegeben (maximaler Schlüsselwert plus 1 oder ähnlich). Damit wird die Zeile

```
111 | Dokument:Autor | 1 |
```

ausgegeben.

Das Wort „*Wirth*" ist dagegen schon in der Deskriptortabelle enthalten, z.B. weil schon vorher ein Buch von Wirth indexiert wurde. Die zugehörige DESK_ID sei 53. Damit wird die Zeile

```
53 | Dokument:Autor | 1 |
```

ausgegeben.

6.4 Stoppwortlisten

Stoppwortlisten benutzt man, um unwichtige Wörter bei der Indexierung aus der Menge der potentiellen Deskriptoren herauszufiltern. Damit wird der Deskriptorspeicher entlastet. Man will den DS von unnötigen Ballast wie Artikeln (*der, die, das, …*), Füllwörtern (*Äh, Oh, …*) und sprachlichen Abkürzungen (*d.h., usw., bzw.*) befreien. Trotzdem erhöht das Herausfiltern von Stoppwörtern den Informationsverlust bei der Indexierung. Deshalb sollte man beim Schreiben von Stoppwortlisten vorsichtig sein und zunächst nur kleine Stoppwortlisten verwenden.

Da die Daten der FDB meist mehrsprachig sind, braucht man auch Stoppwortlisten für alle vorkommenden Sprachen. Die Sprache läßt sich im Parameterfile einstellen. Alternativ kann man auch konkatenierte Stoppworttabellen einstellen. In diesen mehrsprachigen Stoppwortlisten dürfen natürlich keine Worte aufgenommen werden, die in einer anderen Sprache eine klare Bedeutung haben.

Unbedingt zu beachten ist, daß bei der Indexierung und bei der Suche die gleichen Stoppwortlisten verwendet werden müssen. Sonst könnte die Suchmaschine nicht nachvollziehbare Ergebnisse produzieren.

6.5 Indexierung eines Feldes

6.5.1 Allgemeine Anforderungen

Die Aufgabe, ein Feld in eine Wortliste zu zerlegen und danach doppelte Worte und Stoppwörter herauszufiltern, ist leider nicht ganz trivial. Die Worttrennung und die Worterkennung macht z.B. in folgenden Fällen Probleme:

- *Haus-tür*: Der Spiegelstrich ist durch einen Zeilenumbruch entstanden. Der gleiche Spiegelstrich kann jedoch auch Worte trennen.
- *Ausstellung Берлин-Moskau*, *Москва-Berlin*; *ϵ-Umgebung*: Besonders schwierig sind Worttrennungen, wenn Zeichensatzumschaltungen vorgenommen werden.
- *http://www.cs.tu-berlin.de*: Ist dies ein Wort oder sind es mehrere oder ist dies überhaupt kein gültiges Wort?
- *53QM4731*: Ist diese Signatur ein Wort?
- *S.U.S.E.*: Dieser Firmenname sollte als ein Wort verstanden werden.
- *Müller*: Das „ü" wird bei der sogenannten „C-locale" in UNIX [13] nicht als Buchstabe erkannt.

Die Eliminierung von Stoppwörtern kann schwierig werden, wenn die Ansprüche entsprechend hoch sind. Solche Ansprüche könnten sein:

- Beschreibung der Stoppworttabelle mit *regular expressions*, z.B. [D,d] [er,ie,as] für die Artikel „Der, der, Die, die, Das, das".
- Erhaltung von Abkürzungen. (Beispiel: AM soll erhalten bleiben, obwohl „am" und „Am" Stoppworte sind)
- Die Erhaltung von Stoppworten, wenn diese Teil eines Titels sind. Als Beispiel dafür erwähnte ein Projektteilnehmer den Film „Out of Rosenheim". Er konnte über verschiedene WWW–Suchmaschinen keine Informationen zu den Film finden, weil diese die Stoppworte „out" und „of" weggefiltert hatten. Ergebnis der Suche waren demzufolge seitenweise Homepages von Mitarbeitern und Studenten der Fachhochschule Rosenheim.
- Die Eliminierung von Formatierungsanweisungen wie HTML-Tags.

