12

The Multi-model DBMS Architecture and XML Information Retrieval

Arjen P. de Vries, Johan A. List, and Henk Ernst Blok

12.1 Introduction

Since long, computer science has distinguished between information retrieval and data retrieval, where information retrieval entails the problem of ranking textual documents on their content (with the goal to identify documents relevant for satisfying a user's information need) while data retrieval involves exact match, that is, checking a data collection for presence or absence of (precisely specified) items. But, now that XML has become a standard document model that allows structure and text content to be represented in a combined way, new generations of information retrieval systems are expected to handle *semi-structured* documents instead of plain text, with usage scenarios that require the combination of 'conventional' ranking with other query constraints; based on the structure of text documents, on the information extracted from various media (or various media representations), or through additional information induced during the query process.

Consider for example an XML collection representing a newspaper archive, and the information need 'recent English newspaper articles about Willem-Alexander dating Maxima'.[1] This can be expressed as the following query (syntax in the spirit of the XQuery-Fulltext working draft [58]):[2]

```
FOR $article IN document("collection.xml")//article
WHERE $article/text() about 'Willem-Alexander dating Maxima'
  AND $article[@lang = 'English']
  AND $article[@pdate between '31-1-2003' and '1-3-2003']
RETURN <result>$article</result>
```

The terms 'recent' and 'English' refer to metadata about the newspaper articles, whereas the aboutness-clause refers to the news content. Because only

[1] Willem-Alexander is the Crown Prince of The Netherlands, who married Maxima Zorreguieta on 2-2-2002.

[2] Assume an interpretation in which 'recent' is equivalent to 'published during the last month', and *language* and *pdate* are attributes of the article tag. The *between ... and ...* construct does not exist in XQuery, but is used for simplicity.

H. Blanken et al. (Eds.): Intelligent Search on XML Data, LNCS 2818, pp. 179–191, 2003.

recent English articles will be retrieved by this request, precision at low recall levels is likely to be improved. Note that this capability to process queries that combine content and structure is beneficial in ways beyond extending querying textual content with constraints on rich data types like numeric attributes (e.g., price), geographical information and temporal values. Egnor and Lord [105] suggest that new generations of information retrieval systems could exploit the potentially rich additional information in semi-structured document collections also for disambiguation of words through their tag context, and use structural proximity as part of the ranking model. Also, combined querying on content and structure is a necessary precondition for improving the IR process when taking into account Mizzaro's different notions of relevance (see [221]).

12.1.1 Dilemma

So, we observe a trend in software development where information retrieval and data retrieval techniques are combined; as a result of a desire to create applications that take advantage of the additional information made explicit in semi-structured document collections, and, as importantly, to provide a basis for improved information retrieval models, possibly achieving better recall and precision thanks to exploiting this additional information automatically.

Developments in hardware layout of computer systems pose another challenge – to be met in any resource-consuming application, not just IR. Due to the increasing importance of cache memory in modern CPUs, memory access cannot be thought of as random access any more: sequential access patterns can be many times faster than random accesses [39]. Since memory latency is *not* improving with Moore's law (unlike other properties of computer architecture), this problem will gain importance. At the same time, the price of (cheap) personal computers compared to (expensive) professional workstations stimulates distribution of work over 'workstation farms', taking advantage of shared-nothing parallelism. Similarly, server machines with two up to eight processors are not extremely expensive any more, making it more attractive to explore strategies that exploit shared-memory parallelism. The drawback of this increased complexity in computer architecture (hierarchical memory, parallelism, distribution) is that efficient usage of the available resources requires highly experienced programmers, who are fully aware of the low-level details of the machine architecture used.

As a result of these observed trends (software and hardware), a dilemma arises in the engineering of information retrieval systems: should their design be optimized for flexibility or for efficiency? Highly optimized stand-alone systems are, naturally, not very flexible. Experiments (e.g., with new models or adaptive learning strategies) require changes in complex low-level code, with the danger of affecting the correctness of the results. To circumvent such inflexibility, it is common practice to wrap the core IR system in a *black-box*, and implement additional features on top. Considering that we should

optimize our systems for parallel and distributed computing and memory access patterns however, usage of black-box abstractions to obtain flexibility becomes ever less desirable: it leads easily to inefficient systems, as we do not *really* understand what happens inside the ranking process.

The essence of our problem is being trapped in an impasse solving the dilemma: gaining flexibility through abstraction causes an efficiency penalty which is felt most when we exploit this flexibility in new applications of IR or explore improvements upon existing models.

