Bulletin of the Technical Committee on

Data Engineering

March 1996 Vol. 19 No. 1

IEEE COMPUTER SOCIETY
50 YEARS OF SERVICE • 1946-1996

Letters

Special Issue on Integrating Text Retrieval and Databases

Letter from the Editor-in-Chief

Bulletin Related Items

Apologies are in order for the long delay in publishing this issue. There are a number of things that have happened recently that have affected the *Data Engineering Bulletin* and are at least partially responsible for the delay. Let me discuss these in turn.

Web Distribution In addition to distributing the Bulletin via ftp, the Bulletin can now be accessed electronically on the web. I want to thank Toby Lehman of the IBM Almaden Research Center for initiating this move to the web and for authoring the original web page (which has since evolved). The URL for the Bulletin's web site is "http://www.research.microsoft.com/research/debull". The home page contains instructions on how to down load issues of the Bulletin. Most TC members should find the web page easier to use than ftp.

IEEE Copyright Policy The IEEE had not, in the past, required copyright transfer for Bulletin articles. Now authors will be expected to sign a copyright transfer form. Part of the copyright agreement is that IEEE permission is now required before articles can be made accessible through third party web channels, i.e. pages not maintained by the author, his organization, or the IEEE. A possible inference is that this is an effort to protect IEEE paper journals. A better strategy, I believe, is for the IEEE determine how to bring technical information to its members at the lowest possible cost by exploiting the web. That may require eventually abandoning paper journals.

Financial Status As in 1995, the IEEE Computer Society has generously provided the resources needed to publish the Bulletin in hardcopy. One might well ask, given web accessibility, why hardcopy is needed. There are two short term answers.

1. Many readers outside of the United States do not have access to the internet, and hence to the web. Over time, this number should drop rapidly, but this is still a problem.
2. Many authors like to see their work in hardcopy. I believe that as electronic publication becomes more common, that this issue will fade as well.

So the Bulletin continues for now with hardcopy distribution. And we are grateful to the Computer Society for supporting this.

About this Issue

The current issue is on a topic of growing importance in the database field. It is the integration of new forms of data into traditional relation database systems. The issue focuses in particular on text and its retrieval. If the database field is to continue to prosper, and indeed, if databases are to have an important long term role, then they need to support vastly more of the world's data. And most of this data is not formatted, much less formatted as relations. Thanks to Eliot Moss for assembling this issue.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

This special issues of Data Engineering reflects the growing interchange between the database an information retrieval (IR) communities, stimulated by the increasing desire to handle text and other media in overall database / information systems. The particular collection of papers was obtained by inviting submissions from participants of a recent workshop on integrating text and databases, held in conjunction with the SIGIR '95 conference. While I cannot claim that they represent the entire state of the art, I hope you find them interesting and stimulating.

As you will see, there are interesting issues at every level of system conception and implementation: data model, query model, query processing, internal data organization, transaction model, overall system architecture, etc. The first four papers concern themselves with different ways of modeling and obtaining combined database and IR functionality, using different system architectures, models, etc. The last paper explores transaction models for combined database / IR systems. In any case, I hope you find them interesting, useful, and stimulating of new ideas for research or commercial systems. In my view this is an important emerging area that will receive much more attention in the future.

Eliot Moss
University of Massachusetts at Amherst

Models for Integrated Information Retrieval and Database Systems

Norbert Fuhr*
University of Dortmund, Germany

Abstract

In this paper, we show that there is a mismatch between information retrieval (IR) and database (DB) concepts, and we devise solutions for this problem. DB oriented approaches have to distinguish between the logical and the content structure of objects, and should also consider the layout structure. Data independence—not regarded in IR before—can be achieved by using the notion of vague predicates. Since IR is based on uncertain inference, data models with uncertainty are required for an integrated IR-DB system. For this purpose, we present a probabilistic relational algebra. As extensions, probabilistic Datalog yields a more expressive query language, whereas a probabilistic nested relational model is more appropriate for modelling document structures.

1 Introduction

In the classical view of the information retrieval (IR) and databases (DB) fields, databases contain formatted data (or facts), while IR systems deal with unformatted data, i.e., texts. For formatted data, powerful data models and query languages have been developed, whereas for texts, robust text analysis (e.g., stemming) methods, term weighting, and retrieval models based on uncertain inference have been the focus of research. In addition, due to the rather different application environments, the DB field has developed methods for coping with database integrity, security, concurrency, and recovery. In IR, these topics have become an issue only recently. From this point of view, there is little overlap between the two fields, and so there is no common basis from which an integration could be started. In fact, current commercial solutions for IR-DB-integration offer very poor IR functions, since they are based on Boolean retrieval. Thus, they ignore the intrinsic vagueness and uncertainty of IR.

In this paper, we will show that due to the small overlap between the two fields, there is a mismatch of concepts, since fundamental concepts from one field simply do not exist (yet) in the other. Specifically, the DB field should consider that documents have logical, layout, and content structure. The concept of data independence is new to IR. Viewing IR as uncertain inference requires data models with uncertainty. These issues and possible solutions are described in detail in the following sections.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*E-Mail: fuhr@ls6.informatik.uni-dortmund.de

2 Conceptual modelling of objects

In [MRT91], conceptual modeling of multimedia documents is discussed. In electronic publishing, one distinguishes between the logical and the layout structure of documents. For example, for document type "letter", there is a logical structure containing sender, recipient, salutation, several paragraphs, closing, and signature. The layout structure may consist of several rectangular regions placed on the top, the bottom, centered or flush left or right on a page. However, the logical and the layout structure are not sufficient for performing IR on document bases. For this reason, content structure is added, which permits the representation of the meaning of documents. For example, a letter may belong to the conceptual document type "offer", consisting of frame (with sender and recipient) and body, where the latter is a set of product infos with name, description, and price.

In [Fuhr92b], this approach is extended from documents to all kinds of database objects. It is claimed that for databases comprising a large number of different object types, there is a need to support all three types of structures.

Typically, the DB field deals only with the **logical structure** of objects. There are powerful data models for representing a broad range of possible object structures, for retrieving and manipulating these structures. Recently, there has also been increasing interest in data models supporting the complex logical structure of text documents (e.g., [Loef94]). However, the more complex the logical structures, the greater the need for considering the content structure. For example, assume that we have an object-oriented DBMS as part of a CASE tool. In a bank application, the software manager might search for all modules affected by a new tax law. Since the modules were developed with different tools and in different programming languages, their logical structure will vary greatly; even two modules performing the same task may have very different logical structure. As another example, consider an environmental information system where a user seeks information about the pollution of ground water with nitrate. There may be text documents, maps, and relations containing relevant information. A traditional DBMS forces the user to ask specific queries for each type of object.

Generally speaking, if there is a large number of object types, then there is a need for supporting queries that are independent of specific logical structures. This leads to the introduction of a content structure of objects. Below we describe some characteristics of the content view.

The major task of the **content structure** is to offer a unified view over a large number of object types with varying logical structures. In general, this goal cannot be achieved without sacrificing precision. A major reason for different logical structures is the variety of objects to be modelled. If we want to map different logical structures onto a single content structure, then we will lose precision. This can affect details of the (logical) structure as well as the values stored within this structure. A simple example for this process is text indexing: Whereas a document may have a complex logical structure, most IR methods represent its content by a simple set of terms with associated weights. As another example, in materials databases, the properties of a certain material also depend on the product form (e.g., bar, tube, or sheet metal) and on form-specific parameters (e.g., thickness of sheet metal). In seeking materials with certain properties, it should be possible to query without specifying additional parameters first (because these parameters are mostly material-specific).

In most applications, the content structure is derived from the logical structure—similar to a view in a database. Generally, updates to the content view are not possible, since it is not clear how such an update should affect the logical structure. For example, if we want to add a certain term to a document's content view, how should its logical structure change to reflect the new content structure?

In this context, the most important problem is the definition of the mapping from the logical structure to the content structure. For the case of text documents, this has been (and is still) a major issue of IR research. Here a very simple data model has been used for the content structure, and the mapping involves mainly statistical methods. For other types of objects, more complex data models will become necessary, in order to cover at least certain aspects of the logical structure.

Looking at the **layout structure**, we raise the issue of whether or not there should be a layout structure for objects in general. With classical data models, there was no need, since generic output formats were sufficient.

However, for complex objects, CAD applications, or in geographic information systems, the presentation of objects is an important issue. Currently, this problem has to be solved in the application programs. But we think that the layout structure can and should be integrated in the database, providing object-type-specific presentation mechanisms. As a simple example, in object-oriented systems (e.g., Smalltalk), there exisits a "print" method for most object classes, but with different implementations in different classes (by means of method overloading). For an implementation of the layout view, this method should become more flexible, e.g., allowing for displaying only certain parts of an object, or different presentations depending on the display context.

Recent studies in the field of IR and human-computer interaction have shown that small changes in the layout have a significant impact on the performance of humans when using such a system (e.g., [All94]). When databases with complex object structures are used in an interactive mode, layout structure will also be an important issue.

For retrieving objects, queries may specify conditions on all three structures of objects. Assume that we have an office information system; a query about information concerning computer equipment would relate to the content structure, a search for the last letter from Miller & Co. refers to the logical structure, and looking for a letter with two tables on the first page concerns the layout structure.

3 Data independence in IR

Physical and logical data independence are fundamental concepts in the DB field, leading to three levels of data organisation (physical, logical, and external). Unfortunately, data independence is a concept totally unknown in IR. Here the IR task is subdivided into (document) indexing and retrieval. In fact, the former task does exactly what the term "indexing" also means in the DB field: creation of an index (for concepts extracted from documents). This index supports efficient processing of typical IR queries. However, IR systems only allow for queries that can be processed this way. Thus, there is no physical data independence in these systems! In [Fuhr96], we describe some problems where this approach falls short even in typical text retrieval systems: search for full word forms vs. stemming, noun phrase search, and treatment of compound words.

Clearly, implementing logical data independence in IR systems would mean that a query may contain conditions irrespective of the presence of an index for processing them, as long as there are means for evaluating the conditions on the database, e.g., by scanning all documents. The query formulation should also be independent of the presence of an index—in contrast to some commercial IR systems that provide scanning capabilities, but not in a form that yields physical data independence.

In [Fuhr96], we describe an approach to implementing data independence in IR systems. As an underlying concept, we present data types with vague predicates. In the usual IR systems, the concept of data type does not show up explicitly, mainly because there is only one data type: text. However, in real IR applications, there is an obvious need to cope with different data types to allow queries referring not only to the semantic content of documents, but also to other attributes such as author, journal title, or publication year. Most commercial systems are based on Boolean logic, and thus they can use data types with comparison operators as in DB systems. But IR systems consider the intrinsic vagueness of query formulations—a user may be uncertain about the spelling of an author's name, or when he requests documents from this year a paper from last December may also be relevant. For this problem, the concept of vague predicates was proposed in [Fuhr90]. Here a query condition consists of an attribute name, a predicate, and a comparison value (e.g., AUTHOR $\approx$ 'Meier'). For a specific attribute value (e.g., 'Maier') the vague predicate yields an estimate of the probability that the condition is fulfilled from the user's point of view—instead of a Boolean value as in DB systems. In [Fuhr92a], we showed how vague predicates can be integrated into a (probabilistic) IR system.

More formally, a data type D is a pair $(|D|, P_D)$, where $|D|$ is the domain and $P_D = \{p_1, \ldots, p_n\}$ is a set of (vague) predicates, where each predicate is a function $p_i : |D| \times |D| \rightarrow [0, 1]$. Some of the predicates may be Boolean, i.e., restricted to the set $\{0, 1\}$.

Here are a few examples of data types in IR systems. For text, usually there is a single predicate "contains", taking a single word or phrase as the comparison value, with the comparison based on stemming. Names require a strict predicate for equality, and it would be useful to have two vague predicates, for phonetic similarity (e.g., based on Soundex codes) and string similarity (e.g., based on trigrams). For dates or numeric values, there could be strict and vague versions of the usual comparison operators $=, <, >$, etc.

For multimedia IR, vague predicates are even more important, since term-based approaches can hardly be used in such an environment; instead, similarity of values (i.e., vague equality) plays a major role. [Fal96] describes a rather general approach for dealing with similarity of multimedia data types such as time series or images, along with an access structure that allows for efficient processing.

Given this concept of data types with vague predicates, the specification of the conceptual level of IR systems is straightforward. Here documents have attributes of certain data types, for which vague predicates are provided. The implementation of the vague predicates—whether by a simple inverted list, complex access structures, or by scanning—is a matter of internals of the IR system.

Based on this implementation of the conceptual level of the IR system, one can provide different views on a collection of documents (using standard DB techniques), e.g., for data security or for monolingual views of a multilingual database.

4 Data models for IR-DB systems

Having formulated the basic concepts common to IR and DB systems, we can undertake the major task: developing data models (and query languages) for integrated IR-DB systems.

In the logical view of databases, computing the answer to a query q means finding all objects o that imply the query, i.e., for which the logical formula $q \leftarrow o$ is true. If one takes the same approach for document retrieval, then a document d should be retrieved in response to a query if $q \leftarrow d$ can be shown to be true. In fact, this is exactly what Boolean retrieval does. However, since IR must deal with vagueness and imprecision, this approach is not adequate. Thus, it is argued in [Rijs86] that IR should be regarded as an uncertain inference process. Using probability theory as a basis, Rijsbergen claims that document retrieval is equivalent to computing the probability $P(q \leftarrow d)$ for a document d.

Comparing the two types of inference, one can see that uncertain inference as used in IR is just a generalization of the inference used in DBMS. So an integration of IR and DB on the logical level appears feasible. However, this view just sets the frame for a set of possible solutions. In order to arrive at a model that can be implemented and also satisfies the needs of typical applications, one has to take a data model from the DB field and generalize it so that it also comprises probabilistic inference.

4.1 A probabilistic relational algebra

In [FR96b], we present a probabilistic relational algebra (PRA) generalizing standard relational algebra, suitable as the basis of an integrated IR-DB system. Below, we briefly describe PRA.

Unlike similar approaches, PRA is based on intensional semantics, which is the key to a probabilistic data model that is a real generalization of relational algebra. Probabilities are introduced in the relational model in the following way: Let $\bar{R}$ denote an instance of an ordinary relation, and δ a tuple from the corresponding domain. Then we have either $\delta \in \bar{R}$ or $\delta \notin \bar{R}$. In probabilistic relations, we assume that δ is associated with a binary stochastic event η, where η = true if $\delta \in \bar{R}$, and η = false otherwise. So we regard the probability $\beta = P(\delta \in \bar{R})$ as additional information belonging to a tuple in a probabilistic relation. We distinguish between basic events and complex events. The actual database relations (hereafter called base relations) contain only basic events, identified by so-called event keys. Complex events (denoted by event expressions) are Boolean combinations of basic events. They are formed as a by-product of relational operators. By keeping track of the basic events

leading to a tuple in a derived relation, we implement intensional semantics. Thus, for example, in PRA the equality $R \cap S = R - (R - S)$ holds, whereas extensional semantics approaches (that only manipulate the tuple weights) would yield different results for both sides of the equation.

DocTerm

η	β	DocNo	Term
DT(1, IR)	0.9	1	IR
DT(2, DB)	0.7	2	DB
DT(3, IR)	0.8	3	IR
DT(3, DB)	0.5	3	DB
DT(4, AI)	0.8	4	AI

Figure 1: Example probabilistic relation

So a tuple t of a probabilistic relation is a triple (η, β, δ) where $t.\delta$ is the data tuple containing the values for the different attributes, $t.\eta$ gives the event expression, and $t.\beta$ is the probability of the event being true. For base relations, $t.\beta$ is given explicitly, whereas for derived relations, it is computed from $t.\eta$ as the probability of the corresponding Boolean expression of the basic events being true. An example probabilistic relation representing probabilistic document indexing is given in Figure 1.

η	β	Term
DT(1, IR) $\vee$ DT(3, IR)	0.98	IR
DT(2, DB) $\vee$ DT(3, DB)	0.85	DB
DT(4, AI)	0.8	AI

Figure 2: $\Pi_{\text{Term}}(\text{DocTerm})$

η	β	DocNo
DT(1, IR)	0.9	1
DT(2, DB)	0.7	2
DT(3, IR) $\vee$ DT(3, DB)	0.9	3

Figure 3: $\Pi_{\text{DocNo}}(\sigma_{\text{Term='IR'}}(\text{DocTerm})) \cup \Pi_{\text{DocNo}}(\sigma_{\text{Term='DB'}}(\text{DocTerm}))$

Now the five basic operations of relational algebra are redefined in PRA such that in addition to the manipulation of the attribute values, we also form Boolean combinations of the event expressions of the tuple involved: For union and projection, the disjunction of the corresponding event expression is formed, cartesian product leads to the conjunction of the event expression, and for difference, we form the conjunction of the first argument and the negation of the second argument; the selection operation does not change the event expressions of the tuples selected. For example, Figure 2 gives the result of projecting DocTerm (DocNo, Term) onto the attribute Term, and Figure 3 shows the answer to a query asking for documents about IR or DB.

In order to compute the probabilities of result tuples, the inclusion-exclusion formula ([Bill79, p. 20]) is applied. In addition, we need assumptions about the stochastic independence or dependence of the basic events. In the examples shown above, we have assumed that the basic events are independent of each other. This assumption is reasonable in most situations—many IR models are based on the same assumption. Another broad

DY

η	β	DocNo	Year
DY(1,1980)	0.8	1	1980
DY(1,1981)	0.2	1	1981
$\top$	1.0	2	1990
DY(3,1985)	0.4	3	1985
DY(3,1986)	0.4	3	1986
DY(3,1987)	0.2	3	1987

Figure 4: A probabilistic relation for the imprecise attribute Year

GT

η	β	A	Year
GT(1991,1993)	0.1	1991	1993
GT(1992,1993)	0.3	1991	1993
GT(1993,1993)	0.6	1991	1993
$\top$	1.0	1994	1993
$\top$	1.0	1995	1993
$\top$	1.0	1996	1993

Figure 5: A probabilistic relation for the vague predicate $\tilde{>}$

range of applications can be handled by regarding certain events as being disjoint of each other. For example, imprecise attribute values can be handled this way. Therefore, we model these values as probability distributions over the corresponding domain. As an example, the probabilistic relation DY depicted in Figure 4 shows some documents for which the publication year is not known precisely: document 1 was published either in 1980 or 1981, document 2 certainly in 1990 ($\top$ denotes the certain event here) and document 3 in 1985, 1986, or 1987. Events belonging to the same document are disjoint from each other, but events from different documents can be regarded as being independent. For example, the PRA expression $\Pi_{\text{DocNo}}(\sigma_{\text{YEAR}>1985}(\text{DY}))$ searches for numbers of documents published after 1985; as result, we get the event expression (DY(3,1986) $\vee$ DY(3,1987)) for DocNo=3 , with a probability of $0.4 + 0.2 = 0.6$.