Bei der Implementierung der Funktion *indexiere_feld* sollte man versuchen, wenigstens einige der angesprochenen Probleme zu lösen. Meist ist es besser, eine heuristische Lösung zu wählen, als das Problem überhaupt nicht zu bearbeiten.

6.5.2 Nutzung der Tools spell und ispell

Da die Funktion *indexiere_feld* nur eine von vielen Funktionen von IPLUS ist, der möglichst nicht alle Energie der Projektgruppe, die sich mit der Indexierung befaßt, allein verschlingen soll, empfehle ich die Nutzung vorhandener Tools.

Die Aufgabe der Feldindexierung habe ich in meinem Prototypen IP an ein UNIX-Shelltool übergeben. Für diese Aufgabe bieten sich Tools zur Rechtschreibkorrektur an. In Solaris 2.x gibt es das Tool *spell* [4]. Dieses Programm habe ich im Testprogramm IP wegen seiner Einfachheit benutzt. Die Funktion *indexiere_feld* arbeitet dort folgende Schritte ab:

1. Der Inhalt des zu indexierenden Feldes wird in eine temporäre Datei *temp_input* geschrieben.

2. Das Tool *spell* wird mit dem Systemaufruf

   ```
   system("spell +SL temp_input >
     temp_output");
   ```

   aufgefufen. Dabei separiert *spell* die einzelnen Wörter in der Datei *temp_input* und bildet eine sortierte Wortliste. Von dieser Wortliste werden nun alle Stoppwörter der Stoppwortliste *SL* entfernt.

3. *IP* liest die Wortliste *temp_output* aus und stellt die Ausgabe zusammen.

Als Alternative zu *spell* bietet sich das Tool *ispell* [5] an. Es ist Teil der GNU-Distribution und findet zum Beispiel im *EMACS* oder auch als separates Shellprogramm Verwendung. Es bietet einige Vorteile gegenüber von *spell*:

- Der Quelltext ist verfügbar.
- Es können viel mehr Optionen angegeben werden. Insbesondere sind verschiedene Wörterbücher einstellbar. Diese Wörterbücher, die hier als Stoppworttabellen umfunktioniert werden, sind einstellbar.
- Es eignet sich besser für große Textmengen. Insbesondere bleibt die Reihenfolge der Wörter der ausgegebenen Wortliste gegenüber dem eingegebenen Feld erhalten.

Obwohl *ispell* zunächst als interaktives Shellprogramm konzipiert wurde, läßt es sich durch die Option „-a" auch als Filter benutzen. *Ispell* testet im Filtermode jedes Wort des Eingabestroms sequentiell. Abhängig davon, ob ein Wort im Wörterbuch gefunden wurde oder nicht, gibt *ispell* eine Meldung auf dem Ausgabestrom aus.

Beispiel: Die zweite Spalte der Ausgabe von *ispell* in Abbildung 2 kann man zum Zweck der Indexierung ignorieren. Entscheidend ist, ob ein Wort im Wörterbuch gefunden wurde („*", „+" oder „−") oder nicht (alle übrigen Zeichen).

Als Wörterbuch muß man mit der „-d" Option eine selbstdefinierte Stoppwortliste einstellen. Hinweise zum Schreiben eigener Stoppwortlisten / Wörterbücher finden Sie im Manual von *ispell* [6].

```
brie (~/cslib):cat << . | ispell -a -m -P -d deutsch
? In dieser Zeile ist ein falschgschriebenes Wort.
? Es gibt manchmal auch Hinweize.
? .
@(#) International Ispell Version 3.1.08 05/24/94
*
+ DIES
*
*
*
# falschgschriebenes 24
*

*
*
-
*
& Hinweize 1 22: Hinweise

brie (~/cslib):
```

Abbildung 2: Testausgabe von ispell

6.5.3 Nutzung von ispell als Hintergrundprozeß

Aus Effizienzgründen verbietet es sich, das Tool *ispell* für jedes Datenfeld neu zu starten. Es empfiehlt sich deshalb, ispell als Hintergrundprozeß zu starten. Das Indexierungsprogramm IPLUS muß über eine Eingabe- und eine Ausgabe-Pipe mit ispell kommunizieren.

