ON THE INTEGRATION OF IR AND DATABASES

Arjen P. de Vries and Annita N. Wilschut

Centre for Telematics and Information Technology
University of Twente
The Netherlands
{arjen,annita}@cs.utwente.nl

Abstract: Integration of information retrieval (IR) in database management systems (DBMSs) has proven difficult. Previous attempts to integration suffered from inherent performance problems, or lacked desirable separation between logical and physical data models. To overcome these problems, we discuss a database approach based on structural object-orientation. We implement IR techniques using extensions in an object algebra called MOA. MOA has been implemented on top of the database backend Monet, a state-of-the-art high-performance database kernel with a binary relational interface. Our prototype implementation of the inference network retrieval model using MOA and Monet demonstrates the feasibility of this approach. We conclude with a discussion of the advantages of our database design.

INTRODUCTION

Information retrieval (IR) is concerned with the retrieval of (usually text) documents that are likely to be relevant to the user's information need as expressed by his request (van Rijsbergen, 1979). IR retrieves documents based on their content. Because the request (or query) is never a perfect expression of the information need, IR is an iterative process.

Database management systems (DBMSs) traditionally support a different kind of retrieval. Van Rijsbergen summarizes the differences as follows. Retrieval in databases is deductive and exact. Retrieval in IR uses inductive inference and is approximate. Objects in a database possess attributes both necessary and sufficient to belong to a class. Conversely, no attribute of a document's content is necessary nor sufficient to judge relevance to an information need.

As a result, DBMSs do not sufficiently support searching on content and IR systems do not handle structured data. However, many applications of information systems have requirements that can only be matched with a *combination* of data retrieval and information retrieval: e.g. patient's data in hospital systems, multimedia data in digital libraries, and business reports in office information systems. The characteristic feature of such applications is the combination of content management with the usual manipulation of formatted data; these two aspects are often referred to as the **logical structure** and the **content structure** of a document (Meghini et al., 1991).

Consider an information request in a digital library for 'recent news bulletins about earthquakes in California'. In this example, 'recent' and 'news' refer to attributes of the objects in the library (which are in its logical structure), while 'earthquake' and 'California' refer to the content of those objects (which is its content structure). Note that the combination between constraints on content and other attributes of the documents is also important for the implementation of Mizzaro's different notions of relevance in the IR process (Mizzaro, 1998).

Another argument in favour of integrated systems is from an architectural point of view. Databases and IR systems share requirements like concurrency control, recovery, and internal indexing techniques. Database management systems (DBMSs) already provide such support. The integration of IR in databases can also help the IR researcher to concentrate on retrieval models and reduce the effort of implementation involved with experimental IR.

This paper presents an implemented prototype of an integrated IR and database system. In the next section we review previous approaches to such integration, discuss some problems with these systems, and present an outline of our design.

BACKGROUND AND PROBLEM STATEMENT

Information retrieval systems

The development of an IR system consists of three steps (see e.g. (Wong and Yao, 1995)). First, we choose an appropriate scheme to represent the documents. Second, we model query formulation. Third, we select a ranking function which determines the extent to which a document is relevant to a query. The combination of these three steps is known as the **retrieval model**.

Usually, documents and queries are simply represented as a collection of words. Sometimes, the words are reduced to their stems (**stemming**) and often frequent words with little semantics are left out (**stopping**). Techniques from natural language processing may be applied to identify phrases. The resulting features of these procedures are called the indexing **terms**.

A wide variety of ranking formulas exists; see (Zobel and Moffat, 1998) for an overview of the most common ones. Typically, the ranking is determined using statistics about the distribution of the indexing terms over the document collection. Most statistics are based on the **term frequency** (tf), the number of times that a term occurs in a document. Usually, the raw values are

normalized to account for document length before they are used for ranking, e.g. by dividing with $max\ tf$. Sometimes, log-normalization is used because the difference between one or two occurences of a query term in a document is much more significant than the difference between fifteen or sixteen times. For example, the **term frequency** component (tf) in the ranking formula of the popular retrieval system InQuery is (Callan et al., 1995):

$$0.4 + 0.6 \cdot \frac{\log(tf + 0.5)}{\log(max\ tf + 1.0)} \tag{1.1}$$

The document frequency (df) is the number of documents in which a term occurs. Because a high term frequency of 'car' in a document from a collection of documents about cars does not add much information about the relevance of that document for the query 'red sports cars', tf is usually multiplied with the **inverse document frequency**. The inverse document frequency (idf) is usually defined as $\log \frac{N_d}{df}$, and expresses the amount of surprise when we observe a term in a document. The product $tf \cdot idf$ is the most popular weighting in IR and is used in most ranking formulas.