In the examples given so far, we have used the traditional view of text indexing, which is less suitable for a data model that addresses the conceptual level. Now we show how vague predicates can be handled in PRA. We only give an intuitive explanation here; a more formal treatment can be found in [FR96b].

As a simple example, assume that a person searching for relevant articles published after 1993 may also be interested in a highly relevant paper published one or two years before. In principle, any vague predicate $p_i{:}|D| \times |D| \rightarrow [0,1]$ can be modelled by means of a probabilistic relation with two attributes for the arguments and the tuple probability giving the result of the predicate. Figure 5 shows some tuples of the relation for the example from above. Given this relation, it is obvious that we could express a vague selection condition like "Year $\tilde{>}$ 1993" on relation R by the PRA expression R $\bowtie \Pi_{\text{Year}}(\sigma_{\text{A}=1993}(\text{GT}))$.

Mostly, the relation corresponding to a vague predicate will not be given explicitly; rather, it will be specified implicitly by means of an internal function of the DB system.

With the different features as described above, PRA extends relational database systems for coping with uncertainty and vagueness. It is the first model of this kind for which all equivalences from ordinary relational alge-

bra hold. Due to the latter fact, we can exploit the connection between relational algebra and relational calculus: We use the same transformation process from calculus to algebra but apply the algebraic expression to probabilistic relations. So we have a probabilistic relational calculus that yields probabilistic relations as answers.

PRA only forms a starting point for the development of data models for integrated IR-DB-systems. In the DB field, the shortcomings of the relational model are well known. From the IR point of view, the expressiveness of relational algebra is not sufficient for advanced applications, and even for simple document structures, the data modelling capabilities are inappropriate. Below, we present two approaches addressing these issues.

4.2 Probabilistic Datalog

New IR applications require inferential capabilities that are not available with relational algebra. In [Fuhr95a], we show that hierarchical document structures, hypertext documents, or retrieval employing terminological structures such as thesauri raise the need for recursive query languages.

Given PRA, Datalog is a suitable candidate for such a query language. In [Fuhr95b], a probabilistic version of Datalog is presented. We only want to mention the major ideas of this approach here.

Probabilistic Datalog is an extension of stratified Datalog (e.g., [Ull88]). On the syntactic level, the only difference is that with ground facts, a probabilistic weight may also be given, e.g.:

```
0.7 docterm(d1,ir).  0.8 docterm(d1,db).
```

These ground facts represent the probabilistic relations of PRA. Rules and queries are the same as in ordinary Datalog. So a query looking for documents both about IR and DB can be expressed as

```
?- docterm(X,ir) & docterm (X,db).
```

As an example involving recursion, consider retrieval in hypertext structures, where we have directed links between single documents (or nodes). Assume these links also have probabilistic weights, e.g.:

```
0.5 link(d2,d1).  0.4 link(d3,d2).
```

The idea behind these weights is this: If we have a link from D1 to D2, and D2 is about a certain topic, then there is a certain probability that D1 is about the same topic. This probability is given by the weight of the link predicate. Now we can formulate the rules

```
about(D,T) :- docterm(D,T).
about(D,T) :- link(D,D1) & about(D1,T).
```

Due to the recursive definition, a document also may be about a term if it is only indirectly linked to another document indexed with this term. Thus, the query

```
?- about(X,db).
```

would return three documents, namely d1 with probability 0.8, d2 with probability $0.5 \cdot 0.8 = 0.4$ and d3 with probability $0.4 \cdot 0.5 \cdot 0.8 = 0.16$.

The evaluation of probabilistic Datalog programs can be performed in the same way as with ordinary Datalog. The only differences are the construction of event expressions for result tuples during the evaluation process and the final computation of the tuple probabilities.

4.3 A probabilistic NF2 model

As a step towards increased data modelling capability, we developed a probabilistic nested relational model (probabilistic non-first-normal-form, or pNF2 for short) [FR96a]. Using the NF2 model (instead of flat relations) for IR has been proposed by several authors, e.g., [SP82].

As an example, consider the pNF2 relation shown in Figure 6. According to the type of probabilistic events and the (in)dependence of tuples in a pNF2 relations, we distinguish between deterministic, disjoint, and independent relations. Here BOOK is a deterministic relation with a deterministic subrelation AUTHOR. PRICE represents an imprecise attribute value, thus it is a disjoint (sub-)relation. INDEX is an independent relation which gives a set of probabilistic index terms for each book.

BOOK														
η	β	BNO	YEAR	PRICE			INDEX			AUTHOR		
				η	β	VAL	η	β	TERM	η	β	NAME
B⊤	1.0	1	92	B⊤P1 B⊤P2	0.6 0.4	30 25	B⊤I1 B⊤I2	0.9 0.8	IR DB	B⊤A⊤ B⊤A⊤	1.0 1.0	Smith Jones
B⊤	1.0	2	93	B⊤P3	1.0	29	B⊤I3 B⊤I4	0.9 0.8	AI DB	B⊤A⊤	1.0	Miller
B⊤	1.0	3	92	B⊤P4 B⊤P5	0.7 0.3	28 25	B⊤I5	0.9	DB	B⊤A⊤	1.0	Jones
B⊤	1.0	4	90	B⊤P6 B⊤P7	0.5 0.5	32 28	B⊤I6	0.9	DB	B⊤A⊤	1.0	Jones

Figure 6: Relation BOOK_{det}(BNO, YEAR, $\text{PRICE}_{\text{disj}}$ (VAL), $\text{INDEX}_{\text{ind}}$ (TERM), $\text{AUTHOR}_{\text{det}}$ (NAME))

IRDBBOOKS														
η	β	BNO	YEAR	PRICE			INDEX			AUTHOR		
				η	β	VAL	η	β	TERM	η	β	NAME
B⊤ ∧ B⊤I1 ∧ B⊤I2	0.72	1	92	B⊤P1 B⊤P2	0.6 0.4	30 25	B⊤I1 B⊤I2	1.0 1.0	IR DB	B⊤A⊤ B⊤A⊤	1.0 1.0	Smith Jones

Figure 7: Selection

The operations of pNF2 are similar to those of other NF2 models. Thus, we have the PRA operators plus nesting and unnesting. Further, in selection formulas, set comparison and element test may be used as predicates. As an example of selection, consider a search for books about IR and DB:

$$\begin{aligned} \text{IRDBBOOKS} &= \sigma[\text{'DB'} \in \text{INDEX} \wedge \text{'IR'} \in \text{INDEX}](\text{BOOK}) \\ &= \sigma[\{\text{'DB','IR'}\} \subseteq \text{INDEX}](\text{BOOK}) \end{aligned}$$

In the latter formulation, the constant set represents a deterministic relation. The result depicted in Figure 7 also illustrates the interpretation of probabilities in this model: Here tuple probabilities in inner relations are conditional probabilities with respect to the event expressions of the "surrounding" outer tuples. For this reason, the indexing weights of the terms IR and DB both have changed to 1.0, since the original events already affect the probability of the corresponding document tuple.

BL30			
η	β	BNO	PRICES
B⊤ ∧ B⊤P2	0.4	1	25
B⊤ ∧ B⊤P3	1.0	2	29
B⊤ ∧ B⊤P4	0.7	3	28
B⊤ ∧ B⊤P5	0.3	3	25
B⊤ ∧ B⊤P7	0.5	4	28

Figure 8: Unnest followed by selection

Without further operators, a disadvantage of NF2 algebras is that in general operations on inner relations cannot be performed without prior unnesting. This leads to rather complex expressions. For example a query

looking for books with a price less than \$30 must be formulated as follows:

$$\text{BL30} = \sigma[\text{PRICES} < 30](\mu[\text{PRICE:PRICES}](\Pi[\text{BNO,PRICE}](\text{BOOK})))$$

BL30A					
η	β	BNO	PRICE		
			η	β	VAL
B⊤ ∧ B⊤P2	0.4	1	B⊤P1 B⊤P2	0.0 1.0	30 25
B⊤ ∧ B⊤P3	1.0	2	B⊤P3	1.0	29
B⊤ ∧ (B⊤P4 ∨ B⊤P5)	1.0	3	B⊤P4 B⊤P5	0.7 0.3	28 25
B⊤ ∧ B⊤P7	0.5	4	B⊤P6 B⊤P7	0.0 1.0	32 28

Figure 9: Unnest with duplicated attributes, followed by selection

However, in the result some books occur multiple times (see Figure 8), and we do not see all possible prices of a book. These problems can be overcome only by duplicating attributes first (for which we use the colon symbol in the projection list), and a final projection on the relevant attributes (see Figure 9):

$$\begin{aligned}\text{BL30A} \quad = \quad & \Pi[\text{BNO,PRICE}](\sigma[\text{PRICES} < 30](\mu[\text{PRICE}\prime\text{:PRICES}] \\ & (\Pi[\text{BNO,PRICE,PRICE:PRICE}\prime](\text{BOOK}))))\end{aligned}$$

In order to overcome these problems, nested NF2 algebras have been proposed (e.g., [SS86]). For this purpose, the pNF2 algebra supports path expressions as an additional parameter for all operators. In addition, we also allow for path expressions in selection formulas, which we call selection paths. Thus, the last query from above now can be expressed as

$$\text{BL30A} = \sigma[/\text{PRICE(VAL}<30)](\text{BOOK}).$$

Here /PRICE denotes the selection path. This feature allows existential quantification over relation-valued attributes. Universal quantification is achieved by negating the selection path (and the comparison condition, of course). For example, the following asks for books dealing with IR only:

$$\text{IRONLY} = \sigma[/\text{INDEX(TERM='DB')} \wedge \text{INDEX} \neq \emptyset](\text{BOOK}) = \sigma[\text{INDEX}=\{\text{'IR'}\}](\text{BOOK}).$$

5 Conclusions and outlook

In this paper, we have devised major concepts for the integration of IR and DB systems. From the DB point of view, a clear distinction between the logical and the content structure of documents or objects in general should be made, and the layout structure should be considered, too. Clearly, IR mainly has focused on the content structure, so both fields have to broaden their view in this respect.

For IR systems, the implementation of physical data independence is essential in order to avoid a mismatch between DB and IR concepts.

Since IR uses uncertain inference, we need data models with uncertainty for an integrated IR-DB system. As a basic model, we have developed PRA. Probabilistic Datalog yields a more expressive query language, whereas the pNF2 model is more appropriate for modelling document structures.

Clearly, using object-oriented (OO) models for integrated IR-DB systems is the next step. The approach described in [HW92] focuses on the behavioral aspects of IR objects, but does not address the issue of uncertain inference. OO modelling of multimedia objects is used by many authors from the DB field, but the uncertainty issue is hardly ever mentioned.

Our further work will use the pNF2 model as a starting point for the development of an OO data model with uncertainty. As a major extension to other OO models, we also consider inheritance on attributes (see [Fuhr96]), which is essential for performing networked IR. A related topic is the development of an OO retrieval logic. In traditional IR approaches, the unit to be retrieved is always fixed—namely documents as a whole. With complex document or hypermedia structures, determining the relevant units to be returned in response to a query is a completely open problem so far.

References

[All94] Allen, B. (1994). Perceptual Speed, Learning and Information Retrieval Performance. In: Croft, W. B.; van Rijsbergen, C. J. (eds.): *Proc. 17th ACM-SIGIR Conf.*, 71–80. Springer-Verlag.

[Bill79] Billingsley, P. (1979). *Probability and Measure*. John Wiley & Sons.

[Fal96] Faloutsos, C. (1996). Fast Searching by Content in Multimedia Databases. *IEEE Data Eng. Bull. 18(4)*, 31–40.

[FR96a] Fuhr, N.; Rölleke, T. (1996a). *A Probabilistic NF2 Relational Algebra for Imprecision in Databases*. Submitted for publication.

[FR96b] Fuhr, N.; Rölleke, T. (1996b). A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. ACM Trans. on Info. Sys. (to appear).

[Fuhr90] Fuhr, N. (1990). A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In: McLeod, D., et al. (eds.): *Proc. 16th Int'l Conf. on Very Large Databases*, 696–707. Morgan Kaufman.

[Fuhr92a] Fuhr, N. (1992a). Integration of Probabilistic Fact and Text Retrieval. In: Belkin, N., et al. (eds.): *Proc. 15th ACM SIGIR Conf.*, 211–222. ACM.

[Fuhr92b] Fuhr, N. (1992b). Konzepte zur Gestaltung zukünftiger Information-Retrieval-Systeme. In: Kuhlen, R. (ed.): *Experimentelles und praktisches Information Retrieval*, 59–75. Universitätsverlag Konstanz.

[Fuhr95a] Fuhr, N. (1995a). Modelling Hypermedia Retrieval in Datalog. In: Kuhlen, R.; Rittberger, M. (eds.): *Hypertext - Information Retrieval - Multimedia, Synergieeffekte elektronischer Informationssysteme*, 163–174. Universitätsverlag Konstanz, Konstanz.

[Fuhr95b] Fuhr, N. (1995b). Probabilistic Datalog - a Logic for Powerful Retrieval Method. In: Fox, E., et al. (eds.): *Proc. 18th ACM SIGIR Conf.*, 282–290. ACM.

[Fuhr96] Fuhr, N. (1996). *Object-Oriented and Database Concepts for the Design of Networked Information Retrieval Systems*. Submitted for publication. URL: http://ls6-www.informatik.uni-dortmund.de/reports/96/Fuhr-96.html.

[HW92] Harper, D.; Walker, A. (1992). ECLAIR: an Extensible Class Library for Information Retrieval. *Computer Journal 35(3)*, 256–267.

[Loef94] Loeffen, A. (1994). Text Databases; A Survey of Text Models and Systems. *SIGMOD Record 23(1)*, 97–106.

[MRT91] Meghini, C.; Rabitti, F.; Thanos, C. (1991). Conceptual Modeling of Multimedia Documents. *IEEE Computer 24(10)*, 23–30.

[Rijs86] van Rijsbergen, C. J. (1986). A Non-Classical Logic for Information Retrieval. *Computer Journal 29(6)*, 481–485.

[SP82] Schek, H.-J.; Pistor, P. (1982). Data Structures for an Integrated Database Management and Information Retrieval System. In: *Proc. 8th Int'l Conf. on Very Large Data Bases*, 197–207. Morgan Kaufman.

[SS86] Schek, H.-J.; Scholl, M. (1986). The Relational Model with Relation-Valued Attributes. *Information Systems 2*, 137–147.

[Ull88] Ullman, J. (1988). *Principles of Database and Knowledge-Base Systems*, Volume I. Computer Science Press.

Exploiting the Functionality of Object-Oriented Database Management Systems for Information Retrieval

Gabriele Sonnenberger
UBILAB, Union Bank of Switzerland
Bahnhofstrasse 45, 8021 Zurich, Switzerland
sonnenberger@ubilab.ubs.ch

Abstract

In this paper, we present the approach of FIRE to utilizing an object-oriented Database Management System (DBMS) for Information Retrieval (IR) purposes. First, a comprehensive overview of previous attempts to use DBMSs for implementing IR systems is given. Next, differences between DBMSs and IR systems, with regard to indexing and retrieval, are discussed. In addition, some shortcomings of DBMSs with regard to supporting IR systems are pointed out. Then, an overview of FIRE, which is designed as a reusable IR framework, is given and its approach presented in more detail. Special attention is given to the design and implementation of an IR-index and how retrieval efficiency can be improved by using the optimization facilities of the underlying object-oriented DBMS.

1 Introduction

Due to recent advances in Information Technology and Telecommunications, more and more information is produced and distributed by electronic means. Instead of plain ASCII texts, information is increasingly encoded in different forms, e.g., as graphs, tables or formatted texts. These developments impose additional requirements on the management and retrieval of information. Whereas in the past the focus in Information Retrieval (IR) was on text, which was mostly considered to be unstructured, today's IR systems have to face information which usually consists of structured and unstructured parts and which may be composed of different media.

We observe further changes in the flow and distribution of information. In the past, the user of an IR system was typically a passive consumer of information gathered at the special sites of professional information providers; while nowadays a user is often both an information consumer *and* an information provider, e.g., in a corporation's document management system. Hence, data management issues like persistent storage of data, concurrency control, and recovery after failures are gaining importance, and IR systems must cope with these issues.

Developing techniques for the indexing and retrieval of heterogeneous information units is a demanding topic genuine to the IR domain. In contrast to this, data management issues have already been investigated extensively by the Database (DB) community, which has also developed theories and techniques applied in productive systems. Rather than 'reinventing the wheel', it is natural for developers of IR systems to try to profit from the results of the DB community by basing IR systems on Database Management Systems.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In this paper, we report on our experience utilizing a Database Management System (DBMS) for IR purposes. Section 2 reviews previous attempts to use DBMSs for implementing IR systems. In Section 3, we discuss differences between IR systems and DBMSs with regard to indexing and retrieval, and point out some shortcomings of DBMSs with regard to supporting IR systems. Finally in Section 4, we sketch our IR framework, which we call FIRE, and present our approach for using the functionality of an object-oriented DBMS in an IR system. The paper concludes with some remarks on future work.