Dazu muß IPLUS zunächst zwei Pipes [10] erzeugen und danach ispell starten. IPLUS schreibt nun den zu indexierenden Datensatz D in die Pipe 1 und wartet auf die Reaktion R(D) von ispell. Ispell schreibt R(D) in die Pipe 2. Diese Vorgehensweise ist jedoch gefährlich, weil sie die Gefahr einer Verklemmung bietet.

Eine Verklemmung kann bei großen – in Bezug auf die Puffergröße der Pipes – Datenfeldern (wie Abstracts) entstehen. Die Verklemmung entsteht durch

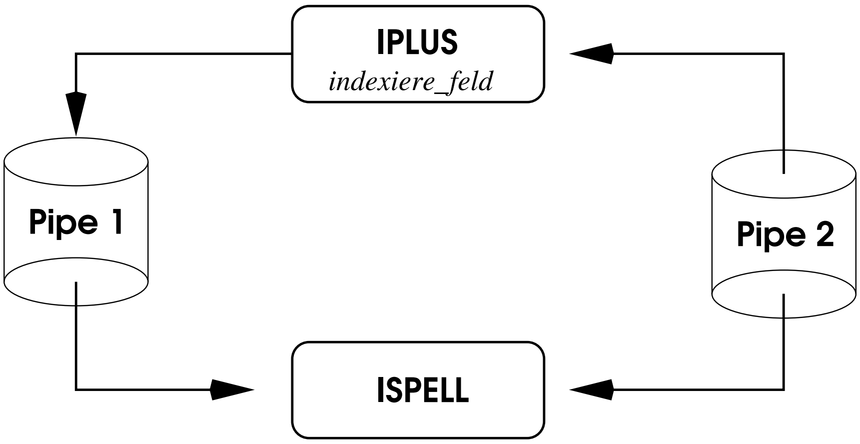

Abbildung 3: Kommunikation zwischen ISPELL und IPLUS

folgenden Programmablauf:

1. IPLUS beginnt, D in die Pipe 1 zu schreiben.
2. ISPELL verarbeitet den Anfang von D und beginnt die Reaktion R(D) in die Pipe 2 zu schreiben.
3. Der Puffer der Pipe 2 wird irgendwann voll, da er von IPLUS nicht geleert werden kann. IPLUS schreibt stattdessen weiter in Pipe 1.
4. ISPELL wird angehalten, sobald die Ausgabepipe 2 voll ist.
5. Die Daten der Pipe 1 können von ISPELL nicht mehr abgeholt werden.
6. IPLUS schreibt solange Daten von D in die Pipe 1 bis diese voll ist.
7. Sobald Pipe 1 voll ist, wird auch der Prozeß IPLUS angehalten.

Die Verklemmung entsteht also durch den Zyklus, daß ISPELL auf IPLUS wartet und umgekehrt. Es gibt zwei Ansätze, solche Verklemmungen auszuschließen.

1. Es wird darauf geachtet, daß die Pipe 1 nicht überfüllt wird. Überlange Datensätze müssen von IPLUS in mehrere kleine Datensätze aufgeteilt werden.