This problem is illustrated clearly in the processing of relevance feedback. Retrieval systems typically rank the documents with the initial query in a first pass and re-rank with an adapted query in a second pass. Jónsson et al. have shown in [182] that the resulting retrieval system is not optimal with respect to efficiency, unless we address buffer management while taking both passes into account. So, the inner workings of the original system must be changed for optimal performance of the full system. In other words, *we must break open the black-box*. This, obviously, conflicts with our previously stated desire for flexibility.

Another illustration of this dilemma appears when extending retrieval systems for multimedia data collections, strengthening our arguments against the pragmatic engineering practice of coupling otherwise stand-alone retrieval systems. In a multimedia retrieval system that ranks its objects using various representations of content (such as the system described in [302]), the number of independent black-box components that may contribute to the final ranking equals the number of feature spaces used in the system. It seems unlikely that computing these multiple rankings independently (i.e., without taking intermediate results into account) is the most efficient approach.

12.2 A Database Approach to IR

We seek a way out of this impasse between flexibility and efficiency by following 'the database approach'. Database technology provides flexibility by expressing requests in high-level, declarative query languages at the conceptual level, independent from implementation details such as file formats and access structures (thus emphasizing *data independence*). Efficiency is obtained in the mapping process from declarative specification (describing what should happen) into a query plan at the physical level (describing how it happens). The query optimizer generates a number of logically equivalent query plans, and selects a (hopefully) efficient plan using some heuristics.

There is not much consensus on how the integration of IR techniques in general-purpose database management systems (DBMSs) should take place. The typical system design couples two standalone black-box systems using a shallow layer on top: an IR system for the article text and a DBMS for the structured data. Their connection is established by using the same document identifiers in both component systems. State-of-the-art database solu-

tions make available new functions in the query language of object-relational database management systems (OR-DBMSs); these functions interface to the otherwise still stand-alone software systems. Therefore, extending an OR-DBMS with an information retrieval module means no more than again 'wrapping' the IR system as a black-box inside the DBMS architecture (see also Figure 12.1). Apart from seriously handicapping the query optimizer, the IR module must handle parallelism and data distribution by itself. Therefore, adapting an OR-DBMS for the changing requirements identified in Section 12.1 may even be *more* complex than enhancing the stand-alone IR system.

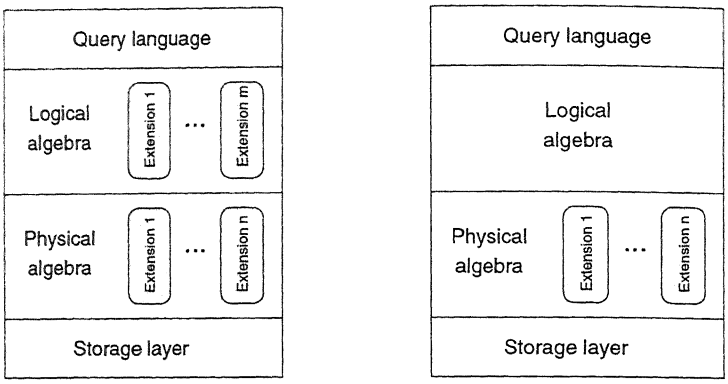

Fig. 12.1. Comparing a Multi-Model DBMS (left) to an Object-Relational DBMS (right).

We propose the *Multi-Model DBMS* architecture as a better alternative. It has been especially designed to enable the integration of databases and (multimedia) information retrieval [90]. As depicted graphically in Figure 12.1, this system can not only be extended at the physical level, but also at the logical level. Its extensibility at multiple layers in the architecture enables a strong notion of data independence between the logical and physical levels. We call this architecture the Multi-Model DBMS, since the data model used at the logical level can be different from that at the physical level.

In a prototype implementation, the Mirror DBMS, we have used the Moa (X)NF^2 algebra [303] at the logical level, and the binary relational algebra provided by MonetDB [37] at the physical level. Knowledge about IR is captured using Moa's extensibility with domain-specific structures. These extensions map high-level operations (such as probabilistic ranking of document content) to binary relational algebra expressions [90].

Creating a clear separation of concerns by introducing a logical and physical layer provides several advantages. First, operations defined at the physical level support facilities for features of the computer system architecture (e.g., shared-memory parallel processing) and can take full advantage of modern

CPU processing power. The mapping of operators by extensions at the logical level onto expressions at the (possibly extended) physical level is the appropriate place for encoding domain-specific knowledge, such as the Zipfian distribution of terms typical for IR (as demonstrated in [32]).