2 IR Systems Based on DBMSs

2.1 Relational DBMSs

A first approach to use a DBMS for IR purposes is to consider IR as a DB application. This has been proposed, for instance, by Macleod & Crawford [Mac83], Blair [Bla88], and Smeaton [Sme90]. Common to these proposals is that the IR systems are based on a relational DBMS, and SQL is used as retrieval language. These IR systems do not store full documents, but bibliographic references to documents; these usually provide the title of the document represented, the names of the authors, an abstract, some content descriptors, and the details about the date and location of the publication.

This approach of making use of a DBMS has received major criticisms. One factor that has been criticized especially, e.g., by Schek & Pistor [Sch82], is that the relational DB model represents information in a rather unnatural way by a set of tables (relations). Further, SQL queries tend to be rather complex and difficult to understand; see for instance the examples given in [Mac91]. In addition, retrieval may be quite computationally expensive, especially when information from different tables has to be combined by a 'join' operation.

A severe shortcoming of this approach is the restriction to reference retrieval. This restriction is not so much voluntary as a matter of the underlying relational DB model. The relational model requires the fields of a record to be of a fixed length, thus making it difficult for a relational DBMS application to manage information units such as ordinary texts, which usually vary in length. (Of course, there are work-arounds like follow-up records, but such work-arounds make the modeling of information even more awkward.) A second severe shortcoming is that SQL in its basic form is restricted to exact matching. An exact match is appropriate for many DB applications, especially when information is structured to a high degree and the vocabulary used is rather fixed. In most IR applications however, information units like texts are significantly less structured and the vocabulary used is usually unrestricted. Correspondingly, users of IR systems find it far more difficult or even impossible to issue a query which successfully delivers all information relevant to a given information need, but excludes irrelevant material. Therefore, more advanced approaches to IR abandoned the exact matching paradigm and find instead the pieces of information which *best match* the user's query by applying weighting schemata. The user receives as result a *ranked list* which is sorted by the assumed probability that a piece of information is suited to answer the user's information need.

Several attempts have been made to provide better DB support for the development of IR systems by extending the relational model, or by adding new features to SQL. To provide more natural external views of information, Schek & Pistor [Sch82] have proposed a generalization of the relational model which allows nested relations. Lynch & Stonebraker [Lyn88] have introduced abstract data types, which make the formulation of content-based search conditions more convenient, although retrieval is still based on the exact matching paradigm. An extension of SQL allowing a 'similar-to' comparison operator has been proposed by Motro [Mot88]. Furthermore, various approaches have been developed for dealing with uncertain information, for instance by Garcia-Molina & Porter [Gar90].

2.2 Coupling an IR System with a DBMS

A different approach to making use of a DBMS for IR purposes is to couple an IR system with a DBMS. Croft et al. [Cro92] attempted a loose coupling between the IR system INQUERY and IRIS, a prototype version of an object-oriented DBMS. They chose an object-oriented DBMS since the object-oriented model allows one, in contrast to the relational model, to store complex textual information in a quite natural way. The IR and the DB system are coupled externally by a control module. There are links from the information units stored by INQUERY to the corresponding textual objects of the IRIS database. The integrated system provides the functionality of both underlying systems. Thus, the user may issue content-based queries as well as DB queries. However, a severe problem of this coupling approach is that information is stored twice, by the IR system as well as by the DBMS. This may cause consistency problems when information is modified. Furthermore, there is no full integration of content-based queries and DB queries.

Gu et al. [Gu93] have chosen an alternative way to couple an IR system with a DBMS. They have embedded the functionality of the IR system INQUERY into the relational DBMS Sybase. Furthermore, they have extended SQL by a function which takes an INQUERY query as input and allows one to choose the INQUERY database to be consulted for evaluating the query. This system avoids storing information twice: textual information is stored and managed by the IR system and other kinds of information by the DBMS, which also provides pointers to the textual information stored and managed by the IR system. The main drawback is that two different approaches for managing and retrieving information are used, which makes the management of information more difficult. Further, the best match retrieval paradigm is restricted to textual information, whereas an application of this retrieval paradigm to other kinds of information would be highly desirable, as for instance pointed out by Fuhr [Fuh92].

2.3 Object-oriented DBMSs

Bearing in mind the shortcomings of the relational model, it has been proposed to use other models, more appropriate than the relational DB model, as a basis for developing IR systems; e.g., an array model [Mac87] or an object-oriented DB model [Har92p]. The object-oriented DB model is especially appealing, since object-oriented technology is maturing and the first commercial systems are already available.

The object-oriented approach has also been adopted for developing our IR framework FIRE. The framework is implemented using ObjectStore [Lam91], a commercially available object-oriented DBMS. In ObjectStore, persistence is not part of the definition of an object, but a matter of allocation at the time of object creation ('persistence by allocation'). Thus, objects of the same type can be allocated persistently as well as transiently. Furthermore, it makes essentially no difference whether we deal with a transient or persistent object. These features of ObjectStore enable us to design and implement an IR system in a problem-adequate way, mostly neglecting data storage details. Thus the developer of an IR system can rely on the DBMS's means for persistent storage, concurrency control, recovery after failure, etc., without having to accept severe restrictions, as experienced from relational DBMSs.

Object-oriented DBMSs provide a useful basis for the development of IR systems, yet could support IR systems even further with regard to performance issues. ObjectStore, for instance, allows one to improve retrieval efficiency by building specialized indexes and by optimizing queries. However, indexing and retrieval in an IR system differ in certain aspects from indexing and retrieval in a DBMS, as described in the following section. Consequently these facilities cannot be used directly for IR purposes. Modifying the core functionality of a DBMS for exploiting these facilities is not in the range of a regular user of a DBMS, thus we cannot expect a solution from the DB side — at least not in the short term. Yet, the optimization facilities of object-oriented DBMSs can be utilized in IR systems by a proper design on the IR side, as is shown in Section 4.

3 Some Differences between IR Systems and DBMSs

3.1 Index

The term 'index' is used both by the DB community and by the IR community, however with different meanings. While the function of a DB-index is to improve performance, an IR-index typically also provides additional data. In detail, we can observe the following differences:

- A typical IR-index is an inverted file with index entries, each consisting of a key representing a feature derived from a source and a set of postings. A posting may include information about the frequency of the given feature within the source, or may specify its position within the document in more detail. A DB-index, however, only maintains the paths to the objects where an attribute has a certain value.
- In an IR application, information about the characteristics of the collection represented by an index is required when computing the relevance of an information unit to a user's query. An example of such information is the maximum frequency of a term in a collection of texts or the mean value and standard deviation of a set of numbers. Since a DB-index serves only to optimize access, no such collection information is needed.

Due to the differences, typical IR-indexes are not supported by DBMSs. Optimized access to information — as is done by a DB-index — is, however, crucial for implementing efficient IR systems. Thus, how we can use DB-indexes to build IR-indexes is an open research problem.

3.2 Retrieval

A typical DBMS provides a retrieval interface which allows one to check whether two values are equal, and to determine whether a value is smaller or greater than another value. Usually such queries can be optimized, e.g., by instructing the DBMS to create and maintain appropriate indexes.

Advanced IR systems are supposed to support approximate ('best') matching. Unfortunately, approximate matching is not supported by DBMSs. This is a severe shortcoming, since approximate matching is far more time-consuming than exact matching, and therefore demands optimization facilities. To deal with the additional complexity, we urgently need specially tuned matching methods and techniques to avoid a linear search through an index. These methods and techniques should ideally work hand in hand with available DBMS techniques.

3.3 Indexing and Retrieval Functionality

In the case of a conventional IR application, i.e., one which is not based on a DBMS, the application developer has to provide all the indexing and retrieval functionality needed. Developing an IR system on top of a DBMS might save design and implementation effort, since DBMSs already provide indexing and retrieval functionality. The task of the application developer would then be to choose appropriate functionalities and to instruct the DBMS correspondingly. In order to index a set of objects for instance, the application developer would have to state which object attributes are to be indexed in which way, and to select the index structures best suited for the attribute values. Indexes would then be set up automatically under control of the DBMS. Unfortunately, the options provided by DBMSs do not cover the full range needed for IR applications and do not allow one, for instance, to index a text directly in a traditional IR manner using the indexing mechanisms of a DBMS. Thus, we need to investigate how we can use indexing and retrieval functionalities of object-oriented DBMSs for IR purposes.

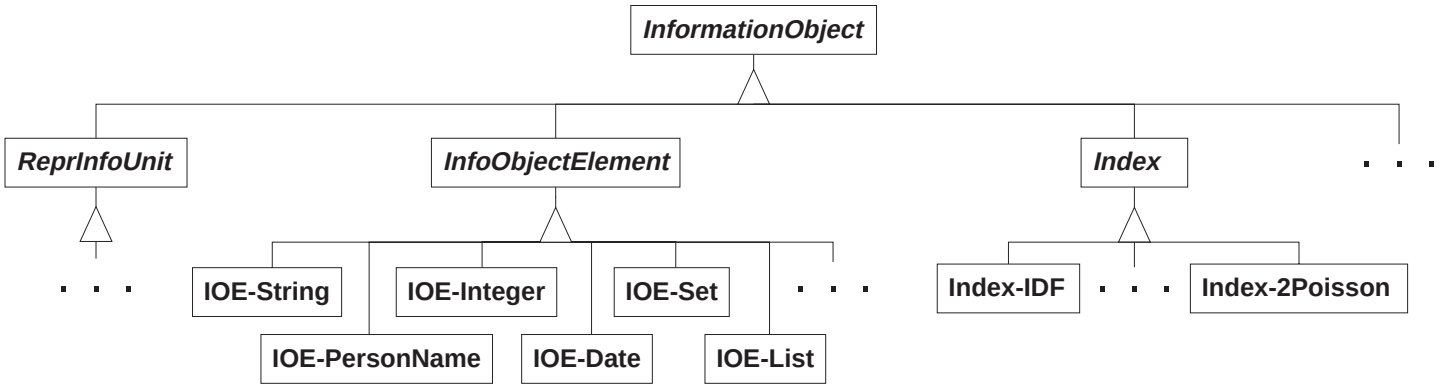

Figure 1: Overview of the most important classes of FIRE's object model

4 The Approach of FIRE

4.1 An Overview of FIRE

FIRE is designed to facilitate the development of IR applications and to support the experimental evaluation of indexing and retrieval techniques. In addition, the framework is supposed to provide basic functionalities for several media and should end up as a flexible and extensible tool that can be used for developing a wide range of IR applications. FIRE is an acronym for '**F**ramework for **I**nformation **Re**trieval Applications'. It is developed in cooperation with the IR group at the Robert Gordon University, U.K.

The design and implementation of FIRE are based on an object-oriented approach. The object model of FIRE defines the basic IR concepts and supplies functionalities that support the realization of an IR application. This section gives a short overview of FIRE focusing on the most essential classes of FIRE's object model. A more comprehensive description of FIRE's design and object model is given in [Son95].

FIRE represents documents by a set of features (or attributes). Note that the term document is used here in a broad sense: documents may consist of structured and unstructured parts and may be composed of different media. The class *ReprInfoUnit* (see Figure 1) models documents in a generic way. It defines methods which provide information about the modeling of a document representation as well as methods for accessing the features of a document representation. In addition, *ReprInfoUnit* and its subclasses are responsible for organizing the indexing and retrieval of documents of the respective type.

Concrete subclasses of *ReprInfoUnit* define how documents of a certain type, e.g., books, tables, etc., are represented in an application. When dealing with text for instance, a new subclass of *ReprInfoUnit* may be introduced, consisting of features like *Title*, *Authors*, *TextBody*, and *PublicationDate*. The modeling of concrete types of documents is not part of FIRE's object model as this is an application-specific task. Nevertheless, the framework supports the application developer in this task. The class *InfoObjectElement* and its subclasses provide a set of data types like string, integer, person name, date, set, and list (cf. Figure 1), which are intended to be used for the application-specific modeling of documents. If needed, the application developer may extend this set of data types by adding new subclasses. *InfoObjectElement* helps to reduce the effort for developing an IR application by providing an associated interface for indexing an information unit, determining the similarity of two units, etc. Further, this branch of the class hierarchy supports a uniform representation of document components, e.g., the author of a book is specified in the same way as the author of a chart table.

The class *Index* is the implementation of a typical IR-index, which not only manages a set of indexing features derived from a collection of documents but also provides information for calculating the probability of relevance of information units to the user's query. The class *Index* is a generic class, which solves general tasks but does not provide specific IR functionality. The latter is supplied by concrete subclasses of *Index* like *Index-IDF* and

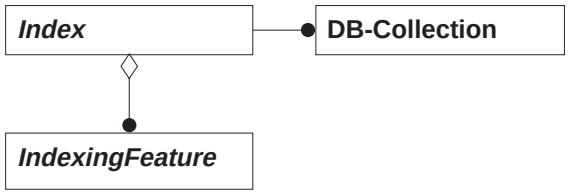

Figure 2: Structure of the class *Index*

Index-2Poisson, which implement particular weighting schemata (see [Har92m] for an overview of weighting schemata).

In the following two sections, we discuss the design and implementation of an IR-index in FIRE more comprehensively. First, we present the design of the class *Index* with focus on IR related issues. Then, we discuss some details of *Index* which support an efficient evaluation of queries and enable us to exploit the optimization facilities of the underlying DBMS.

4.2 Design of an IR-Index

An *Index* basically consists of a set of *IndexingFeature*s (see Figure 2). In addition, it may be associated with zero or more objects of the type *DB-Collection*, whose purpose is explained in the next section. An *IndexingFeature* consists of a feature, e.g., a normalized word from a text, and a source specification. The latter is essentially a reference to the document from which the feature has been derived. In addition, an *IndexingFeature* may specify a position within the source. Positional information is needed for indicating to the user why a particular information unit has been retrieved, which is done by 'highlighting' the relevant pieces of the unit. Furthermore, weighting schemata may use positional information, e.g., it may be assumed that words occurring in a heading are more important than words in regular paragraphs. Finally, an *IndexingFeature* is associated with the indexing method that has been used for deriving the feature. Information units may be indexed in many different ways, and in many cases more than one method can be applied for indexing a particular unit, e.g., weak or strong stemming when indexing textual units. For consistency reasons, it is important that all *IndexingFeature*s of an *Index* have been derived by the same method. Hence, FIRE associates *IndexingFeature*s with the indexing method applied, and checks whether the *IndexingFeature* to be added to an *Index* is compatible with the ones previously added.

The class *Index* defines a set of operations and methods (see Figure 3), which provide a uniform interface independent of any particular retrieval model. The method *addIFs* of *Index* incorporates a set of *IndexingFeature*s into an *Index*. The adding of features does not cause any updating activities like sorting the *Index* or recalculating indexing weights. Updates have to be invoked explicitly by an *update* message. We chose to separate the processes in order to avoid unnecessary computations, for instance, sorting an *Index* again and again when indexing a whole collection of documents. The class *Index* also provides a method, called *getIFsOf*, for determining the *IndexingFeature*s which have been derived from a particular source. By the method *retractIFsOf*, the *IndexingFeature*s referring to a given set of sources can be retracted, whereas the method *clear* removes all *IndexingFeature*s from an *Index*. Finally, the method *retrieve* serves for retrieving information by evaluating single query conditions. The results of the evaluation are passed to the query document, which is a *ReprInfoUnit* object. The query document invokes appropriate methods for combining the results and for computing the scores ('Retrieval Status Values', RSVs) indicating the estimated relevance of an information unit to the user's query.

FIRE supports an approximate matching on different data types. It allows one, for instance, to retrieve documents which cover a particular topic with a high probability and to retrieve documents which have a similar author name or a similar publication date. As discussed before, an approximate matching may be quite time consuming. In order to compensate computing efforts, an *Index* object may be instructed to support particular matching methods. This can be done by an authorized user at run-time via the method *supportMatchers*. Also, previous

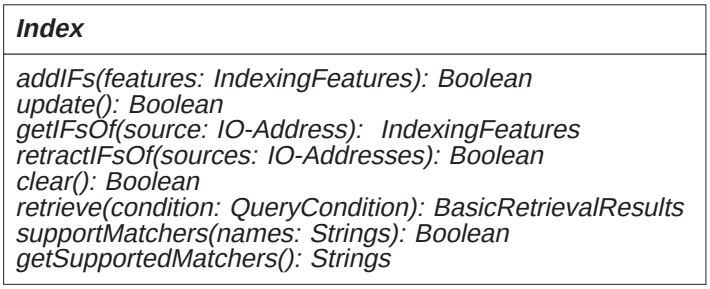

Figure 3: Operations and methods of *Index*

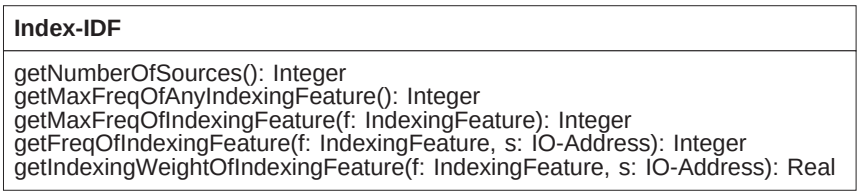

Figure 4: A concrete subclass of *Index*

instructions may be overwritten by new ones. Note that an *Index* always allows one to use any matching method applicable to the given type of *IndexingFeature*s, but performs supported methods more efficiently. The details of the optimization of the matching process are fully encapsulated by the class *Index*. This is advantageous as it avoids bothering the user with optimization details.

The subclasses of *Index* implement particular weighting schemata. They define additional methods which calculate the information required by the respective weighting schema. Figure 4 shows an example subclass of *Index* which implements a version of the 'Inverse Document Frequency' (IDF) weighting schema.

The subclasses of *Index* are not specialized to a particular type of indexing features. Hence, they can be used in different contexts and the application developer needs to define a new subclass only if an additional weighting schema is to be supported.

4.3 Improving Retrieval Efficiency

An *Index* may be associated with zero or more objects of the class *DB-Collection* (see Figure 2). A *DB-Collection* serves to improve retrieval efficiency. It exploits the optimization facilities of the DBMS and may support approximate matching methods.