2. Das Schreiben in die Pipe 1 und das Auslesen der Pipe 2 erfolgt parallel. Um diese Parallelität zu erzeugen, muß man zwei Instanzen von IPLUS bilden. Traditionell wird dazu der fork [12] Systemaufruf verwendet. Die Kommunikation zwischen den beiden unabhängigen Instanzen von IPLUS kann aber aufwendig werden. Als modernes Konzept bieten sich daher Threads [9] an. Der abgespaltene Thread liest die Pipe 2 aus, während die ursprüngliche Instanz von IPLUS in die Pipe 1 schreibt. Die Zugriffe auf globale Variablen müssen dabei synchronisiert werden.

Ispell trennt den Eingabestrom nach bestimmten Regeln in Worte auf. Leider wird in der Manualpage von *ispell* [5] nicht dokumentiert, nach welchen Regeln dies geschieht. Einleuchtend wäre, wenn nur nach Whitespaces (Space, Newline, Tabulator, ...) und nach Satztrennzeichen (Punkt, Komma, Semikolon, ...) getrennt würde. Allerdings trennt *ispell* ohne Zusatzoption z.B. auch nach Anführungszeichen und dem Kaufmanns–UND („&”). Möglicherweise hängt die Klassifikation der Zeichen in druckbare Zeichen, Whitespaces, Ziffern und Sonderzeichen auch noch von der eingestellten *locale* [13] ab, wie dies in der *ctype*-Library [14] der Fall ist.

Solange man den Trennmechanismus von *ispell* nicht genau kennt, kann man die Klassifikationen im Ausgabestrom von *ispell* nicht direkt bestimmten Worten im Eingabestrom zuordnen. Da man sich ohnehin nur für die Worte interessiert, die nicht in der Stoppwortliste stehen, ist das kein großer Nachteil. Diese im Sprachgebrauch eines Rechtschreibprogramms wie *ispell* quasi falschgeschriebenen Worte kann man über den ausgegebenen Offset (siehe Abbildung 2 und Manualpage [5]) im Eingabestrom lokalisieren.

Um das Ende eines Datenfeldes zu erkennen, empfehle ich, den Eingabestrom mit einem „*Ende_des_Feldes*”-Wort zu terminieren.

Zu testen wäre allerdings noch, in wieweit der *ispell*-Trennmechanismus den Anforderungen aus Abschnitt 6.5.1 erfüllt. Falls der Mechanismus unbefriedigend arbeitet, wird man einen eigenen Mechanismus implementieren müssen, der das Eingabefeld so vorverarbeitet, daß *ispell* richtig trennt.

7 Auswertung der Zeitstempel–Information

Bisher wurde noch nicht darauf eingegangen, welchen Sinn das Zeitstempel–Attribut in der Beispielrelation in Tabelle 2 hat.

Der Zeitstempel gibt an, wann der Datensatz das letzte Mal geändert wurde. Wenn IPLUS zusätzlich die Information hat, wann diese Relation das letzte Mal indexiert wurde, kann es sich darauf beschränken, nur die geänderten Datensätze zu indexieren.

Falls man die Verwaltung der Zeitstempel–Information dem DBMS überläßt, kann man dazu übergehen, nur noch die geänderten Datensätze an IPLUS zu übertragen. Durch die geringeren Datenmengen wird es möglich, daß der DS sehr schnell gegenüber der FDB aktualisiert wird. Wenn z.B. neue Dokumente in die FDB eingefügt werden, könnte das DBMS danach sofort IPLUS starten, um die neuen Daten zu indexieren. Die Zeitspanne der Inkonsistenz der Daten von FDB und DS ließe sind damit in den Minutenbereich absenken.

Vollständige Konsistenz bekommt man jedoch erst, wenn man die Transaktionsmechanismen des DBMS benutzt bzw. das Indexierungsprogramm vollständig in das DBMS integriert (siehe Abschnitt 3.1).

8 Erzeugung des SOUNDEX–Codes

Laut informeller Spezifikation soll der Suchmaschine eine effiziente Suche mit Ähnlichkeitsmaßen ermöglicht werden. Wie in meinem Vortrag „Suche mit Ähnlichkeitsmaßen in Datenbanken" [3] ausführlicher dargestellt, müssen die Ähnlichkeitsmaße schon beim Design des Deskriptorspeichers berücksichtigt werden.