A good example is given by the tree-awareness for relational databases, introduced by Grust and Van Keulen (Chapter 16 of this book). The staircase join proposed is an example of an extension at the physical level, improving the efficiency of structural joins. A logical extension for handling trees based on their work encodes, e.g., the pruning predicates added to the WHERE clause of the SQL query, or the criteria for using the staircase join in the physical query plan.

Another advantage of the separation of concerns in the Multi-Model DBMS architecture is that it allows IR researchers to concentrate on retrieval models and reduce the effort of implementation involved with empirical studies. It separates representation from evidential reasoning and query formulation, which reduces the effort of changing the application logic significantly: the IR researcher makes necessary changes only to the IR processing extension, while the user applications do not require adaptation whenever the retrieval model is changed. This advantage has been called content independence [91], as a counterpart of data independence in database management systems. The notion of content independence helps to keep changes local (changes to the retrieval model) when experimenting with new theory.

12.3 Extensions for XML Information Retrieval

Our prototype system for information retrieval on XML collections exploits the separation of concerns discussed in Section 12.2, by introducing two separate extensions at the logical level: an XML extension for handling path expressions over XML documents and an IR extension for IR primitives. To illustrate a typical retrieval session (including the translation of a logical layer query to a possible physical query execution plan), consider again the example query of Section 12.1 (an example XML document is shown in Figure 12.2):

```
FOR $article IN document("collection.xml")//article
WHERE $article/text() about 'Willem Alexander dating Maxima'
  AND $article[@lang = 'English']
  AND $article[@pdate between '31-1-2003' and '1-3-2003']
RETURN <result>$article</result>
```

This query can be rewritten to include a ranking preference, typical for IR applications:

```
FOR $article IN document("collection.xml")//article
LET $rsv := about($article/text(), 'Willem-Alexander dating Maxima')
WHERE $article[@lang = 'English']
  AND $article[@pdate between '31-1-2003' and '1-3-2003']
ORDER BY $rsv DESCENDING
RETURN <result><rsv>$rsv</rsv>$article</result>
```

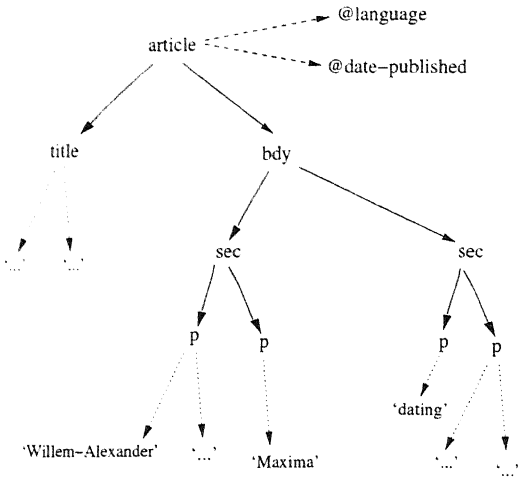

Fig. 12.2. Possible article excerpt of an XML newspaper archive; the leaf nodes contain index terms.

The *about* statement in the query above can be seen as an instantiation of the retrieval model used at the logical layer, taking as input the text of the articles in the collection (i.e., the article text regions) and the query text. An instantiation of the retrieval model requires the collection of component text and computation of term statistics, as well as calculating a score for the component under consideration.

The logical XML extension manages the document structure using a *storage scheme* at the physical level that is based on *text regions* [78, 173]. Text regions support dynamic computation of term statistics in any projection of the XML syntax tree, including transitive closures. They also offer flexibility for determining structural relationships between given sets of nodes, an important property for, e.g, efficient traversal along XPath axes.

We view an XML document instance as a linearized string or a set of *tokens* (including both entity tags and document text tokens), instead of as a syntax tree. Each component is then a *text region* or a contiguous subset of the entire linearized string. The linearized string of the example document in Figure 12.2 is shown below:

```
<article ... ><title>...</title><bdy><sec><p> ... </bdy></article>
```

A text region a can be identified by its starting point s_a and ending point e_a within the entire linearized string, where assignment of starting and ending points is simply done by maintaining a token counter. Figure 12.3 visualizes the start point and end point numbering for the example XML document and we can see, for example, that the *bdy*-region can be identified with the closed interval [5..24].