An object of the class *DB-Collection* consists of a collection of *IndexEntry*s. An *IndexEntry* is composed of a key, which may be of any type, and a set of pointers to the *IndexingFeature*s of an *Index* which yield the same key. (Thus, *IndexEntry*s have essentially the same structure as entries of an inverted file.) The key of an *IndexEntry* is derived from the corresponding *IndexingFeature*, and may have the same value as the feature or may be assigned some code for optimizing access. The interface of *DB-Collection*, which is depicted in Figure 5, is similar to the interface of *Index*. However, *DB-Collection* is internally concerned with *IndexEntry*s rather than *IndexingFeature*s. The derivation of *IndexEntry*s from *IndexingFeature*s is invoked by *DB-Collection*.

A *DB-Collection* utilizes the query optimization facilities of the underlying DBMS by instructing the DBMS to create an appropriate DB-index for the given collection of *IndexEntry*s. Since the DBMS does not index document representations but collections of *IndexEntry*s, we can apply IR indexing techniques without being restricted in any way by the DBMS. Nevertheless, we can take advantage of the DBMS's facilities for optimizing the evaluation of queries. As a further advantage, FIRE does not need to provide any specialized index structures (B-trees,

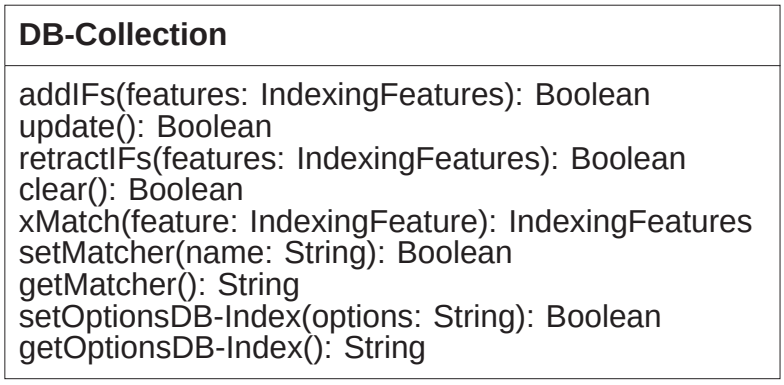

Figure 5: Interface of *DB-Collection*

hash tables, etc.) as well as query optimization strategies, but can rely on ObjectStore's means.

In addition, *DB-Collection* serves to optimize approximate matching. When we try to find the objects of a collection which are similar to a given object, we have to consider all objects. This is in contrast to an exact matching where search can be restricted in most cases, e.g., by a binary search in an ordered index. To reduce complexity, FIRE allows one to perform the approximate matching in a *restricted* form, which is a combination of an exact and an approximate matching. In this matching mode, a key is generated in a first step for the *IndexingFeature* given with the query condition. Such a key may be for instance a phonetic code for a person name. Then the corresponding *DB-Collection* is consulted and its *IndexEntry*s looked-up to determine the *IndexingFeature*s for which the same key has been generated. Finally, a full approximate matching is performed with the selected *IndexingFeature*s. Note, an *Index* may support more then one approximate matching method at the same time by creating different *DB-Collection*s. This is useful, since different retrieval situations may require different matching methods. In the case no appropriate *DB-Collection* exists for the matching method to be performed, the *Index* does the look-up itself, of course in a less efficient way, by going through the associated set of *IndexingFeature*s.

The derivation of keys from *IndexingFeature*s is a matter for the matching algorithms, since they know best how to build appropriate keys. Also, the matching algorithms determine which kind of DB-index is most appropriate for the resulting keys. In FIRE, matching algorithms are represented by specialized classes rather than by methods of other classes. This design decision allows the user to browse through the class hierarchy in order to see which matching algorithms are available and how they are to be used; see [Son96] for further details. The classes implementing particular matching algorithms are subclasses of an abstract class called *Matcher*, which is depicted in Figure 6. The important features of this class with regard to the current topic are the method *key* for deriving keys from *IndexingFeature*s (or *InfoObjectElement*s) and the attribute *OptionsDB-Collection* for specifying the DB-index options to be applied for supporting a particular matcher.

The design of the class *Matcher* and its subclasses eases the optimization of approximate matching methods. It is fully sufficient to instruct an *Index* to support certain *Matcher*(s). Knowing the names of the matching algorithms to be supported, the *Index* itself can gather the necessary details by asking the proper *Matcher*(s).

Conclusions

In this paper, we have reviewed previous attempts to use DBMSs for implementing IR systems and pointed out some shortcomings of DBMSs with regard to the support of IR systems. Furthermore, we have presented FIRE's approach for using the functionality of an object-oriented DBMS in an IR system. FIRE allows one especially

- to represent complex heterogeneous information in a natural way,

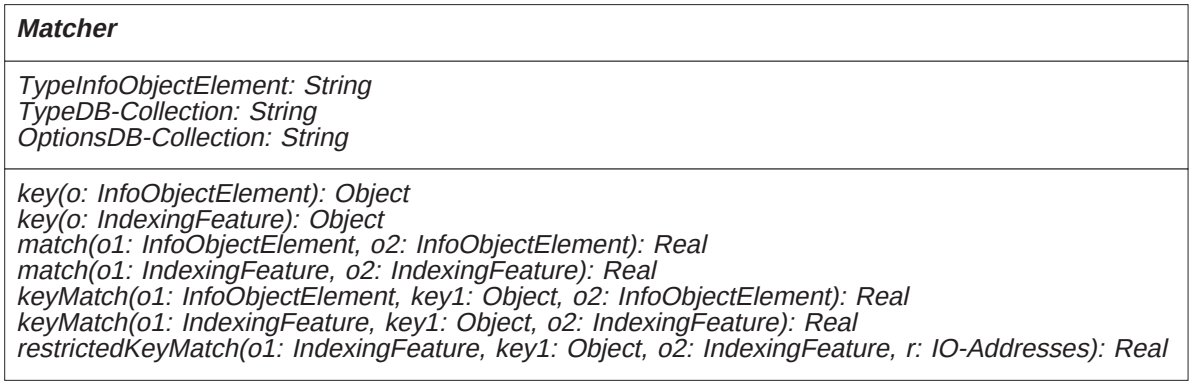

Figure 6: Class *Matcher*

- to utilize the query optimization facilities of the underlying DBMS for IR purposes, and
- to reduce the complexity of an approximate matching.

In the near future, we will perform experiments to quantify the effect of the DBMS's optimization facilities in IR applications and to test the performance of the restricted approximate matching.

Acknowledgments

The author would like to thank Tore Bratvold, UBILAB, and Andreas Geppert from the University Zurich for many fruitful discussions on the design of FIRE and the integration of DBMSs and IR systems. Thanks are also due to Manuel Bleichenbacher, UBILAB, for his work on ETOS, which is the software development platform used for implementing FIRE, and his technical assistance with ETOS.

References

[Bla88] D.C. Blair: An Extended Relational Document Retrievahrl Model. In: *Information Processing & Management*, Vol. 24 (3), pp. 349-371, 1988

[Cro92] W.B. Croft, L.A. Smith, and H.R. Turtle: A Loosely-Coupled Integration of a Text Retrieval System and an Object-Oriented Database System. In: *Proc. of the 15th Annual Int. ACM SIGIR Conference on R & D in Information Retrieval (SIGIR-92)*, pp. 223-232, 1992

[Fuh92] N. Fuhr: Integration of Probabilistic Fact and Text Retrieval. In: *Proc. of the 15th Annual Int. ACM SIGIR Conference on R & D in Information Retrieval (SIGIR-92)*, pp. 211-222, 1992

[Gar90] M. Garcia-Molina and D. Porter: Supporting Probabilistic Data in a Relational System. In: *Proc. of EDBT*, pp. 60-74, 1990

[Gu93] J. Gu, U. Thiel, and J. Zhao: Efficient Retrieval of Complex Objects: Query Processing in a Hybrid DB and IR System. In: G. Knorz, J. Krause and C. Womser-Hacker (eds.): *Information Retrieval '93, Von der Modellierung zur Anwendung*, pp. 67-81, Konstanz/Germany: UVK, 1993

[Har92m] D. Harman: Ranking Algorithms. In: W.B. Frakes and R. Baeza-Yates (eds.): *Information Retrieval: Data Structures & Algorithms*, Englewood Cliffs/N.J.: Prentice Hall, 1992

[Har92p] D.J. Harper and A.D.M. Walker: ECLAIR: An Extensible Class Library for Information Retrieval. In: *The Computer Journal*, Vol. 35 (3), pp. 256-267, 1992

[Lam91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb: The ObjectStore Database System. In: Communications of the ACM, Vol. 34 (10), pp. 50-63, 1991

[Lyn88] C.A. Lynch and M. Stonebraker: Extended User-Defined Indexing with Applications to Textual Databases. In: Proc. of the 14th *International Conference on Very Large Data Bases*, pp. 306-317, 1988

[Mac83] I.A. Macleod and R.G. Crawford: Document Retrieval as a Database Application. In: *Information Technology: Research and Development*, Vol. 2, pp. 43-60, 1983

[Mac87] I.A. Macleod and A.R. Reuber: The Array Model: A Conceptual Modeling Approach to Document Retrieval. In: *Journal of the American Society for Information Science*, Vol. 38 (3), pp. 162-170, 1987

[Mac91] I.A. Macleod: Text Retrieval and the Relational Model. In: *Journal of the American Society for Information Science*, Vol. 42 (3), pp. 155-165, 1991

[Mot88] A. Motro: VAGUE: A User Interface to Relational Databases that Permits Vague Queries. In: *ACM Transactions of Office Information Systems*, Vol. 6 (3), pp. 187-214, 1988

[Sch82] H.J. Schek and P. Pistor: Data Structures for an Integrated Database Management and Information Retrieval System. In: *Proc. of the 8th International Conference on Very Large Data Bases*, pp. 197-207, 1982

[Sme90] A. Smeaton: Retriev: An Information Retrieval System Implemented on Top of a Relational Database. In: *Program*, Vol. 24 (1), pp. 21-32, 1990

[Son95] G. Sonnenberger and H.-P. Frei: Design of a Reusable IR Framework. In: *Proc. of the 18th Annual Int. ACM SIGIR Conference on R & D in Information Retrieval (SIGIR '95)*, pp. 49-57, 1995

[Son96] G. So, T.A. Bratvold, and H.-P. Frei: Use and Reuse of Indexing and Retrieval Functionality in a Multimedia IR Framework. (Paper presented at the final MIRO Workshop in Glasgow, Sept. 1995; publication forthcoming)

Integrating INQUERY with an RDBMS to Support Text Retrieval

Vasanthakumar S. R.,*James P. Callan, and W. Bruce Croft

Department of Computer Science
University of Massachusetts, Amherst, MA 01003, USA
vasant@cs.umass.edu

Abstract

Information is a combination of structured data and unstructured data. Traditionally, relational database management systems (RDBMS) have been designed to handle structured data. IR systems can handle text (unstructured data) very well but are not designed to handle structured data. With present day information being a combination of structured and unstructured data, there is an increasing demand for an IR-DBMS system that incorporates features of both IR and DBMSs. We discuss a framework that incorporates powerful text retrieval in relational database management systems. An extended SQL with probabilistic operators for text retrieval is defined. This paper also discusses an implementation of the probabilistic operators in SQL.

1 Introduction

The state of the art is that much information, especially multi-media, is represented as a combination of both structured and unstructured data. Structured data comprises data types like integer, real, fixed-length string; unstructured data comprises text, images, audio *etc*. Structured data has been efficiently stored and retrieved using relational database management systems (RDBMS). Text, an unstructured component of information, has been traditionally stored and retrieved using Information Retrieval (IR) systems. RDBMSs use exact matching to retrieve data. while IR systems use approximate matching. IR systems are not suitable for structured data and RDBMSs are not suitable for unstructured data. RDBMSs have the additional advantage of addressing the issues of concurrency, recovery, security and integrity, while most IR systems don't. The gap between structured and unstructured components in data has been recently narrowed (*e.g.*, medical information systems, pharmaceutical systems) and has demanded a system that incorporates the features of both RDB and IR systems.

Our goal is to add powerful *text* retrieval capabilities to an RDBMS using the relational framework and SQL. Regular boolean operators are used on the non-text attributes and probabilistic operators are used on the text

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This research was supported in part by Center for Intelligent Information Retrieval and Digital Equiment Corporation.

attributes. The probabilistic answers are converted into boolean values and later combined with the results of the non-text component of the query. The final result set is ranked on the probability (belief) that a record is relevant to the text query.

We have an existing information retrieval system INQUERY [2]. We are experimenting on implementing the same retrieval strategy in a relational database management system, DEC Rdb. Such an implementation will add powerful text retrieval capabilities to an RDBMS, facilitating the construction of IR systems that have all the features we get from an RDBMS (concurrency, recovery, *etc.*) [1]. This paper discusses the issues, our experiences and status.

2 Integrating IR and RDBM Systems

Integrating IR and RDBMS could be viewed at different levels:

- a loosely-coupled IR/RDBMS system, and
- a tightly-coupled IR/RDBMS system.

2.1 Loosely-coupled IR/RDBM system

A loosely-coupled IR/RDBMS system can be viewed in different ways:

- IR system as an application of RDBMS,
- A Hybrid of IR and RDBM systems, and
- Using RDBMS for storing IR data structures.

2.1.1 IR System as an Application of RDBMS

We could build an IR system as a RDB application without any major modification to the existing RDBMS [6]. These applications are based on "exact matching", and query evaluation is "boolean" in nature. Probabilistic evaluation of queries is very effective for *text retrieval*. Blair [1] uses the concept of probability for ranking the records. The inability of such systems to handle fuzzy queries results in an IR system with poor retrieval performance (low *precision* and *recall*). Also, IR data structures tend to vary in size greatly, and thus the application would be inefficient.

2.1.2 A Hybrid Approach

A hybrid IR/DB system utilizes both an IR and DB system. An *embedded full integration* is proposed by Gu *et al.* [5]. This approach proposes the use of two distinct systems, an IR system (INQUERY) and RDBMS (Sybase). The inverted lists for the *text* fields in the RDBMS tables are stored in INQUERY. An extended SQL (ESQL) is proposed which has both boolean and IR operations. A form-based IR interface is provided for the end users and the user's intention is interpreted into a program described by a query language called ESQL which is an extension of SQL. The ESQL program is then translated to a standard SQL program and an INQUERY query by a *parser and interpreter*. The INQUERY query is sent to *ProcINQUERY* - an INQUERY version which can be invoked as a procedure, and output the information about ranked textual data into Sybase. The SQL query is then sent to Sybase which searches the corresponding data based on the outputs of the ProcINQUERY. The disadvantages of such an approach are that we use two different systems, and we lack flexibility in combining IR and boolean parts of the query. This motivates us to develop an RDBMS system which does not make use of any IR system, but instead, stores all the IR data structures in the RDBMS and implements all the IR operators in SQL itself. Section 3 explains our approach to achieve the above mentioned goal.

2.1.3 Using RDBMS for storing IR data structures

Information retrieval systems index unstructured text into an *inverted index* or *inverted file* [8]. For each term a separate index is constructed that stores the record identifiers, or document identifiers, for *all* the records containing that term. With an inverted index, the record set corresponding to a given query formulation is easily determined. The identifiers for all retrieved items can be obtained by extracting from the inverted index the list of record identifiers corresponding to each query term and combining these record identifiers appropriately. For supporting probabilistic retrieval the term statistics are also stored along with record identifiers. In order to support complex query operators like *phrase* or *proximity*, the locations of each occurrence of the term in a record are also stored (*proximity* information). The number of tuples of an inverted file is huge when compared to the number of documents or records they represent. Thus the number of tuples in an inverted file will be the number of unique terms in the documents times the average number of terms in a document. As stated by Blair [1], any discussion of database management system implementation must address the controversial issue of processing speed. Traditional beliefs tend to hold that relational systems trade flexibility of query and database structuring for reduced processing speeds. In order to overcome this bottleneck, we sacrificed some flexibility and reduced the number of tuples to the number of unique terms in the collection or documents database. The term statistics and the proximity information are stored in a *binary object* or *blob*. We used the INQUERY information retrieval system and DEC Rdb RDBMS for this experiment. The following paragraphs discuss the lessons learned from this implementation.

Rdb was used to store all of INQUERY's file-based data structures (inverted file, *db* file, and term dictionary). The inverted list file contained term ids and their inverted lists. The *db* file contained document indices necessary for providing user interface functions in the API. In this implementation, Rdb did not know the internal structure of the inverted list and the *db* file. The encoded inverted list and *db* file information was stored in Rdb as blobs. An SQL-based interface was used between INQUERY and Rdb. There was an overhead in this implementation for INQUERY to decode the blob and extract the required information. Other than the query language, all the other features like concurrency control, recovery, *etc.* , (refer Section 1) of an RDBMS were exploited without sacrificing performance during document indexing and query evaluation.

2.1.4 Results of *Blob* Implementation

The results from the *blob* implementation are compared with the *keyfile* implementation of INQUERY in Table 2. *Keyfile* is a B-tree package which is used in INQUERY to store all its data structures. The experiments were done on a DEC Alpha running OpenVMS with 80 Mbytes of main memory. DEC Rdb V 6.0 was used. Several test databases with different characteristics were used to analyze the performance of the keyfile and Rdb versions of INQUERY. Table 1 shows the characteristics of the test databases used. Table 2 shows that the elapsed time of the Rdb version is in the same order of magnitude as that of the keyfile version and we get all the advantages of using an RDBMS.

2.2 Tightly-coupled IR/RDBM system

Different approaches have been proposed for a tighter integration of IR and RDBM systems [4, 7, 1, 9]. All these methods suggest implementing a new system incorporating the proposed theories. Schek *et al.* [9] propose an extension to the relational model allowing Non First Normal Form (NF^2) relations. They propose extensions to relational algebra with emphasis on new "nest" and "unnest" operations, which transform between first normal form relations and NF^2 ones. This allows the attribute domains to be sets and sets of sets, suitable for IR (*e.g.*, list of words as a single attribute).