Wenn man einen einfachen SOUNDEX–Algorithmus [3] verwenden will, braucht man in der Relation DOK_DESKRIPTOR (siehe Tabelle 4) nur zusätzlich die Werte für das Attribut SOUNDEX–Code berechnen.

In Originalform [3] berücksichtigt der SOUNDEX–Algorithmus nur das englische Alphabet. Die Klassifikation der Zeichen in Vokale und die verschiedenen Konsonantenklassen muß man für andere Sprachen selbst entwickeln. Für die erste Implementierung schlage ich vor, die Zeichen des ISO–8859–1 Zeichensatzes zu klassifizieren. In Tabelle 6 werden dafür Beispiele gegeben.

Zeichenklasse	Zeichen
Vokale	a e u é û ...
Konsonanten 1	b f p v ...
Konsonanten 2	c g j k q s x z ß ...
...	...

Tabelle 6: ISO–8859–1 SOUNDEX–Zeichenklassen

Da die Aussprache einzelner Buchstaben in verschiedenen Sprachen variiert, müßte man korrekterweise Deskriptoren zunächst in eine phonetische Transkription transformieren. Der SOUNDEX–Algorithmus ließe sich dann sprachunabhängig auf dem phonetischen Alphabet definieren.

Über die theoretischen Grundlagen des Information Retrieval und speziell der Indexierung informieren die Ausarbeitungen der Gruppen „DELTA" und "miauh". Die übrigen Verweise sind technische Dokumentationen zu Shelltools und Manualpages zu Funktionen und Bibliotheken auf Solaris 2.x Workstations.

Der Projektgruppe, die sich der Indexierung annimmt, wünsche ich viel Erfolg.

Literatur

[1] „Indexierung von Dokumenten". Ausarbeitung im WWW unter „http://www.cs.tu-berlin.de/~ul/csliblv/www-neu/gruppe4/indexa.html". Gruppe DELTA. 1995.

[2] „Gruppe DELTA". Homepage im WWW unter „http://www.cs.tu-berlin.de/~ul/csliblv/www-neu/gruppe4/gruppe4.html". Gruppe DELTA. 1995/96.

[3] „Suche mit Ähnlichkeitsmaßen in Datenbanken". Ausarbeitung im WWW unter „http://www.cs.tu-berlin.de/~ul/csliblv/www-neu /gruppe4/suche.html". Gruppe DELTA. 1996.

[4] „spell ... – report spelling errors". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/man/man1/spell.1".

[5] „ispell ... – Interactive spelling checking". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/gnu/man/man1 /ispell.1".

[6] „ispell – format of ispell dictionaries and affix files". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/gnu/man /man4/ispell.4".

[7] „IP - Ein Testprogramm zur Indexierung von Dokumenten". Ausarbeitung im WWW unter „http://www.cs.tu-berlin.de/~ul/csliblv/www-neu/gruppe4/ip.html". Gruppe DELTA. 1995.

[8] „Tools.h++ Introduction and Reference Manual" in „C++ 4.0.1 Answerbook". SunSoft. 1994.

[9] „thr_create – create a new thread of control". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/sol2/man /man3t/thr_create.3t".

[10] „pipe – create an interprocess channel". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/sol2/man/man2/pipe.2".

[11] „ndbm ... – data base subroutines". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/sol2/man/man3/ndbm.3"

[12] „fork ... – create a new process". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/sol2/man/man2/fork.2"

[13] „locale – subset of a user's environment that depends on language and cultural conventions". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/man/man5/locale.5"

[14] „ctype ... – character handling". Manualpage im WWW unter „http://www.cs.tu-berlin.de/cgi/rtfm/usr/man/man3c/ctype.3c"

Index