Integration of IR in object databases

Previous approaches to IR and database integration have suggested the application of object-oriented databases. Object data models allow nesting, and in addition to sets, work with multiple collection types. Therefore, an object-oriented model is very suited for the implementation of IR. A good example of the design of IR in an OODBMS is the Cobra system described in (Mills et al., 1997).

Unfortunately, the design of OODBMSs often provides little support for **data independence**. Although object-orientation is very useful for data modeling of complex objects and their behaviour, this does not necessarily imply that the implementation of the objects at the database level should be the same as the view on the objects at the conceptual level. Often, the implementation of object databases uses conventional programming languages to extend the OODBMS. The structure of the complex objects is hard-coded in the implementation of the object, and only the interface of the complex object is known to the database. The system cannot 'look inside' the objects to plan the execution of a query that involves several method-calls. For many operations on the complex objects, this will lead to object-at-a-time processing instead of the often preferrable collection-at-a-time processing. Algebraic optimization of query expressions, as proven very succesful for relational databases, is hard to achieve because the structure of the objects is hidden inside of the object. Concurrency control and parallellization can only be defined at object granularity.

Data independence in OO: structural object-orientation

We can obtain better data independence in object databases when we base their design on **structural object-orientation**. In a database system with

support for structural object-orientation, we distinguish between **atomic** data types and **structures** over these types. The structure definitions are part of the extensible data model. Because the database manages the structure of the objects, and not an extension module written in an external programming language, it *can* 'look inside' the objects and make better decisions with respect to query processing and concurrency control.

In this paper, we study an approach based on structural object-orientation for the design of an integrated IR and database system. This design consists of three levels. At the top level of our design, we model the information retrieval functionality. We define extensions that model the three components of a retrieval model: document representation, query formulation, and the ranking process. This level implements the management of content and adds it to the object algebra. The second level is our object algebra called **MOA**. MOA provides a nested object data model using bags, objects, and tuples. The formal definition of **structuring primitives** allows the addition of the specific structures for information retrieval. MOA provides the functionality required to manage the logical structure of the documents. At the bottom level, MOA object algebra is mapped to a flat binary relational model. This datamodel is employed by **Monet**, an efficient database kernel that is used as a backend for query execution. The use of a different physical data model brings us the benefit of data independence and avoids the pitfalls of object-oriented databases mentioned before.

In the remainder of this paper, we discuss the implementation of an integrated IR and database system using structural object-orientation. We concentrate on the database aspects of our design. We start in section 1 with a quick overview of the Monet database and MOA object algebra. In section 2, we discuss the integration of IR in MOA and Monet. We design new MOA structures that support IR functionality, introduce the physical data model, and define algebraic operations to support IR. After discussing the resulting design in section, we finish the paper with conclusions and further research.

MONET AND THE MOA OBJECT ALGEBRA

In this section, we give overviews of the database Monet, the MOA object algebra, and its implementation on Monet.

Monet

Monet is an extensible parallel database kernel that has been developed at the UvA and the CWI since 1994 (Boncz and Kersten, 1995). Monet implements a binary relational model; the data is stored in **Binary Association Tables** (BATs). BATs are tables with two columns, the **head** and the **tail**, each storing an atomic value. Structured data is decomposed over many narrow tables. The system is intended to serve as a backend in various application domains. In this research, we used Monet to implement an IR retrieval model with algebraic database operations. Monet has also been used succesfully as

backend for geographic information systems as well as commercial data mining applications.

Monet's design is based on two trends. First, the average main memory in workstations gets larger and larger. Therefore, processing should be focused on operations that are performed in main memory instead of on disk; all primitive operations in Monet assume that their data fit in main-memory. Second, operating systems evolve towards micro-kernels, i.e. they make part of the OS functionality accessible to applications. A DBMS should therefore not attempt to 'improve' or 'replace' the OS-functionality for memory management. If the tables in Monet get too large for main memory, the database uses memory mapped files. It uses the lower level OS primitives to advice on the buffer management strategy. This way, the MMU can do the job in hardware.