Fuhr [4] proposes a probabilistic relational model which combines relational algebra with probabilistic retrieval. He proposes a special join operator implementing probabilistic retrieval. This model retrieves not only

Table 1: Characteristics of test databases

Attribute	*Database*		
	cacm	*arman*	*wsj89*
Raw data	3 Mbytes	10 Mbytes	39 Mbytes
Number of documents	3204	628	12380
Number of unique words	5942	31838	68058
Total number of words	383182	907668	5451898
Number of transactions	112599	388066	2606670
Number of queries	50	50	50
Average number of words per query	7	94	94

Table 2: Resources used by INQUERY v1.6 on different test collections

Performance Metric	*Collection*					
	cacm		*arman*		*wsj89*	
	Keyfile	*Rdb*	*Keyfile*	*Rdb*	*Keyfile*	*Rdb*
Buffered I/O count	98	154	124	226	125	233
Peak working set size	4896	31072	6496	32120	17616	40960
Direct I/O count	345	406	1813	937	12595	3544
Peak page file size	19040	81760	20656	85072	329286	106496
Page faults	314	3174	424	4239	1193	21704
Charge CPU time (seconds)	9	12	20	66	111	163
Elapsed time (seconds)	15	22	40	85	221	246

documents but also any kind of objects. Further, probabilistic retrieval provides implicit ranking of these objects. Fuhr argues that with independence assumptions, the relational model is a special case of this probabilistic relational model.

The above approaches demand a new design of the DBMS. This is expensive and would satisfy only IR requirements. Instead we propose a method in which the probabilistic retrieval can be done in the existing relational framework and also suggest ways to implement special join operations using SQL. Section 3 explains how IR data structures can be stored in an RDBMS so that SQL can be used on them to support probabilistic operators and special joins.

3 Adding *Text* Retrieval Capabilities to RDBMS

The two main issues that must be addressed in order to add IR capabilities to an RDBMS are the storage of IR data structures and query language support for IR operators. We observe from Section 2.1.3 that the blob implementation to store IR data structures in RDBMS is quite effective. Since our framework proposes to implement IR operators using SQL, we have a requirement that the data structures be accessible through SQL, obviously a table. If IR data structures are stored as regular tables, then it leads to poor data storage and retrieval performance.. To solve this problem, *Cooperative indexing* [3] can be used for efficient storage and retrieval. In this approach, the IR components of the system define what is extracted from documents (text attributes) along with the related index structure, and the database system provides efficient access to the index. The cooperative index

can be accessed, like any regular table, through SQL. Our main focus here is to provide support for IR operators in the query language and a method to evaluate such complex queries.

3.1 Retrieval Model

Our text retrieval framework is based upon a type of Bayes net called a *document retrieval inference network* [10, 2] (which is used in INQUERY). The inference net has two components *i.e.,* the document network and the query network. The document network represents the content of the text and the query network represents the need for information. This framework creates a document network for the *text* attributes, creates a query network for the *text* component of the query, and uses the network to retrieve records that satisfy the *text* query. The result from the text and the non-text query components are combined to obtain the final result.

The document network is created automatically by mapping *text* attribute onto content representation nodes, and storing the nodes in an inverted file for efficient retrieval. For each term a separate index is constructed that stores the record identifiers, term statistics and term position information for *all* the records identified by the term. This information is stored in a relational table, say INV_LIST (TERM, DOC_ID, TF, MAX_TF, PROX), where TF is the term frequency, MAX_TF is the maximum term frequency and PROX is the position information.

3.2 Extending SQL to Support IR Operators

Text retrieval is based on partial matching and inference and thus returns scores (beliefs) as answers. These beliefs represent the relevance of a particular document (record) to the query. The traditional SQL operators are not suitable for handling beliefs since SQL operators are boolean in nature. Thus, additional text handling operators need to be added to SQL, as well as methods to combine the results from such operators with the traditional boolean operators. An extended SQL (ESQL) is defined as follows to support *text* retrieval. The ESQL will have a non-text component and a text component. The non-text component uses the regular *WHERE* conditions and operators of SQL. The following probabilistic operators [10, 2] are supported in the text component:

PAND: Probabilistic ("fuzzy") *and* of the terms in the scope of the operator.

POR: Probabilistic *or* of the terms in the scope of the operator.

PNOT: Probabilistic *negation* of the term in the scope of the operator.

PSUM: Value is the mean of the beliefs in the arguments.

PWSUM: Value is the mean of the weighted beliefs in the arguments.

Here it is assumed that all the probabilistic operators are localized to a subtree of an ESQL query. An example ESQL query on a table DOCUMENTS (DOC_ID, DATE_PUBLISHED, AUTHOR, TEXT) to get records about "operating system design" and published after "04/30/90" will be:

Example 1:

```
SELECT DOC_ID
  FROM DOCUMENTS
  WHERE DATE_PUBLISHED > '04/30/1990'
    AND TEXT_QUERY( TEXT CONTAINS 'operating'
                    PAND ( TEXT CONTAINS 'systems'
                           POR TEXT CONTAINS 'design') );
```

The query tree for the text component of the query in Example 1 is shown in Figure 1(a).

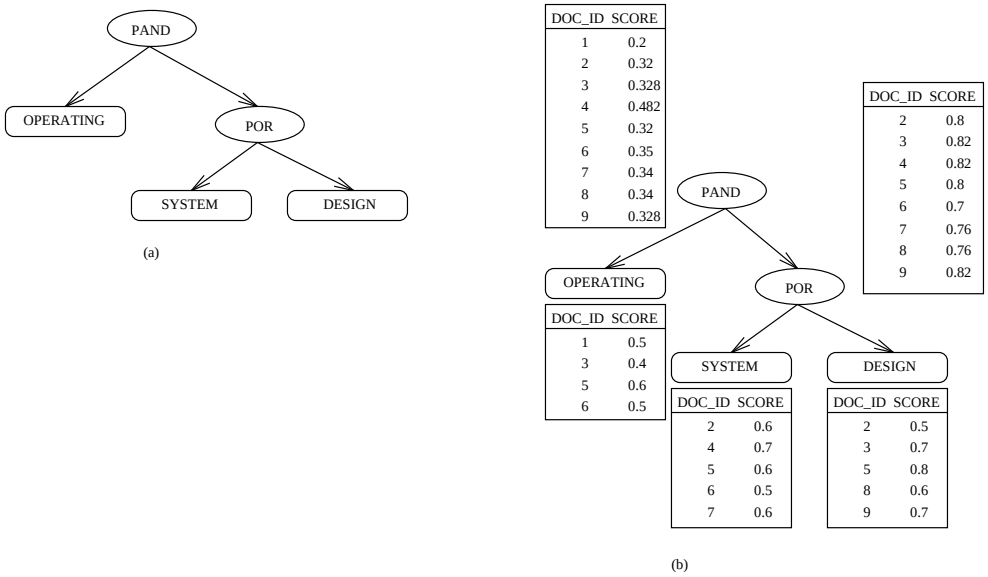

Figure 1: (a) Text query tree (b) Belief lists at different nodes

3.3 Query Evaluation

An ESQL parser is used to divide the query into *text* and *non-text* components. The non-text component is evaluated using regular SQL statements. The text component is evaluated using SQL statements with external functions. External functions are used to support IR operators. The result set from such an evaluation has record identifiers and belief scores. The result set is sorted in the descending order of belief scores. A threshold is applied to the result set for the text component. The threshold can be either the top n records or records greater than a specific threshold (say 0.4). Finally, the result set from the non-text component is used as a filter to generate the final result set.

A query network is created from the *text* component of the user query. In this section we show how the *text* component of the query can be evaluated using SQL and later combined with the non-text component. The query evaluation can be term-at-a-time or record-at-a-time.

3.4 Term-at-a-Time Processing

In term-at-a-time processing, each node in the query tree is evaluated for all documents or records. We evaluate the tree bottom up, as follows.

3.4.1 Generating belief lists at the leaf nodes

Belief lists are generated for each leaf node. A belief list is a list of record identifiers and associated belief values at a given node, as well as default beliefs and weights. Node belief scores are calculated [10, 2] and normalized using the statistics stored in the inverted list (INV_LIST table). Belief lists can easily be generated from INV_LIST in SQL. External functions [12] are used to calculate the belief scores. The belief lists at the leaf nodes for Example 1 are shown in Figure 1(b).

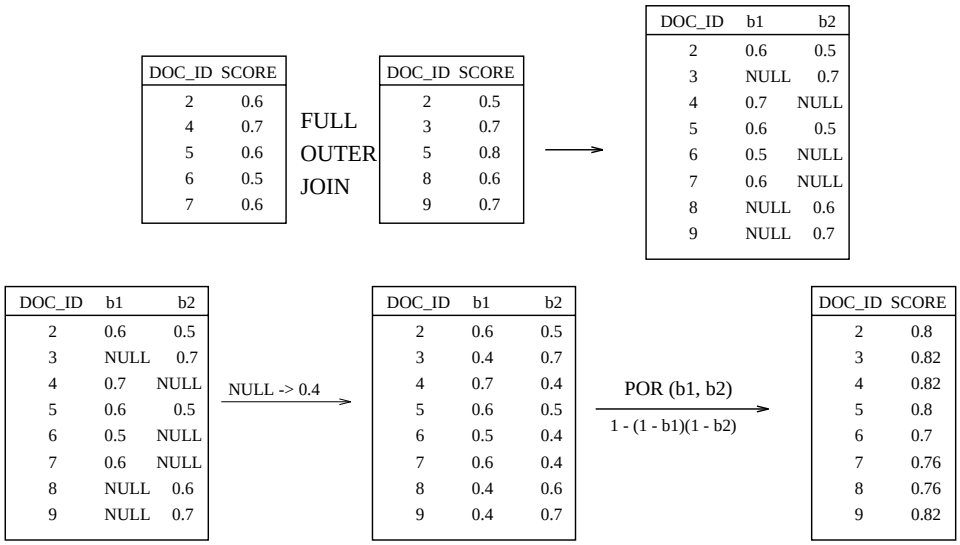

Figure 2: Special Join operation

3.4.2 Evaluating probabilistic operators

The probabilistic operators in the query tree are evaluated with a bottom-up strategy. Each operator is evaluated by executing a special join operation (different for different operators) on the belief lists of its children. A special join is achieved in two steps:

- A *full outer join* [12] of the two belief lists of the children is done, replacing all the NULLs with a default belief value, such as 0.4.
- Combine the two belief values for each record using the formula for each operator [10]:
  1. POR: $1 - (1 - b_1)(1 - b_2)$
  2. PAND: $b_1 * b_2$
  3. PNOT: $1 - b_1$
  4. PSUM: $\left(\frac{b_1+b_2}{2}\right)$
  5. PWSUM: $\left(\frac{(w_1 b_1+w_2 b_2)w}{w_1+w_2}\right)$ where w_1 and w_2 are weight associated with the child nodes.

  This is also implemented using external functions.

The output of the special join is again another belief list. By evaluating all the nodes, bottom-up, we will finally have a belief list at the root, which is a list of record identifiers and belief values. The special join operation for POR node in Example 1 is shown in Figure 2.

3.4.3 Generating the final result set

The non-text component of the ESQL query is later applied as a filter on the result set of the previous step to obtain the final result set. The belief values in the belief list are used to rank order the result set. The `ORDER BY` clause in SQL can be used to rank order the records. The number of records in the result set is restricted by either using a *threshold* on the belief value or by using the *top* n records. If threshold is used, then a `WHERE` clause

like `BEL > 0.3` can be used, where `BEL` is the belief for this document. If the top n strategy is used, then a condition like `LIMIT TO n ROWS` can be used.

The SQL statement for evaluating the text component of the ESQL query of Example 1 is as follows:

```
SELECT DOC_ID, (1 - (COALESCE(T1.B1, 0.4) * COALESCE(T2.B2, 0.4)))
  FROM ((SELECT DOC_ID, BEL(TF, MAX_TF, DOC_FREQ)
           FROM INV_LIST
           WHERE TERM = 'system' ) AS T1 (DOC_ID, B1)
         FULL OUTER JOIN
         (SELECT DOC_ID, BEL(TF, MAX_TF, DOC_FREQ)
           FROM INV_LIST
           WHERE TERM = 'design' ) AS T2 (DOC_ID, B2)
        ) AS T4 (DOC_ID, B4);
```

Here `BEL()` is an external function which calculates the belief score for a record.

3.5 Record-at-a-Time Processing

In contrast to term-at-a-time processing, where each query node is evaluated for all the records, in this method the entire query tree is evaluated for each record. This can be better because it avoids the expensive special joins. The non-text query is used as a filter to obtain the record set on which the text query is evaluated. For each record in the filtered record set, we do the following:

Step 1: Calculate the belief value for all the leaf nodes (query terms) for this record. The belief value is calculated from the inverted list (INV_LIST) as discussed in Section 3.4.

Step 2: Evaluate the entire query tree for this record. All the probabilistic query operators are implemented as external functions [12]. These external functions take two belief values as their arguments and return another belief value. Thus these external functions can be nested. Since nesting can be done, the entire query is easily implemented. If b1, b2 and b3 are the belief scores for a specific record (say `DOC_ID = ID_1`) at the leaf nodes for Example 1, the text component is evaluated as shown in the SQL statement below. Here `PAND` and `POR` are external functions. It should be noted that b1, b2, and b3 are themselves SQL statements which calculate belief scores from the term statistics stored in INV_LIST.

Step 3: A threshold is applied on the final belief b (*e.g.*, b $>$ 0.5) to convert the probability into a boolean result similar to the method of Gu *et al.* [5].

```
SELECT DOC_ID FROM DOCUMENTS
       WHERE DOC_ID = ID_1
       AND PAND (b1, POR (b2, b3)) > 0.5
       AND DATE_PUBLISHED > '04/30/1990';
```

The result set is ranked in the descending order of belief using the `ORDER BY` SQL clause to obtain the final result. The SQL statement for the entire ESQL query is:

```
SELECT DOCUMENTS.DOC_ID,
  PAND ((SELECT COALESCE (BEL(TF, MAX_TF, DOC_FREQ), 0.4)
           FROM INV_LIST
           WHERE TERM = 'operating'
           AND DOCUMENTS.DOCID = INV_LIST.DOCID
```

```
        ), POR ((SELECT COALESCE (BEL(TF, MAX_TF, DOC_FREQ), 0.4)
                    FROM INV_LIST
                    WHERE TERM = 'system'
                    AND DOCUMENTS.DOCID = INV_LIST.DOCID
                  ), (SELECT COALESCE (BEL(TF, MAX_TF, DOC_FREQ), 0.4)
                        FROM INV_LIST
                        WHERE TERM = 'design'
                        AND DOCUMENTS.DOCID = INV_LIST.DOCID
                       ))) AS BEL
  FROM DOCUMENTS;
```

This SQL statement would generate a table of `DOC_ID` and `BEL` for all the documents in the DOCUMENTS table. It should also be noted here that the `DOC_FREQ` in the above SQL statement is again an SQL statement like

```
SELECT COUNT(*) FROM INV_LIST WHERE TERM = 'operating';
```

Even though both the *term-at-a-time* and *record-at-a-time* approaches return the same result set, the latter has an advantage in speed since there are no JOIN operations, which tend to be expensive. More optimization can be added in Step 2, by choosing a small set of records to evaluate the query on, depending on the operators in the query.

3.6 Evaluating PROXIMITY Operators

PROXIMITY operators are those which rely on the the relative positions of the terms in a document. Some examples of PROXIMITY operators [2] are

P#n: A match occurs whenever all of the arguments are found, in order, with fewer than n words separating adjacent arguments. For example A P#3 B matches "A B", "A c B" and "A c c B".

PHRASE: Value is a function of the beliefs returned by the P#3 and PSUM operators. The intent is to rely upon full phrase occurrences when they are present, and to rely upon individual words when full phrases are rare or absent.

Evaluating proximity operators is much more complicated than evaluating the simple operators explained in earlier sections. These operators require *proximity lists* for evaluation. A proximity list contains statistical and proximity (term position) information by document for a particular term. The proximity lists should be instantiated at the term nodes of the proximity operator nodes and propagated upwards. The proximity lists are converted into belief lists before being propogated to simple operators. Proximity lists are transformed into belief values using the information in the list, and are combined using weighting or scoring functions. Belief lists may be computed from proximity lists but the reverse derivation is not possible. Creating, merging, and transforming proximity lists can all be implemented partly as external functions and partly in SQL.

4 Conclusion

An RDBMS can handle text more efficiently by storing inverted lists of the text fields in cooperative indexes, and SQL can be used to support IR operators. An extended SQL can be defined with additional IR operators. A pre-processor can be designed to transform the ESQL query into the corresponding SQL query. Performance completely depends on how efficiently cooperative indexing is implemented. More efficient implementations can be done by modifying the SQL engine to support the probabilistic operators. In this paper, we have assumed

that there exists only one text field, but there can be any number of text fields, with one cooperative index for each text field. The corresponding cooperative index should be selected during ESQL processing. With such a system, both structured and unstructured data can be handled efficiently and effectively without designing a totally new system. We are presently looking at allowing probabilistic operators anywhere in the query without the restrictions that they occur together, and the impact of such a design on precision and recall.

References

[1] David. C. Blair. An Extended Relational Retrieval Model. *Information Processing and Management*, 24(3):349–371, 1988.

[2] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of the Third International Conference on Database and Expert Systems Applications*, 78–83, Valencia, Spain, 1992. Springer-Verlag.

[3] Samuel DeFazio, Amjad Daoud, Lisa Ann Smith, Jagadishan Srinivasan, Bruce Croft, and Jamie Callan Integrating IR and RDBMS using cooperative indexing In *Proceedings of SIGIR 95*, ACM, July1995.

[4] N. Fuhr. A probabilistic model for the integration of IR and databases. In *Proceedings of SIGIR 93*, 309–317, ACM, June 1993.