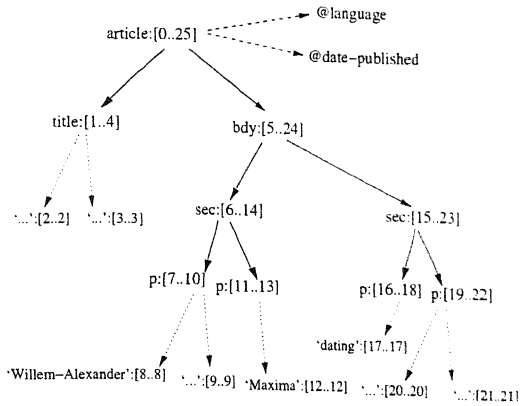

Fig. 12.3. Start point and endpoint assignment

At the physical level, our system stores these XML text regions as four-tuples $(region\_id, start, end, tag)$, where:

- *region_id* denotes a unique node identifier for each region;
- *start* and *end* represent the start and end positions of each region;
- *tag* is the (XML) tag of each region.

The set of all XML region tuples is named the *node index* $\mathcal{N}$. Index terms present in the XML documents are stored in a separate relation called the *word index* $\mathcal{W}$. Index terms are considered text regions as well, but physically the term identifier is re-used as both start and end position to reduce memory usage. Node attributes are stored in the attribute index $\mathcal{A}$ as four-tuples $(attr\_id, region\_id, attr\_name, attr\_val)$. Furthermore, we extended the physical layer with the text region operators, summarized in Table 12.1. Note that we have put the text region operators in a relational context, delivering sets or bags of tuples.

Table 12.1. Region and region set operators, in comprehension syntax [56]; s_r and e_r denote the starting and ending positions of region r, o_r its *region_id*.

Operator	*Definition*
$a \supset b$	$true \iff s_b > s_a \wedge e_b < e_a$
$A \bowtie_{\supset} B$	$\{(o_a, o_b) \mid a \leftarrow A,\ b \leftarrow B,\ a \supset b\}$

12.4 Query Processing

To clarify the difference between a 'traditional', black-box system architecture and our proposed architecture, this section presents the query processing

employed in the prototype system that takes place to evaluate the example query. To keep things simple, we present the physical layer as a familiar SQL database; in the prototype implementation however, we use the Monet Interface Language (MIL, [38]) gaining better control over the generated query plans.

Wordindex (W)

position	term
2	'...'
3	'...'
8	'Will..'
9	'...'
12	'Maxi..'
17	'dating'
20	'...'
21	'...'
...	...

Nodeindex (N)

region_ id	start	end	tag
0	0	25	article
1	1	4	title
2	5	24	bdy
3	6	14	sec
4	7	10	p
5	11	13	p
6	15	23	sec
7	16	18	p
8	19	22	p
...	...	...	...

Query (Q)

qterm
'Maxima'
'dating'

Attribute index (A)

region_ id	attr_id	attr_ name	attr_ val
0	0	lang	...
0	1	pdate	...
...	...	...	...

Fig. 12.4. Database schema of our XML IR system.

12.4.1 XML Processing

The first part of the generated query plan focuses on the processing of structural constraints, and is handled in the logical XML extension. For the example query, it identifies the document components in the collection that are subsequently ranked by the IR extension, which implements the about function. The XML processing extension produces its query plans based upon the region indexing scheme outlined in Section 12.3, using the physical database schema shown in Figure 12.4. It selects the collection of article components specified by XPath expression `//article/text()` (a collection of bags of words), filtered by the specified constraints on publication date and language attributes:

```
articles :=
    SELECT n.region_id, start, end
    FROM nodeindex n,
         attributeindex al, attributeindex ap
    WHERE n.tag = 'article'
      AND al.region_id = n.region_id
```

```
        AND al.name = 'lang'
        AND al.value = 'English'
        AND ap.region_id = n.region_id
        AND ap.name = 'pdate'
        AND ap.value BETWEEN '31-1-2003' AND '1-3-2003';

mat_articles :=
    SELECT a.region_id, w.position
    FROM articles a, wordindex w
    WHERE a.start < w.position AND w.position < a.end;
```

The resulting query plan is rather straightforward; it selects those article components satisfying the attribute constraints, and materializes the text occurring in these article components. Materialization of the text is handled by the containment-join $\bowtie_{\supset}$, specified in the second SQL query by the range predicates on the position attribute of word index $\mathcal{W}$.