The **Monet Interface Language** (MIL) consists of the BAT-algebra, which contains basic operations on bags including `select`, `join`, and `semijoin`, as well as a collection of control structures. The algebra operations materialize their results and never change their operands. They perform an additional **dynamic optimization** step before execution. Based on the properties of the operands, the most efficient algorithm is chosen. For instance, the join algorithm may be a `hashjoin`, but also a `mergejoin` that assumes the join columns to be ordered, or a `syncjoin` that assumes the join columns to be identical. When two BATs have an identical head column, they are said to be **synced**. Join operations can be performed very efficiently on synced BATs, because we do not have to compare the values - we know beforehand that the head values in the operands are equal. This way, the extra cost for re-assembling the vertically fragmented multi-attribute data is reduced significantly, which has been demonstrated on the TPC-D decision support benchmark in (Boncz et al., 1998).

MOA, an algebra on an extensible object datamodel

MOA consists of an extensible object datamodel and an algebra on this data model. This section summarizes the datamodel and the algebra. Note that we do not introduce MOA as a language for end-users. We aim at support for a full-fledged declarative object query language like OQL (Catell et al., 1997). MOA is only an implementation platform for such a language. At present, we have not yet implemented a full translation of OQL to MOA. Therefore, our current systems should be considered only as a basis for application developers.

Datamodel. The MOA data model is based on the concepts of **base types** and **structuring primitives**. MOA assumes a finite set of ADT-style base types. Base values (which are instances of base types) are atomic; their internal structure cannot be accessed, and is only accessible via operations. Base types are implemented at the level of the physical storage system, and therefore allow efficient implementation in a general-purpose programming language. The base types for our implementation of MOA are provided by the underlying physical

system, Monet. Therefore, MOA inherits the base type extensibility provided by Monet.

A structuring primitive, or structure for short, combines known types to create a **structured type**. Structures can be used recursively. The set, bag, and tuple are well-known structuring primitives that are provided in all OO data models. The MOA kernel supports a generic notion of orthogonal structures. Actual implementations of structures are added to the kernel in separate modules. MOA also supports the notion of classes and objects. As these concepts are not directly relevant in the context of this paper, they are not discussed here. MOA's type system can now be summarized as follows:

base types: τ is a type if τ is an atomic type.

structured types: If $\tau_1, \cdots, \tau_n$ is a, possibly empty, list of types and $\mathcal{T}$ is a structure defined over $\tau_1, \cdots, \tau_n$, then $\mathcal{T}(\tau_1, \cdots, \tau_n)$ is a structured type.

The collection of structures constitutes the datamodel at the logical level. These structures are mapped by the MOA implementation on the physical system. This mapping provides data independence between the MOA data model and the physical storage system. The logical-to-physical mapping also allows the system to optimize query execution plans. Our implementation of MOA provides bag and tuple structures. The result is an orthogonal type system that consists of atomic types, a bag structure, and a tuple structure, that is extensible with user-defined structures. In this paper, we further assume that MOA is supplied with the bag and the tuple structure, and we will define a number of structures to model documents, and statistics over document collections.

Algebra. The algebra on the MOA data model consists of the operations defined on the available atomic and structured types. Each atomic type has its own accessor functions. The accessor functions on the atomic types are executed directly in the physical system. Each structure definition comes with operations on the structure. The logical MOA operations on structures are translated into efficient physical execution plans. In MOA, the bag structure implements many operations that are usual in query algebras, like the selection, map, join, semijoin, nest, unnest, etc. The tuple structure implements an operation to extract an attribute from the tuple. Atomic types simply add their accessor functions to the algebra.

MOA example. Let `X` be a value of the nested structured type defined in code example 1. MOA's `select` operation selects those tuples from the operand bag into a result bag for which the zeroth (integer) attribute has value 1. The result is a value of the same type as the operand. After the selection operation, we may remove the integer attribute from the resulting tuples using a `map` operation.

Example 1

```
X: BAG<TUPLE<integer, string, BAG<string>>>;
```

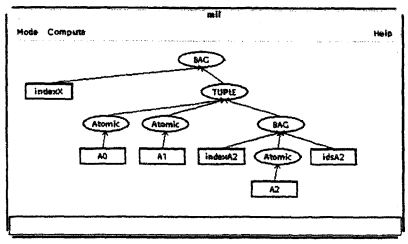


Figure 1 The mapping of the MOA data model on Monet's binary tables

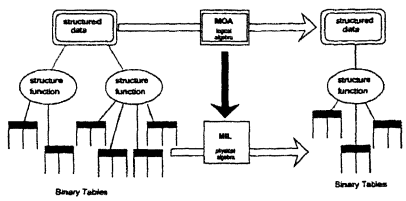


Figure 2 MOA query execution by translation to MIL

```
select[attr(THIS,0) = 1](X);
map[TUPLE<attr(THIS,1), attr(THIS,2)>](
  select[attr(THIS,0) = 1](X)
);
```