[5] Junzhong Gu, Ulrich Thiel, and Jian Zhao. Efficient Retrieval of Complex Objects: Query Processing in a Hybrid DB and IR System. In *GESELLSCHAFT FUR MATHEMATIK UND DATENVERARBEITUNG MBH.*

[6] L. A. Macleod and R. G. Crawford. Document Retrieval as a Database Application. In D. K. Harman, editor, *Information Technology: Research and Development*, 2:43–60, 1983.

[7] A. Motro. VAGUE: A User Interface to Relational Database that Permits Vague Queries. In *ACM Transactions on Office Information Systems*, Vol 6, No 3, 187–214. July 1988.

[8] G. Salton. *Automatic Text Processing*. Addison-Wesley Publishing Company, 1989.

[9] H. J. Schek and P. Pistor. Data Structures for an Integrated Database Management and Information System. *Proceedings of the Eighth International Conference on Very Large Data Bases*, 197–206. 1982.

[10] Howard R. Turtle and W. Bruce Croft. Efficient probabilistic inference for text retrieval. In *RIAO 3 Conference Proceedings*, 644–661, Barcelona, Spain, April 1991.

[11] P. Cotton ISO-ANSI Working Draft SQL Multimedia Application Packages (SQL/MM) - Part 2: Full-text. ISO/IEC SC21/WG3 N1679, SQL/MM SOU-004, March 1994.

[12] The DEC Rdb Version 6.0 Documentation Kit, Digital Equipment Corporation, 1995.

An OODBMS-IRS Coupling for Structured Documents

Marc Volz, Karl Aberer, and Klemens Böhm
GMD-IPSI, Dolivostraße 15,
64293 Darmstadt, Germany,
{volz, aberer, kboehm}@darmstadt.gmd.de

Abstract

Requirements of modern Hypermedia Document Systems include support for structured documents and full DBMS and IRS functionality.[1] *An objective of DBMSs is to store and facilitate updates of highly structured and typed data. In conventional IRSs, however, documents are perceived as flat. Uncertainty, on the other hand, is a principal notion in the IRS context. In order to combine the advantages of both DBMSs and IRSs we have coupled the IRS INQUERY with the OODBMS VODAK. To model document structure, we have chosen SGML. SGML documents are stored in the OODBMS, and additional full text indexing is done by the IRS. The coupling consists of OODBMS classes encapsulating the IRS functionality. As SGML elements are modeled by database objects in an object oriented framework, queries can be modeled by means of methods—including content based queries.*

1 Introduction

Due to the proliferation of information highways and digital libraries, administering very large hypermedia document bases is becoming more and more important. Existing systems do not yet provide all the functionality needed by the information systems of the future. As an example, consider the MultiMedia Forum (MMF) [Sue+94], an interactive online journal from our institute. There are different ways in which articles of this journal may be accessed: via the table of contents of a particular issue, by means of navigation, or by means of specific database queries. In more detail, queries may refer to document characteristics, e.g., all press releases may be selected. As MMF-documents are SGML documents ([Bry88]) that conform to a proprietary document-type definition, this is feasible. Database functionality is essential, because the document stock may be modified by the editors concurrently with other access operations. Moreover, one would like to be able to formulate one's information needs with a certain degree of vagueness with regard to document content. The content may be either in textual form or of a continuous media type.

In this article, we describe the approach we have taken to build a system incorporating these characteristics. Some of the concepts described here are dealt with in [VAB96] in greater detail.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] In this article, we use the following abbreviations: DBMS = Database Management System, IRS = Information Retrieval System, OODBMS = Object Oriented Database Management System, SGML = Standard Generalized Markup Language.

1.1 IRS and OODBMS Features

In this section we briefly summarize some of the main IRS and OODBMS features relevant for Hypermedia Document Systems. IRSs manage sets of independent documents called "collections". In most IRSs documents are seen as a list of words. No inner structure (e.g., hierarchies of chapters, sections, and subsections) is supported. Collections are mapped to an index structure (often an inverted list of words), which is usually stored within the file system. Indexing is a complex process, where, e.g., word stemming and thesauri are used. An IRS query usually consists of terms (words) and is against the documents in a collection. During query processing, the IRS computes relevance values for each document. The result is a ranked list of documents that are supposed to be relevant to the query. A central aspect of IRSs is that uncertainty is taken into account in index structures, in queries, and in the matching process during query processing.

OODBMSs can store highly structured data. According to [Atk+89], features of an OODBMS are persistence, concurrency control, recovery, and declarative access (from the DBMS perspective); complex objects, object identity, encapsulation, types and classes (including inheritance), and extensibility (from the OO perspective).

1.2 Analysis of Hypermedia Document System

The following properties should be supported by a hypermedia document management system:

(1) Support for structured documents: The document model should include support of hierarchies and hyperlinks.

(2) Support of full DBMS functionality.

(3) Support of IRS functionality: Regular expression search in text documents is not sufficient.

With regard to (1), document standards such as SGML [Bry88], HyTime, ODA, etc., have been developed. With regard to (2), DBMSs are commercially available. OODBMSs allow management of documents with complex structure [BAN94], such that (1) and (2) are addressed simultaneously.

When considering requirement (3), there is an important difference from the other requirements. Standardization, which is a major issue with the first two cases, is not possible in this particular context. Namely, a formal definition of the semantic interpretation of document content cannot be given. Rather, there is a variety of approaches with regard to automatic interpretation of documents, not just one correct way of extracting information extraction from text. If other media types, such as images, video, or audio, come into the fray, the situation is aggravated. In consequence, when building an integrated system, the architecture must be flexible enough to support eventual replacement of the retrieval component. Combining the three basic requirements leads to a further important property.

(4) Integration of the features mentioned so far. While all of requirements (1), (2), and (3) should be met, this should not give rise to unnatural restrictions on a logical level. Beyond that, efficiency of the implementation should not be traded for full integration on the logical level. The particular semantics of the data model and access operations for more efficient processing must be exploited by the system.

1.3 Handling SGML Documents with OODBMSs

The objective of the HyperStorM ('Hypermedia Document Storage and Modeling') project is to build an object-oriented database application framework for structured document storage [ABH94, BAN94].

Documents' database-internal representation reflects their logical structure. In principle, there is a database object corresponding to each logical document component (e.g., one object corresponding to the title-element,

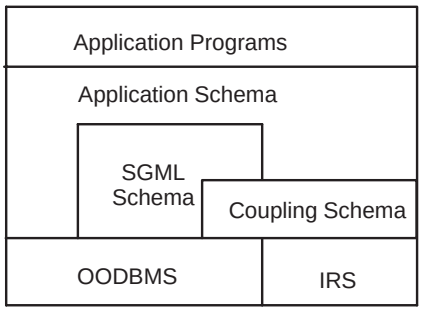

Figure 1: Building Blocks of the Coupled System

one to the abstract, another one to the introduction, etc.). The database objects corresponding to a document's elements make up a hierarchy. Text or raw data of other media types is contained in the leaf objects. For each element type from the DTD, there is a corresponding so-called element-type class. Its instances are the database objects corresponding to the respective elements. With our database application, there is no restriction to a particular set of document types; rather, documents of arbitrary types can be administered without giving up the element-type information. In [ABH94] it is described how to insert a document-type definition and corresponding documents into the database.

2 Our Coupling Approach

We have applied our coupling approach to the SGML framework described in the previous section. The integration of an IR-component with the OODBMS requires careful design of the interface between the IR component and the OODBMS. On the one hand, it must not restrict the generality of the approach, on the other hand, it must allow for efficient implementation.

Our approach allows the creation of arbitrary IRS collections, e.g., a collection which consists of paragraphs, whereas another collection consists of sections. Object-oriented mechanisms are used to obtain a database object's textual representation and relevance values.

Combined structure- and content-oriented queries are done within the OODBMS query language.

2.1 General Design Decisions

We have chosen to focus on a *loose coupling* between the OODBMS and the IRS for the following reasons: a) it allows for a greater flexibility with regard to the IRS paradigm used; and b) it can be implemented with less effort. The advantages of a tight coupling are a) increased performance, b) higher degree of concurrency, and c) easier handling of updates.

Another important decision has been to let the OODBMS be the control component. The application programmer has access only to the OODBMS. Access to the IRS is indirect via the OODBMS. The advantages are: a) full OODBMS functionality, e.g., with respect to the query language, does not have to be reimplemented; and b) kernel manipulations in both the OODBMS and IRS are avoided.

The overall architecture is shown in Figure 1. The application-, SGML-, and coupling schema are OODBMS schemata. An OODBMS schema consists of class definitions. Information about the SGML schema can be found in [ABH94]. It defines the framework for storing SGML documents. The application schema models the general environment for documents, e.g., document containers and application semantics. A short overview of the coupling schema is given in the next section.

2.2 The Coupling Schema

The coupling schema essentially provides two classes: IRSObject and COLLECTION. Class IRSObject is used as a base class for each application specific class whose instances may be subject to content-oriented queries (on their textual representation). Instances of the database class COLLECTION correspond to one IRS collection. This is similar to [HaW92]. The number of IRS collections in use is arbitrary. Two basic methods of COLLECTION are:

- indexObjects(specQuery: DBQuery) creates an IRS index using the text of the database objects which are specified through database query specQuery.
- findIRSValue(IRSQuery: STRING, obj: IRSObject) returns the IRS value for the parameter object with regard to the parameter query.

Each document element type is a subclass of the database class IRSObject. Basic methods of IRSObject are:

- getText() returns an object's textual representation. It is used by indexObjects; its implementation is application-specific and must be provided by the application programmer.
- getIRSValue(c: COLLECTION, IRSQuery: STRING) returns the relevance value of the target object for a given IRS context (a document collection) and an IRS query.

2.3 Creation and Usage of IRS collections

As we have described, SGML documents are stored in the OODBMS. On a logical level, each SGML element is represented by one database object. To incorporate IRS functionality, IRS collections have to be created from the OODBMS data. In other words, a mapping from database objects to IRS documents is needed.

One purpose of the mapping is the assignment of relevance values (returned by the IRS as a result of an IRS query) to database objects. An efficient strategy is to assign exactly one IRS document to each database object. Most IRSs, including INQUERY, allow the storage of attributes along with the IRS document. We use that feature to store the object identifier (OID) of a database object directly with the IRS document. Then it is easy to retrieve the OIDs from the IRS documents.

The second purpose of the mapping relates to the creation of the IRS collection. Within an object oriented environment, it is easy to provide each database object with the functionality to create 'its' IRS document. The open question is: Which parts of possibly large documents should become IRS documents? Some answers are (see [VAB96]):

- Each SGML document becomes an IRS document. The disadvantage is that granularity is coarse with potentially big documents. No information can be obtained about the relevance of document elements, e.g., chapters or paragraphs.
- Each document element of a specified element type becomes an IRS document. This approach is used in most coupling approaches, e.g., [CST92, GTZ93].
- Each leaf node of a document becomes an IRS document (finest granularity).
- One might want to have IRS documents of approximately the same size [Cal94].
- In some cases, one might want to support the processing of certain query types. For instance, this might be accomplished by choosing a fine-grained granularity for documents or document components written by authors which are referenced frequently.

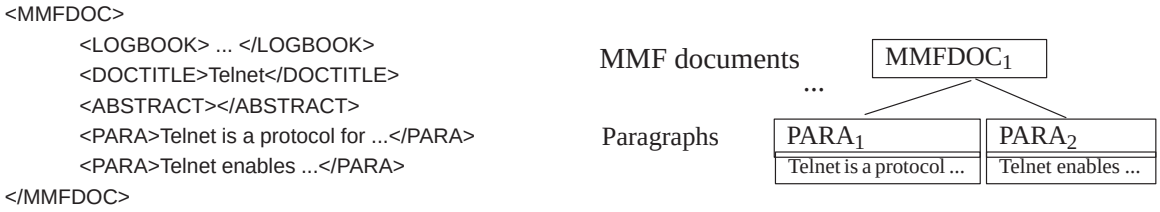

Figure 2: Sample SGML Document and its Internal Representation

- An objective may be to keep update mechanisms simple.

If one wants to furnish each document element with IR functionality, problems occur with hierarchically structured text. Consider the fragment in Figure 2 of an MMF document. On the right hand side the logical structure of this fragment is depicted, which essentially is the database-internal representation: If indexing is at the document-level (the complete text of each document becomes an individual IRS document), content-based queries referring to individual paragraphs cannot be evaluated. This can be avoided by additionally inserting the textual representation of each paragraph into the IRS collection. Then, however, the same text is stored at least twice within the IRS.

A general solution for the mapping problem cannot be given. However, the application programmer can be furnished with a flexible mechanism to define the granularity of IRS documents. The mechanism consists of two steps: identifying the database objects (IRSObject instances) which should be represented within an IRS collection, and specifying the textual representation of each selected database object.

Identifying the database objects is done through an arbitrary database query (the *specification query*), which is a parameter of the COLLECTION instance method indexObjects. COLLECTION instances are created by the application. They are used as retrieval context during query processing. It heavily depends on the application semantics which objects are selected to make up a collection - some criteria were given above.

The textual representation of each database object is identified by the getText method of IRSObject, which has to be implemented by the application programmer. getText is invoked for each object identified by the specification query. The results are stored within a file, which is indexed by the IRS, making up the IRS collection. The application programmer is free to decide which text is returned by getText: it could be the complete text stored in properties of the IRSObject instance, parts of that text, manipulated text, or even text which is derived from related objects. To provide more flexibility, the getText method is parameterized, so that the textual representations can depend on a given parameter, and different representations can be defined for different collections.

The specification query and the implementation of the getText methods determine the degree of text redundancy within the IRS collections. If the application programmer decides to create two collections, and the textual representations overlap, there will be redundancy in the IRS index. To avoid redundancy our approach supports the derivation of relevance values from known values, so that a collection at the finest level (normally the leaf objects within an object hierarchy) will be sufficient. This is described in full detail in [VAB96].

The result of the indexObjects() method call is an IRS collection. With each IRS document the object's OID is stored. On the right hand side of Figure 3 the OODBMS content is shown (shadowed rectangle). It consists of database classes (ellipses), including the coupling classes IRSObject and COLLECTION. Domain specific classes are Chapter and Paragraph, which are derived from IRSObject. Database instances are depicted as bullets. The result of the specification query of one COLLECTION instance is represented by the shadowed area, which includes three Chapter instances and one Paragraph instance. The textual representations of those four objects are stored in the IRS collection on the left hand side of the figure.

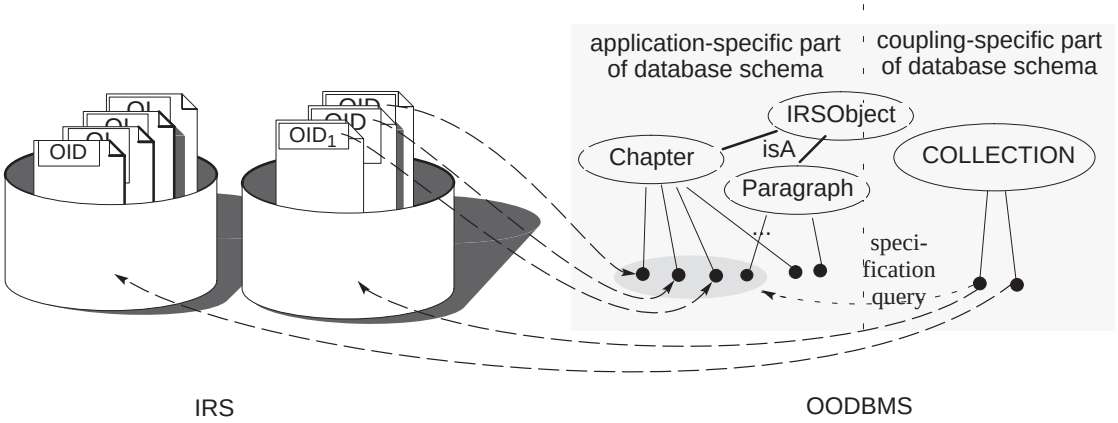

Figure 3: Modeling/Architecture Juxtaposition

2.4 Examples of Mixed Queries

Mixed queries combine structural and content oriented access. The framework is given by the OODBMS query language. VODAK supports OQL (Object Query Language), which is an object oriented extension of SQL. The IRS query part is formulated as arguments of the IRSObject method getIRSValue. The examples are based on MMF documents. collPara is a collection of MMF paragraphs.

"Select all paragraphs and their lengths having an IRS value greater than 0.6 for IRS query 'WWW' ":

```
select p, p -> length() from p in PARA
where p -> getIRSValue (collPara, 'WWW') > 0.6;
```

"Select the title of each MMF document created in 1994 that contains a paragraph element relevant to 'WWW', immediately followed by one relevant to 'NII' ":

```
select d -> getAttributeValue ('TITLE') from d in MMFDOC, p1 in PARA, p2
in PARA
where d -> getAttributeValue ('YEAR') = '1994' and
p1 -> getNext() = p2 and
p1 -> getContaining ('MMFDOC') = d and
p1 -> getIRSValue (collPara, 'WWW') > 0.4 and
p2 -> getIRSValue (collPara, 'NII') > 0.4;
```

3 Application Issues

First we indicate how the coupling approach can be applied to non-textual data. Improved query expressiveness and the application to the WWW are issues we are currently dealing with.

3.1 Applying the Coupling Approach to Non-Textual Media Types

Though we have focused on the problems of structured text documents, our coupling is not limited to those. A practical approach to facilitate information retrieval from images or other multimedia data in documents, for instance, is to have as IRS documents the text fragments that reference the image [CrT91], [DuR93]. The method getText for image objects would return exactly this text.

To give another example, consider an audio segment which can be mapped to text by a speech recognition program. This text can be used for retrieval. The getText method derives this text during the indexing process.

It is even possible to use the same approach for coupling retrieval systems for non-textual data. For example, for image data, an IRS that can build image indexes can be used. The database system passes an image (instead of a text) to the retrieval system, which returns a set of matching images.