Notice that the logical XML extension generates a query plan that is understood by the physical level of the system (and could be executed as is), but that this plan has not been executed yet, neither has the ordering in which the relational operators are to be evaluated been fixed at this point! A main advantage of deferring the query evaluation is that the physical layer can still use its statistics maintained about the data, for instance to decide upon the predicate that is most selective. This also improves the likelihood that intermediate query results can be reused, e.g., between sessions (when the same user always reads English articles only) or shared across different users (all selecting usually article nodes with recent publication date). A third advantage will be discussed after the logical extension for information retrieval processing has been introduced.

12.4.2 IR Processing

The next step in this discussion focuses on the logical extension for IR processing; in our example query, this extension handles the ranking of article components selected by the XML extension.

The prototype system uses Hiemstra's statistical language modeling approach for the retrieval model underlying the about function (Chapter 7 of this book). The selected XML sub-documents are thus ranked by a linear combination of term frequency (tf) and document frequency (df). The language model smoothes probability $P(T_i|D_j)$ (for which the tf statistic is a maximum likelihood estimator) with a background model $P(T_i)$ (for which the df statistic is a maximum likelihood estimator), computing the document component's retrieval status value by aggregating the independent scores of each query term.

The IR processing extension at the logical level manipulates collections of bag-of-words representations of the document components to be ranked. Let us first consider the calculation of the term probabilities. This requires the

normalization of term frequency with document component length. Calculation of $P(T_i|D_j)$ is thus outlined in the following SQL fragment:

```
mat_art_len :=
    SELECT mat_articles.region_id, count(*) AS length
    FROM mat_articles GROUP BY mat_articles.region_id;

ntf_ij :=
    SELECT mat_articles.region_id, w.term,
      (count(*) / mat_art_len.length) AS prob
    FROM mat_articles, mat_art_len, wordindex w, query q
    WHERE w.term = q.qterm
      AND mat_articles.position = w.position
      AND mat_articles.region_id = mat_art_len.region_id
    GROUP BY mat_articles.region_id, w.term;
```

For generality, the computation of these probabilities has been assumed completely dynamic, for the IR extension cannot predict what node sets in the XML collection will be used for ranking. In practice however, when the collection is mostly static and the same node sets are used repeatedly for ranking (e.g., users ranking always subsets of `//article/text()`), the relations storing term counts and component lengths should obviously be maintained as materialized views.

Similar arguments hold for the estimation of $P(T_i)$, the term probability in the background model. As explained in [89] however, the collection from which the background statistics are to be estimated should be specified as a parameter of the about operator (alternatively, the right scope could be guessed by the system). Let the collection of all article nodes be appropriate in the example query (and not the subset resulting from the attribute selections), and the following queries compute the background statistics:

```
num_art :=
    SELECT COUNT(*) FROM nodeindex WHERE tag='article';

art_qterm :=
    SELECT DISTINCT n.region_id, w.term
    FROM nodeindex n, wordindex w, query q
    WHERE w.term = q.qterm
      AND n.tag = 'article'
      AND n.start < w.position AND w.position < n.end;

ndf_i :=
    SELECT term, (count(*) / num_art) AS prob
    FROM art_qterm GROUP BY term;
```

The final step computes the ranking function from the intermediate term probabilities in document and collection:

```
ranks :=
    SELECT ntf_ij.region_id,
      sum(log(1.0 + ((ntf_ij.prob / ndf_i.prob) *
```

```
        (0.15 / 0.85)))) AS rank
FROM ntf_ij, ndf_i
WHERE ntf_ij.term = ndf_i.term
GROUP BY ntf_ij.region_id
ORDER BY rank DESC LIMIT 100;
```

12.4.3 Inter-extension Optimization

We now present the third advantage of the Multi-Model DBMS architecture: optimization is not limited to *within* extensions themselves, but can also be performed *between* extensions. An example of such inter-extension optimization is the processing of the *text()*-function. Formally, this function materializes *all* text within an XML node and the generated plan for materialization of all article text has been presented. Assuming for example an object-relational approach with two blackbox extensions, one for XPath expressions and one for IR processing, the XPath blackbox extension would have had to materialize all text occurring in article nodes as result of `//articles/text()` that is the input for the about operator of the IR extension.

In our case however, the IR extension takes a query plan as input that it simply augments with its own operations. The physical layer of the architecture can easily detect that *only* the terms occurring in the query string have to be materialized to compute the correct results. The query term selection predicate (`wordindex.term = query.qterm`) is simply pushed up into the expression for `mat_articles`. The much smaller intermediate result to be materialized reduces significantly the bandwidth needed for evaluating the full XQuery expression - especially in cases when the query terms occur infrequently in the corpus.