Assume now that Y is a value of type BAG<integer>. The following join operation joins the elements of bag X with bag Y on the equality of the count of the bag attribute in the tuples in X and value of the integers in Y:

```
join[count(attr(THIS,2)), THIS, TUPLE<>](X,Y);
```

This join uses the TUPLE structure to generate the result. Consequently, the type of the result is

```
BAG<TUPLE<TUPLE<integer, string,BAG<string>>,integer>>.
```

Implementation. We illustrate the main idea using the definition of structured value X given in example 1. The full details of the mapping of MOA on Monet and an evaluation of its performance are described in (Boncz et al., 1998). Because Monet only has binary tables, we have to use full vertical decomposition (Copeland and Koshafian, 1985) to store structured data. The combination of BATs storing values with a **structure function** on those BATs forms the *representation of a structured value.*

The structure function allows the reconstruction of value X from its decomposition. Figure 1 shows the mapping of X on Monet. The rectangles correspond to BATs, and the ovals together constitute its structure function:

```
BAG< indexX,
  TUPLE< Atomic<A0>, Atomic<A1>,
         BAG< indexA2, Atomic<A2>, idsA2 >>>;
```

There is a one-to-one correspondence between a structure function and the structured type that is mapped via the structure function. The idea behind the algebra implementation is to translate a query on the representation of the structured operands into a representation of the structured query result. Figure 2 illustrates this process: the query is a MOA expression on a structure

function on BATs; its translation is a MIL program on the operand BATs that generates result BATs, which in turn are the operands of another structure function representing the result.

INTEGRATION OF IR IN MOA/MONET

IR in MOA

In this section, we extend MOA with structures for querying document collections by content. Note that, aiming for data independence, we add the representation of documents *as a structured type* rather than following common practice using base type extensions.

We first define structuring primitive DOCREP for the representation of document content. DOCREP's implementation manages the document representation at the physical level. The combination of DOCREP with other MOA structures allows us to flexibly model document collections. The data model most similar to 'normal' IR systems is the specification of a document collection as a BAG of DOCREPs. A more interesting specification uses the standard structure TUPLE to describe a document as the combination of its content and its logical structure. Example 2 shows the selection of 'News' documents from a document collection docs, in which documents are modelled as a tuple of category and content.

Example 2

- *document collection structure definition:*

```
BAG< TUPLE< Category :  str,
            Content  :  DOCREP > >;
```

- *its associated structure function:*

```
docs ::=
  BAG<docidx, TUPLE<Atomic<Category>,
    DOCREP< dj, ti, tfij, docidx >>>;
```

- *expression for subset selection:*

```
select[ =( THIS.Category, "News" ) ] ( docs );
```

When querying a collection on content, we compute a score to express the belief that a document is relevant for the query. The computation of this belief usually requires global statistics of the document collection. Because we may have several document collections in one database, we model the collection statistics explicitely in the DCSTAT structure. Also, when we reduce a collection using conditions on the logical structure of documents, we may in some cases want to use the statistics of the subcollection for content querying (e.g. when selecting documents that are news items), but in other cases use the statistics of the original collection (e.g. when selecting the documents written at University of Twente). An explicit structure for the collection statistics makes both strategies possible.

In the current implementation, a DCSTAT is constructed using a collection name, the dictionary (or indexing vocabulary), the number of documents in the collection, and the document frequencies (df) of the indexing terms. A DCSTAT can also be obtained from a given document collection, using the aggregate getStats defined in structure DOCREP. DCSTAT operation nidf encapsulates the calculation of normalized inverse document frequency scores for the query terms in its operand. If we leave out the query terms, nidf returns tuples of term identifier with normalized idf values for all terms in the indexing vocabulary. Thus, we get the same answer with a semijoin between its result and the query terms. Example 3 shows the queries for both approaches, assuming that stats and query are extents of the structure definitions of nidf's input arguments.