3.2 A New Kind of Query Formulation and Evaluation?

The coupled system (ideally) allows the user to retrieve the relevance value of each object of his/her modeling domain (as assumed by the IRS), and to use an arbitrary way of combining relevance values with each other and with structured data. Particularly if hierarchical structures are modeled, this opens opportunities which go beyond those of current IRSs.

Consider a user who is searching for a description of how to use FTP with a WWW browser. With a standard IRS, the user has to submit terms like "WWW" and "FTP" as a query. The result is a list of IRS documents. Matching documents might include 30 page documents which describe network problems (one chapter about WWW, another about FTP) and which do not match the information need of the user. The following query is more likely to select relevant documents; it can be formulated with our system: Return paragraphs relevant to "FTP", which belong to a chapter (or another container object) relevant to "WWW". Although the user can formulate such queries with our coupling approach, a formal model for this kind of query is an open issue. In standard IRSs the combination of terms (e.g., boolean or probabilistic) is possible and is logically evaluated against single documents. In contrast, our approach allows one to combine different query terms with structural features. A query can be given as a pattern for a relevance graph that represents a document structure where the nodes are related with individual IR queries. The query result is not only a single relevance value for each retrieved document (like in IRSs), but a relevance graph. Those IR queries are evaluated against whole document structures or substructures. Similarity matches between query graphs and document graphs is an interesting open issue.

Our future work is to formalize the queries on hierarchically structured documents, and to define or evaluate an appropriate semantics.

3.3 The World Wide Web

Searching for information in the WWW is a difficult task. Numerous search engines try to support it. Most of those search engines are based on simple pattern matching algorithms and do not achieve the functionality of modern IRSs. Moreover, most search engines have a very restricted query interface: complex queries concerning the content and the structure of Web pages cannot be formulated.

WWW server data usually consists of HTML files, stored in the file system. Unfortunately, HTML tags do not have appropriate semantics. In many cases, the tags are used for layout reasons, and do not carry any particular semantics. This severely restricts the benefit of using structural features when searching documents. On the other hand, we see that database solutions become available on the WWW. HTML documents are generated from an internal document representation (e.g., SGML) on-the-fly, providing consistent document structures for large document collections.

Using an expressive query facility might help in the (well-known) case of re-finding information. Many people know *what* HTML document they are searching for, and can often remember some subtle layout details

(e.g., “there was a list of at least five bullets, and the second item was bold”). Here the structural search can enhance content-oriented search in a useful manner. That kind of application is independent from the consistency of HTML tags.

In contrast to the well-known centralized search engines (e.g., Lycos, Alta Vista), we plan to design and implement a non-centralized search engine, where each WWW server handles its own index. We see the following benefits from a non-centralized search engine: a) the documents of all participating WWW servers are covered; b) the index is always up to date; and c) distributed query processing is enabled. An example of an established information system using non-centralized indexes is Hyper-G ([AKM94]). Nevertheless, there remain important open issues with non-centralized search engines, e.g., the query must be distributed to several WWW servers. Index buffering might be necessary if queries are against a large number of WWW servers. Those indexes might be installed on Internet hosts. A drawback of index buffering is increasing index update costs.

Important parts of such a non-centralized search engine are realized by our SGML database. A prototype exists, connected to the WWW, including a WWW search interface. We are investigating mechanisms for distributed querying and index replication which might improve performance.

4 Conclusions

In this article we have described our approach to realize a system providing the integrated functionality of an OODBMS and an IRS. In the introduction, we have argued that there should be some flexibility with regard to the retrieval paradigm, and this is best met with a loose coupling. Likewise, instead of the OODBMS VODAK that we have used for the implementation, another OODBMS having the characteristics described in [Atk+89] can be used. We have pointed out the problems that arise when such a coupling is realized. Further, we believe that our approach is rather flexible, for the following reasons: (1) Arbitrary database objects making up an IRS collection can be chosen. (2) The text forming a database object’s textual representation can be chosen freely. (3) The way a database object’s relevance value is derived from other objects’ relevance values can be defined freely.

The following issues remain for future investigation: Relevance feedback has not yet been examined. Uncertainty is not yet adequately considered within the database component of the coupling. Beyond that, we are interested in finding appropriate formulae for calculating an object’s relevance value from its subobjects’ relevance values. We believe the formula depends on the underlying retrieval paradigm.

Finally, the issue of query optimization in this particular context has to be investigated. As the information-retrieval component is quite fast, in principle it seems worthwhile to generate query-evaluation strategies where information-retrieval subqueries are evaluated first. However, rules for the query optimizer and an appropriate cost model have not yet been specified.

References

[ABH94] K. Aberer, K. Böhm, C. Hüser (1994): “The Prospects of Publishing Using Advanced Database Concepts”, Proceedings of Conference on Electronic Publishing, April 1994, C. Hüser, W. Möhr, and V. Quint, eds., pp. 469-480, John Wiley & Sons, Ltd., 1994.

[AKM94] Keith Andrews, Frank Kappe, Hermann Maurer (1994): “The Hyper-G Network Information System”, Information Processing and Management, Special issue: Selected Proceedings of the Workshop on Distributed Multimedia Systems, Graz, Austria, Nov. 1994.

[Atk+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik (1989): “The Object-Oriented Database System Manifesto”, Proceedings DOOD, Kyoto, December 1989.

[BAN94] K. Böhm, K. Aberer, E.J. Neuhold (1994): “Administering Structured Documents in Digital Libraries”, Advances in Digital Libraries, N.R. Adam, B. Bhargava, and Y. Yesha, eds., Springer, Lecture Notes in Computer Science, 1995.

[Bry88] M. Bryan (1988): “SGML: An Author’s Guide to the Standard Generalized Markup Language”, Addison-Wesley.

[Cal94] J.P. Callan (1994): “Passage-Level Evidence in Document Retrieval”, SIGIR ’94.

[CrT91] W.B. Croft, H.R. Turtle (1991): “Retrieval of complex Objects”, Proceedings of EDBT 92 (S. 217-229), Berlin, Springer-Verlag.

[CST92] W.B. Croft, L.A. Smith, H.R. Turtle (1992): “A Loosely-Coupled Integration of a Text Retrieval System and an Object-Oriented Database System”, SIGIR ’92.

[DuR93] M.D. Dunlop, C.J. van Rijsbergen (1993): “Hypermedia and Free Text Retrieval”, Information Processing & Management, Vol. 29, No. 3, pp. 287-298, 1993.

[GTZ93] J. Gu, U. Thiel, J. Zhao (1993): “Efficient Retrieval Of Complex Objects: Query Processing In a Hybrid DB and IR System”

[HaW92] David J. Harper, Andrew D.M. Walker: “ECLAIR: an Extensible Class Library for Information Retrieval”, The Computer Journal, Vol. 35, No. 3, 1992.

[HeP93] M.A. Hearst, C. Plaunt (1993): “Subtopic Structuring for Full-Length Document Access”, SIGIR ’93.

[SAB93] G. Salton, J. Allan, C. Buckley (1993): “Approaches to Passage Retrieval in Full Text Information Systems”, SIGIR ’93.

[Sue+94] K. Süllow et al. (1994): “MultiMedia Forum—An Interactive Online Journal”, Proceedings of Conference on Electronic Publishing, pp. 413-422, John Wiley & Sons, Ltd.

[VAB96] Marc Volz, Karl Aberer, Klemens Böhm (1996): “Applying a Flexible OODBMS-IRS-Coupling to Structured Document Handling”, ICDE ’96.

The System Architecture and the Transaction Concept of the SPIDER Information Retrieval System

Daniel Knaus, Peter Schäuble
{knaus|schauble}@inf.ethz.ch
Swiss Federal Institute of Technology (ETH)
CH-8092 Zürich, Switzerland

Abstract

A relational database (DB) management system is extended by the SPIDER Information Retrieval (IR) system to provide IR operations on data stored in the DB system. The IR data stored in the IR system is obtained by analyzing the objects in the database. This derived data is needed for advanced IR operations such as relevance ranking and relevance feedback. It is kept consistent with the DB data by triggers and by a synchronization process. Our architecture is based on a loosely-coupled integration which is scalable to a large extent. We developed a new, more powerful transaction model for the IR system where serializability of schedules can be relaxed because the transactions have special properties. Furthermore, the ability to abort transactions is not required in the case of updates of the IR index. We present SPIDER's transaction manager which takes advantage of these properties. We also describe a query optimization strategy to achieve an optimal trade-off between retrieval effectiveness and update efficiency.

1 Introduction

The usefulness of an Information Retrieval (IR) system is bounded by the usefulness of the data collection to which it is providing access [Rij79, pp.145]. The usefulness of a data collection is often related to the number of transactions that are performed on the collection. The more transactions performed, the more the data collection reflects the latest facts, which are usually considerably more valuable than old facts. As a consequence, frequently changing data stored in database (DB) systems is often more valuable than the (almost) static data stored in IR systems.

The integration of IR systems and DB systems is therefore of great practical interest. There are, however, hard research problems to be solved. These problems can be categorized into five classes: architectural problems, problems related to concurrency control and transaction management, problems related to retrieval models and query languages, performance problems, and access control.

Architectural problems are reflected by the numerous attempts to build an IR system on top of a DB system that failed because of unsatisfactory retrieval and update efficiency resulting from this approach [Sme90, CST92, HW92]. Efficient weighted retrieval requires special data structures to have fast access to a large amount of small

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

pieces of information. Such access structures are not very well supported by DB systems because they are designed for general-purpose use. Recently, a first attempt to provide access structures dedicated to IR in a relational DB system has been proposed in [DDS+95].

DB systems provide the so-called ACID properties which ensure the quality of the data to a certain extent [GR93]. To improve the performance, *transactions* are executed in parallel such that the ACID properties are always guaranteed. IR methods are probabilistic in their nature, and as a consequence, it is acceptable to not fulfill completely the ACID requirements in favour of efficiency improvements. Relaxing the ACID properties is also investigated in the areas of distributed systems [Gar83], and advanced transaction models for long-lived activities [Elm92].

Data models are usually based on Boolean algebras [CH80] whereas *IR models* are usually based on probability spaces or vector spaces which do not satisfy the axioms of a Boolean algebra. Attempts to integrate data models and retrieval models are described in [BGP90] and [Fuh90]. In addition to this basic modelling problem, there is the problem of how to take into account the DB schema when evaluating IR queries. The DB schema may contain information that is useful to distinguish between objects that are relevant and objects that are irrelevant to the user. IR methods taking into account the DB schema are proposed in [Fuh92] and [Sch93].

Various approaches have been proposed to accelerate the evaluation of IR queries [BL85, Per94, KMSS96]. The more recent methods achieve a high *performance* by computing approximate results rather than exact results. As mentioned, in probabilistic IR, slightly inexact results are acceptable whereas in most DB applications exact results are required.

The *access control* of a DB system ensures that users see only that subcollection they are allowed to see. They must not learn about the existence of objects outside of this subcollection. Current IR models assume one global data collection that is accessible by everybody. Incorporating access control into an IR system is rather difficult because the ranking order may reveal information about the existence of an object that the user is not allowed to see.

In this paper, we present the new system architecture and the new transaction model of the SPIDER retrieval system [Sch93]. We use a loosely-coupled integration of the SPIDER IR server with a relational DB system to provide advanced IR operations such as relevance ranking and relevance feedback on data stored in the DB system. The IR server is highly optimized to this tasks in terms of retrieval effectiveness and retrieval efficiency which is rather difficult to achieve when building an IR system on top of a DB system. Our approach is not aligned to a specific DB system. An already existing DB environment (consisting of a DB server and some applications) can be extended by IR functions without affecting the existing DB applications. Furthermore, our architecture is highly scalable and hence, suitable for very large, distributed data collections. The performance of a single SPIDER IR server is further improved by a new transaction model where serializability of schedules is relaxed and the ability to abort update transactions is not required. By relaxed serializability, IR query evaluations are executed immediately using all of the data stored so far in the IR server even if there are uncompleted update transactions. We show how updates in the IR server have to be arranged such that query evaluation transactions yield reasonably ranked lists from partially inserted (or partially deleted) objects. In addition, the transaction manager of the SPIDER IR server optimizes the schedules in the SPIDER retrieval system with respect to retrieval effectiveness and with respect to update efficiency. In this paper we do not address recovery from system or disk crashes.

The paper is organized as follows. In Section 2, we describe the loosely-coupled integration of the SPIDER IR system with a relational DB system. In Section 3, we define what are the transaction on the IR index and in Section 4, we present the transaction concept of the SPIDER IR server together with the corresponding optimizations. Finally, we summarize the paper in Section 5.

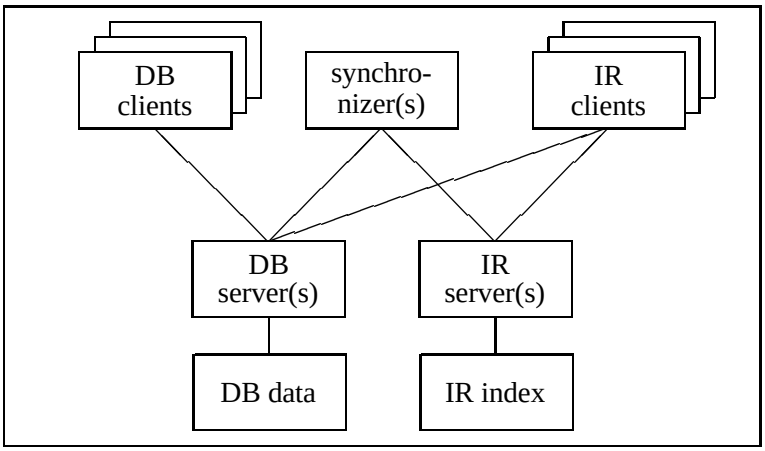

Figure 1: SPIDER's System Architecture.

2 System Architecture

In the SPIDER approach, there exists a DB server and an IR server that operate completely independent of each other in the case of read operations (see Figure 1). In the case of modifications, the DB data and the IR data (i.e., the IR index) are synchronized by a process called synchronizer.

The *DB server* manages a set of relations in the usual way. We assume that the DB administrator defined a set of retrievable items which we call *documents*. The current set of documents is denoted by $D = \{d_0, ..., d_{n-1}\}$. A document d_j may be a tuple of a relation or of a view. In either case, a document d_j is identified by a unique identifier *ID*(d_j) consisting of the primary key of the tuple and the name of the relation (or the view) to which the tuple belongs.

The *IR server* manages an access structure called IR index. The *IR index* can be considered as the precomputed results of primitive queries. A general query is evaluated efficiently by combining the results of the primitive queries that constitute the general query.

The *IR clients* accept queries q which are indexed to obtain the corresponding query descriptions $\vec{q}$. Indexing a query (as well as indexing a document d_j) is usually a fully automatic process where statistical information about the distribution of the indexing features (e.g., stemmed words) is determined. For details about indexing see for instance [SM83]. The result of an indexing process consists of a set of weighted features which is represented by a feature vector ($\vec{q}$ or $\vec{d_j}$) where every indexing feature is assigned a dimension. After indexing the query, the IR client sends the corresponding query description $\vec{q}$ to the IR server which computes so-called Retrieval Status Values $RSV(q, d_j) = \rho(\vec{q}, \vec{d_j})$ where $\vec{q}$ and $\vec{d_j}$ denote the query and document descriptions and ρ denotes a retrieval function which matches the query and document descriptions. The IR server returns a ranked list of documents d_j that are ordered in decreasing order of their retrieval status values. Such lists may also contain subsumptions of the documents (e.g., titles) that can be presented to the user immediately without accessing the DB server. While viewing the ranked list a user can inspect documents by loading them from the DB server.

The IR index is synchronized with the DB data by triggers. Because in most of today's DB systems, triggers cannot call external procedures (for good reasons, e.g., security reasons), we use an internal *update table* to keep track of the DB updates. The triggers insert a tuple into the update table whenever a document has been inserted, deleted, or modified. The update table as well as the triggers have to be added to the database at the outset when the DB server is extended by the SPIDER retrieval system. The tuples in the update table consist of document identifiers *ID*(d_j) and information about the kind of the update of d_j. The *synchronizer* periodically reads the update table. If there are new entries in the update table, the synchronizer performs the corresponding actions as shown in Figure 2.

```
loop
  check update table for a new entry u_k
  if new entry u_k (corresponding to a document d_j) detected then
    case kind of update(u_k) was
    insertion:      load new document d_j from DB server
                    index document d_j to obtain description d_j⃗
                    insert (ID(d_j), d_j⃗, title(d_j)) into IR server
    deletion:       delete all information of the document with ID(d_j) from IR server
    modification:   load modified document d_j from DB server
                    index new d_j to obtain description d_j⃗
                    replace d_j⃗ and title(d_j) of the document with ID(d_j) in the IR server
    end case
    delete entry u_k from the update table
  else sleep t seconds
  end if
end loop
```

Figure 2: Pseudo-program of the Synchronizer.

It is possible that loading a document d_j by the synchronizer fails because it has already been deleted after its insertion or modification. In such a case, the update table must also contain an entry u'_k corresponding to the deletion of d_j because the execution of triggers is part of the DB transactions [SQL95]. If the synchronizer encounters such a situation, it immediately continues with u'_k (i.e., the deletion of d_j) and upon completing u'_k it deletes both u_k and u'_k from the update table. We omitted these cases in the pseudo-program for easier readability.

This kind of synchronization introduces a delay between the update of a document in the database and the time when it is retrievable through the IR server. The delay depends on how frequently the synchronizer is polling the update table.

The advantages of the loosely-coupled integration of the SPIDER IR system with a DB system are summarized as follows. (a) It is possible to extend an already existing (operational) DB environment consisting of many DB applications and possibly several DB servers by IR functions (best-match retrieval, document analysis, support for interactive searches) without affecting the existing applications. (b) The IR system may incorporate more sophisticated query evaluation techniques (e.g., similarity thesauri, automatic query expansion, etc.) and useful document inspection tools like highlighting words of the query or passage retrieval [MS94] which are difficult to build into a DB server. (c) The throughput of query evaluations and updates within a single IR server is optimal because the IR server's work is reduced to a minimum: indexing documents and IR queries is performed outside the IR server by the synchronizer (if necessary several synchronizers) and by the IR clients. (d) The proposed system architecture is scalable to a large extent to improve the performance.