Summarizing, the main benefit of the proposed Multi-Model DBMS architecture is the ability to make such optimization decisions (either within extensions or between extensions) at run-time. The logical extensions expose their domain knowledge in terms understood by the physical layer, still allowing it to intervene if necessary (instead of fixing the query execution order by themselves). Also, reuse of intermediate results is easier realized. The precise nature of the optimization process is a central focus in our current research.

12.5 Discussion

The information retrieval models discussed so far have been straightforward, ignoring semantic information from XML tags, as well as most of the logical and conceptual structure of the documents. In spite of the simplicity of the retrieval models discussed, these examples demonstrate the suitability of the 'database approach' for information retrieval applications. The next step in our research is to determine *what* extra knowledge we need to add to increase retrieval effectiveness. Development of new retrieval models (that exploit the

full potential benefit of XML documents) can only be based on participation in evaluations (like INEX, see Chapter 19), and we expect the flexibility of the database approach to information retrieval to help invent these models more easily.

The integration of IR and database technology will offer more apparent advantages in future retrieval systems, supporting more complex search strategies and advanced query processing techniques. Consider for example a slightly modified scenario for our running example: we drop 'recent' from the information need, and assume automatic query expansion to be part of the IR process. It is quite likely that 'Willem-Alexander' is only referred to as 'the Dutch Crown Prince' in some of the English newspaper articles. Similarly, it is likely that some other articles mention both 'Willem-Alexander' and 'the Dutch Crown Prince'. Thus, we hypothesize that query expansion with terms from documents containing both 'Willem-Alexander' and 'Maxima' would improve recall with high likelihood.

Generalizing this scenario, we define a strategy for queries containing several (more than one) named entities. For these queries, we first retrieve a small number of documents that contain (most of) these named entities *and* rank high using the full query. Next, we perform query expansion with blind feedback using these documents. Finally, we rank the full collection using the expanded query.

In a retrieval system based on black-boxes, implementing a strategy like this can be rather tricky. Terms from the collection that are tagged as named entities would be stored outside the IR system, probably in a DBMS like structured data. The ranking system cannot usually be instructed to only retrieve documents that contain at least these terms. And, as argued before, the blind feedback process is often implemented on top of the core IR engine, which probably does not cache intermediate results. To process the full query, we travel between the boundaries of systems more than once, which will clearly reduce the efficiency of the system.

Advanced query processing techniques include optimization strategies adapted to an interactive environment, allowing search with precise queries, browsing based on online clustering of search results, and query refinement using relevance feedback. Horizontal fragmentation might be exploited for handling larger collections, like in [32]: distribute large sets of XML documents over a farm of servers and fragment the vocabulary based on term frequency, either globally or within documents (or document regions).

Using the quality assessments from the INEX evaluation, we can investigate trading quality for speed, as we did for full-text retrieval using TREC benchmark data [33]. Like in text retrieval, XQuery-Fulltext retrieval will result in many documents, and now also document regions, that match a query more or less. As mentioned before, this does require ranking and user feedback, resulting in an iterative process that should converge to a good final result set as quickly and effectively as possible. This means the user does not want the steps to get there to take very long. Also, in the first iterations ex-

plicitly presenting expectedly bad results to the user might very well speed up the entire process as the negative user feedback on those results will rule out significant parts of the search space for processing in further iterations. Trading quality for speed is an interesting option for the first steps of the user.

12.6 Conclusions

We have identified two types of challenges for IR systems, that are difficult to address with the current engineering practice of hard-coding the ranking process in highly optimized inverted file structures. We propose that the trade-off between flexibility and efficiency may be resolved by adopting a 'database approach' to IR. The main advantage of adhering to the database approach is that it provides a system architecture allowing to balance flexibility and efficiency. Flexibility is obtained by declarative specification of the retrieval model, and efficiency is addressed through algebraic optimization in the mapping process from specification to query plan.

Existing (relational) database system architectures are however inadequate for proper integration of querying on content and structure. The Multi-Model DBMS architecture is proposed as an alternative design for extending database technology for this type of retrieval applications. Discussing the query processing strategies for an example query combining content and structure, the main differences with existing blackbox approaches for extending database technology are explained.

The chapter has been concluded with a discussion of future directions in IR system implementation for which our proposed architecture is of even more importance. In particular, we claim that both fragmentation as well as the optimization though quality prediction would benefit greatly from an open, extensible, layered approach, i.e., the advantages of the Multi-Model DBMS architecture.