Some issues are still open for discussion: what is the architecture of the integration when more than one IR server is involved? The IR index could be replicated or split in different ways [TG93]. Such configurations improve the throughput and response times of IR query evaluations (and updates) but it is still unclear how the partial results of different IR servers have to be merged together [VGJ95] and how the IR servers are coordinated [CM95].

3 Transactions on the IR Index

In the IR server, two types of activities occur: either an IR query has to be evaluated or the IR index has to be updated according to changes in the DB server(s). In this section we define how these activities are embedded in

the transaction concept of the overall system.

An IR search is usually an iterative process of evaluating IR queries and inspecting the documents of the ranked list. After some document inspections the IR query may be reformulated or extended by relevance information and then evaluated again and so on. Regarding this process as a single transaction it can be long (it may even be continued over several days) with many documents involved, in particular, the documents used for relevance feedback. Moreover the users may not want to explicitly state the end of their searches. Thus, it is impractical to lock all involved documents for the period of the IR search. We, therefore, commit IR searches after each query evaluation or document inspection. The price of this relaxation is the following. First, it is possible that a document appears in a ranked list but when the user tries to access the document, it cannot be loaded anymore because it has been deleted meanwhile. A similar situation is modification of a document after query evaluation, but the content of the document will probably not be much different and it may still be relevant to the user's query. Second, documents marked for relevance feedback may also be deleted or modified before the query with the feedback information is evaluated. Depending on the application, these situations can be handled in different ways, e.g., by simply displaying a message to the user. Below we will refer to the evaluation of a single query in the IR server as an *IR evaluation transaction*.

The result of an IR evaluation transaction is a ranked list consisting of document Id's, Retrieval Status Values (RSVs) and possibly document subsumptions (e.g., titles). The RSVs are estimates of probabilities that the documents are relevant to the IR query. Since estimates naturally introduce errors (namely the deviations between the estimates and the unknown probabilities) we can also accept errors by the transaction manager as long as they are within certain limits. Our transaction manager allows the computation of RSVs from partially inserted document descriptions. For most retrieval methods, the resulting (partial) RSVs are always smaller than or equal to the correct RSVs derived from the complete document descriptions. If a partial RSV is much smaller than the full RSV it looks as if the document is not available yet. On the other hand, deleted documents, whose descriptions are not yet removed completely, are eliminated from the ranked lists before finishing the query evaluations. Hence, partially deleted documents never appear in a ranked list of documents.

Inserting, deleting, or modifying a document entails updating the DB data (*DB update transaction*) as well as updating the IR index (*IR update transaction*). Because indexing a document usually needs all of its content, it has to take place entirely after the DB update transaction but of course before the IR update transaction. Hence, the DB update transaction has to be executed completely before the IR update transaction.

Once the DB update transaction is committed, there is no reason to allow a user to abort the corresponding IR update transaction. Remember that IR update transactions are controlled by the synchronizer process and not by users. As we will show in Section 4.2, deadlocks, and integrity violations cannot cause aborts because they are avoided completely by our transaction manager. Aborting IR evaluation transactions is no problem because query evaluations are read-only transactions, and hence, they need no undo actions.

4 Transaction Manager of the SPIDER IR Server

The transaction manager of the SPIDER IR server is based on a specific access structure of the IR index. Therefore we first describe SPIDER's access structure which supports several of the most effective retrieval methods and a fast query evaluation algorithm.

4.1 Retrieval Model — Access Structure

The access structure of the IR index depends on the retrieval method (what IR data is to be stored) and on the retrieval algorithm (how is the IR data accessed). The primary goal of an IR search is a high retrieval effectiveness. We compared several retrieval methods that performed very well at the latest TRECs (Text REtrieval Conferences) [Har96, Har95, Har94]. At TRECs, different IR research groups run their retrieval methods against a large

collection of textual data. Looking at some successful ad-hoc methods [CCB94, CCG94, RWJ+95, BSM96], the following IR data that can be precomputed (and stored) has been used:

n	number of documents
$\mathit{ff}(\varphi_i, d_j)$	feature frequency of feature φ_i within the document d_j (number of occurrences of φ_i within d_j)
$\mathit{df}(\varphi_i)$	document frequency of feature φ_i (number of documents containing φ_i)
$\mathit{len}(d_j)$	length of document d_j (number of features or words, or the length of the description vector)

Fast retrieval algorithms usually work on *inverted posting lists*, i.e., the $(\mathit{ID}(d_j), \varphi_i, \mathit{ff}(\varphi_i, d_j))$-tuples are stored by feature, not by document Id. Then the retrieval algorithm accesses only postings of query features. A further acceleration is achieved by sorting the postings of a feature by descending feature frequencies. Because a user is usually interested in the beginning (top) of the ranked list only, the RSVs of documents ranked very low do not have to be computed. The usual approach is to process high feature frequency postings first, and to abort the processing when some condition becomes true [Per94, BL85, KMSS96]. From these requirements we derived a model of an access structure and the necessary operations on it.

IR index	$\longmapsto$ Globals, Documents $\{..., \mathit{ID}(d_j), ...\}$, Features $\{..., \varphi_i, ...\}$
Globals	$\longmapsto n$, other global values
$\mathit{ID}(d_j)$	$\longmapsto$ non-inverted postings $\{..., (\varphi_i, \mathit{ff}(\varphi_i, d_j)), ...\}$, $\mathit{len}(d_j)$, other per-document information
φ_i	$\longmapsto$ inverted postings $\{..., (\mathit{ID}(d_j), \mathit{ff}(\varphi_i, d_j)), ...\}$, $\mathit{df}(\varphi_i)$, other per-feature information

The curly braces denote a set of elements. In the case of the inverted lists, the postings are sorted firstly by feature frequencies and secondly by document Id's. We store non-inverted postings to guarantee correct deletion of all inverted postings of a document and to support relevance feedback [Roc71] and a similarity thesaurus [SK92].

4.2 Transaction Concept Based on Steps

We adopt the notion of *semantic types* and *steps* defined in [Gar83]. Transactions are classified into a set of semantic types. Every transaction is divided into subsequences of actions called steps.

In our case we have the four semantic types T^{ins} (insertion of a new document), T^{del} (deletion of a document), T^{mod} (modification of an existing document), and T^{eval} (query evaluation). In contrast to Garcia-Molina, we require that steps of a single IR update transaction (of type T^{ins}, T^{del}, or T^{mod}) and steps of different IR update transactions always commute. Commutativity is achieved by encapsulating the adjustment of global values and per-feature information into steps. In this way, a *stepwise serial* schedule facilitates interleaving IR update transactions without using costly locking mechanisms. A stepwise serial schedule represents an execution in which all steps are performed as indivisible units. Most importantly, deadlocks, and integrity violations (lost updates, dirty reads, and unrepeatable reads) are avoided completely because of the commutativity of the steps.

In detail the steps of IR update transactions are defined in the following. The steps *B* are performed for every feature of a document.

T^{ins}	step *A*:	insert per-document information ($\vec{d_j}$, $\mathit{len}(d_j)$, ...) and adjust global values (n, ...)
	steps B_{ins}:	insert the posting $(\mathit{ID}(d_j), \mathit{ff}(\varphi_i, d_j)))$ into the inverted list of φ_i and adjust per-feature information ($\mathit{df}(\varphi_i)$, ...)
T^{del}	step *A*:	delete per-document information ($\vec{d_j}$, $\mathit{len}(d_j)$, ...) and adjust global values (n, ...)
	steps B_{del}:	delete posting $(\mathit{ID}(d_j), \mathit{ff}(\varphi_i, d_j)))$ from the inverted list of φ_i and

adjust per-feature information (*df*(φ_i), ...)

T^{mod} step *A*: replace per-document information ($\vec{d_j}$, *len*(d_j), ...) and
adjust global values (n, ...)
steps B_{del}: delete old posting (*ID*(d_j), *ff*(φ_i, d_j))) from the inverted list of φ_i and
adjust per-feature information (*df*(φ_i), ...)
steps B_{ins}: insert new posting (*ID*(d_j), *ff*(φ_i, d_j))) into the inverted list of φ_i and
adjust per-feature information (*df*(φ_i), ...)

How to divide IR evaluation transactions (of type T^{eval}) into steps depends very much on the retrieval method and on the evaluation algorithm. Additionally, evaluation steps do not commute with steps of IR update transactions in general. However, integrity violations (dirty reads and unrepeatable reads) following from interleaving steps of evaluation transactions and steps of update transactions are admissible because of the probabilistic nature of the RSV computations (see Section 3).

4.3 A Simple Scheduler for IR Transactions

The transaction manager of the SPIDER IR server consists of a serial scheduler for steps. Because we use a fast retrieval algorithm [KMSS96] where inverted posting lists are processed in parallel, we perform query evaluations as single, short steps. Postings that have to be inserted into or deleted from the inverted lists are managed by maintaining a *todo list*. The todo list contains the postings and the corresponding actions (see Figure 3).

```
loop
    check for new transaction requests (non-blocking accept)
    if new transaction T then
        case type(T) is
        T^eval: compute RSVs and return the ranked list (single step)
        T^ins:  perform step A of T^ins
                put all postings of d_j into the todo list for insertion
        T^del:  read (non-inverted) postings of d_j
                mark d_j as deleted such that d_j is not ranked anymore
                perform step A of T^del
                put all postings of d_j into the todo list for deletion
        T^mod: read old postings of d_j
                perform step A of T^mod
                put all old postings of d_j into the todo list for deletion
                put all new postings of d_j into the todo list for insertion
        end case
    end if
    rearrange the todo list as described in Section 4.4
    process some postings of the todo list (steps B)
        depending on the workload and on the length of the todo list
end loop
```

Figure 3: Pseudo-program of the IR Server.

Because the query evaluations can be performed between two steps of update transactions and no locks are maintained, the RSV computations may be based on partial document descriptions only, and hence, the schedules produced by this scheduler are not serializable. By storing per-document information and adjusting global values

before inserting the postings of a document into the inverted lists, it is the case for most retrieval methods that the partial RSVs are not larger than the RSVs computed from the full document descriptions.

Serial scheduling has the advantage that no locking mechanism, and hence, less work overhead is necessary. Moreover, the processing of the todo list can be optimized with respect to the correctness of evaluation transactions and with respect to the efficiency of modifications (see Section 4.4).

4.4 Optimizing Query Evaluations and Updates in the IR Server

The *todo list*, holding postings that have to be inserted into or deleted from the inverted lists, can be optimized having the following objectives in mind:

1. In the case of insertions, those postings that are already inserted into the inverted posting lists should result in the best possible RSV estimations to optimize retrieval effectiveness.

2. In the case of deletions, those postings that are still stored in the inverted posting lists should result in very low RSVs to avoid unnecessary computations (deleted documents are cancelled from the ranked lists).

3. In the case of modifications, those postings that are already replaced in the inverted posting lists should result in the best possible RSV estimations to optimize retrieval effectiveness. Postings that do not change at all should not be deleted and reinserted.

4. Postings of the same feature can be collected and processed in common. By so doing, the inverted list of a feature is updated with much fewer disk accesses than by inserting or deleting each posting individually.

The first three cases can be handled by processing those postings of the todo list first that contribute most to the RSVs. If we do not know exactly the contribution of a posting to an RSV without accessing the document frequency of the feature and maybe even more information, the decision of which posting to process next may be based on the feature frequency only. Thus, high frequency postings would be processed first. Of course, this is not optimal because a high feature frequency does not necessarily mean a high contribution to the RSVs, but it is a reasonable and simple approximation. The RSVs that have been computed based on the inverted lists are approximations of the correct RSVs. These approximations can be further improved by incorporating postings which are in the todo list and belong to documents of the ranked list.

The fourth objective is aimed at grouping those postings in the todo list which are inserted into or deleted from the same inverted list. In this way disk access is minimized. There is a trade-off between long todo lists that support well this kind of grouping versus short todo lists that improve the approximations of the RSVs. We would like to emphasize that the optimization strategy can be adapted *dynamically* to the workload and to the ratio between evaluation and update transactions.

5 Summary

The loose coupling makes it easy to extend any commercial database system with information retrieval functionality as long as the database system provides the concept of triggers. The impact of the extension on an existing database environment is very little except for the synchronizer's polling activities.

The detailed knowledge about the semantic of IR transactions leads to a transaction model where serializability of schedules is relaxed and update transactions need not be abortable. As a result the "Isolation" of transactions in the IR server is relaxed while the other three ACID properties (Atomicity, Consistency, and Durability) are still satisfied.

The steps of update transactions are inserted into the todo list. Query evaluations and IR index updates are optimized by rearranging the entries of the todo list. The optimization strategy can be adapted dynamically depending on the current requirements.

References

[BGP90] Barbara, D., Garcia-Molina, H., and Porter, D. (1990). A Probabilistic Relational Model. In *Advances in Database Technology—EDBT'90*, pp. 60–74, Berlin. Springer-Verlag.

[BL85] Buckley, C., and Lewit, A. F. (1985). Optimization of Inverted Vector Searches. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 97–110.

[BSM96] Buckley, C., Singhal, A., and Mitra, M. (1996). New Retrieval Approaches Using SMART: TREC-4. In *Proceedings of the Forth Text REtrieval Conference (TREC-4)*. National Institute of Standards and Technology (NIST).

[CM95] Cahoon, B., and McKinley, K. (1995). Performance Analysis of Distributed Information Retrieval Architectures. Technical Report CMPSCI 95-54, Department of Computer Science, University of Massachusetts, Amherst, MA.

[CCB94] Callan, J. P., Croft, W. B., and Broglio, J. (1994). TREC-2 Routing and Ad-Hoc Retrieval Evaluation using the INQUERY System. In *Proceedings of the Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, pp. 75–83.

[CH80] Chandra, A., and Harel, D. (1980). Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21(2), pp. 156–178.

[CCG94] Cooper, W. S., Chen, A., and Gey, F. (1994). Full Text Retrieval based on Probabilistic Equations with Coefficients fitted by Logistic Regression. In *Proceedings of the Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, pp. 57–66.

[CST92] Croft, W. B., Smith, L. A., and Turtle, H. (1992). A Loosly-Coupled Integration of a Text Retrieval System and an Object-Oriented Database System. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 223–232.

[DDS+95] DeFazio, S., Daoud, A., Smith, L. A., Srinivasan, J., Croft, W. B., and Callan, J. (1995). Integrating IR and RDBMS Using Cooperative Indexing. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 84–92.

[Elm92] Elmagarmid, A. (1992). *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, San Mateo, CA.

[Fuh90] Fuhr, N. (1990). A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In McLeod, D., Sacks-Davis, R., and Schek, H., editors, *International Conference on Very Large Data Bases*, pp. 696–707, Los Altos, CA. Morgan Kaufman.

[Fuh92] Fuhr, N. (1992). Integration of Probabilistic Fact and Text Retrieval. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 211–222.

[Gar83] Garcia-Molina, H. (1983). Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems*, 8(2),186–213.

[GR93] Gray, J., and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA.

[Har94] Harman, D. (1994). Overview of the Second Text REtrieval Conference (TREC-2). In *Proceedings of TREC-2*, Special Publication 500-215, pp. 1–20.

[Har95] Harman, D. (1995). Overview of the Third Text REtrieval Conference (TREC-3). In *Proceedings of TREC-3*, Special Publication 500-225, pp. 1–19.

[Har96] Harman, D. (1996). Overview of the Forth Text REtrieval Conference (TREC-4). In *Proceedings TREC-4*. National Institute of Standards and Technology (NIST). To appear.

[HW92] Harper, D. J., and Walker, D. M. (1992). ECLAIR: an Extensible Class Library for Information Retrieval. *The Computer Journal*, 35(3), pp. 256–267.

[KMSS96] Knaus, D., Mittendorf, E., Schäuble, P., and Sheridan, P. (1996). Highlighting Relevant Passages for Users of the Interactive SPIDER Retrieval System. In *Proceedings of the Forth Text REtrieval Conference (TREC-4)*. National Institute of Standards and Technology (NIST). To appear.

[MS94] Mittendorf, E., and Schäuble, P. (1994). Document and Passage Retrieval Based on Hidden Markov Models. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 318–327.

[Per94] Persin, M. (1994). Document Filtering for Fast Ranking. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 339–348.

[RWJ+95] Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M., and Gatford, M. (1995). Okapi at TREC-3. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, NIST Special Publication 500-225, pp. 109–126.

[Roc71] Rocchio, J. (1971). Relevance Feedback in Information Retrieval. In Salton, G., editor, *The SMART Retrieval System—Experiments in Automatic Document Processing*, chapter 14. Prentice-Hall Inc.

[SM83] Salton, G., and McGill, M. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill, New York.

[Sch93] Schäuble, P. (1993). SPIDER: A Multiuser Information Retrieval System for Semistructured and Dynamic Data. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 318–327.

[SK92] Schäuble, P., and Knaus, D. (1992). The Various Roles of Information Structures. In *Proceedings of the 16th Annual Conference of the "Gesellschaft für Klassifikation e.V."*, pp. 282-290. Springer-Verlag.

[Sme90] Smeaton, A. F. (1990). retriev: an information retrieval system implemented on top of a relational database. *program, automated library and information systems*, 24, pp. 27–38.

[SQL95] SQL/3 (1995). ISO and ANSI SQL3 standard. Working Draft.

[TG93] Tomasic, A., and Garcia-Molina, H. (1993). Performance of Inverted Indices in Shared-Nothing Distributed Text Document Information Retrieval Systems. In *Parallel and Distributed Information Systems*, pp. 8–17.

[Rij79] van Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworths, London, second edition.

[VGJ95] Voorhees, E., Gupta, N., and Johnson-Laird, B. (1995). The Collection Fusion Problem. In *TREC-3 Proceedings*, pp. 95–104.

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398