Example 3

- *nidf method definition:*

```
in:  DCSTAT, BAG< oid >
out:  TUPLE< termid:  oid, nidf:  dbl >
```

- *equivalent expressions for idf of query terms:*

```
nidf( stats, query );
semijoin[ THIS.nidf, THIS, TUPLE<> ](
  nidf( stats ), query );
```

Because an IR retrieval model calculates a belief score for a document given a query, one would expect the full belief calculation to be specified in a method of DOCREP. Anticipating the multimedia retrieval model that we proposed in (de Vries and Blanken, 1998), we prefer to introduce an extra structure INFNET that models belief combination. Structure DOCNET is a subclass of this structure, that handles networks with default beliefs. The complete process of belief computation is now divided in two steps, see example 4. First, for each document in the collection, the getBL operation, defined on DOCREP, constructs a DOCNET that encapsulates the term beliefs for the query terms. Next, a DOCNET operation combines these term beliefs into a final document judgement. This separation of belief computation in two steps allows for future extensions of the retrieval model with structured types for multimedia content, e.g. IMGREP, that will also produce DOCNETs for a given query.

Example 4

```
map[sum(THIS)](
  map[getBL(THIS, query, stats)]( docs )
);
```

In example 5 we take full advantage of the integration of IR and databases, combining retrieval by content with constraints on the document's logical structure: the MOA expression computes the ranking of documents that match the

query *and* are in the 'news' category. When applied first, the select on category may significantly reduce the number of computations required in the IR processing of the document content.

Example 5

```
map[sum(getBL( THIS.Content, query, stats )) ](
  select[ =( THIS.Category, "News" ) ]( docs )
);
```

We conclude the overview of MOA's IR extensions with an example illustrating another advantage of this combination. Often, we would prefer to rank **compound documents** on logical units like sections or chapters, rather than on their full content. MOA's nested data model makes it relatively easy to model IR on compound documents. In example 6, we model the document content as a bag of items. The topology of the inference network specified by this particular query is taken from (Callan, 1994). His experiments suggested that the best results are achieved when a document is ranked by the contribution of its best component. Of course, variations in the network topology can be expressed in MOA in a similar manner.

Example 6

- *document collection structure definition for compound documents:*

```
BAG< TUPLE< Category :  str,
            Content  :  BAG< DOCREP > > >;
```

- *ranking news documents by their best items*

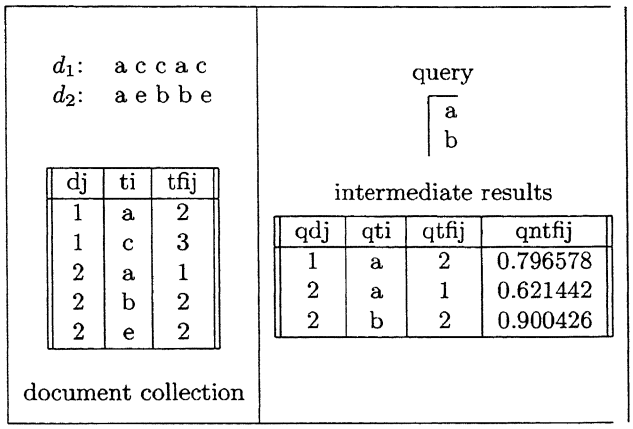
```
map[max( INFNET<THIS> ) ](
  map[ map[ sum(getBL( THIS,
              query, stats ))]( THIS.Content ) ](
    select[ =( THIS.Category, "News" ) ] ( docs )));
```

Algebraic processing in IR implementation

Representation of document content. In this subsection, we describe the bottom layer of our IR implementation. The MOA structures and expressions on document content are translated to operations in the physical database. Thus, the information about the documents that is used in the IR process must be stored in BATs. We term this the *flattened document representation.* At a first sight, this mapping may seem clumsy and only an introduction of extra complexity. This representation is however the foundation of an algebraic implementation of IR in Monet, hence the key to data independence, query optimization and parallellization.

Table 1 shows an example of a flattened collection consisting of two documents.[1] A document representation is described with the three synced BATs `dj`, `ti`,

Table 1 Representation of documents in BATs



and `tfij`. These BATs store the frequency tf_{ij} of term t_i in document d_j for each term t_i occuring in document d_j. When computing document scores for query q we proceed as follows. First, we join the query terms with the document terms. Next, using additional joins we look up the document ids and term frequencies. Note that these joins are executed very efficiently, because the BATs are synced. Assuming the belief in a term is given by equation 1.1, we need the document-specific values of *max tf*. Given our flattened document representation, these are computed using a set-aggregate version of `max`. The normalized term frequencies are computed from the *tf* and *max tf* tables with a user-defined operator `ntf`. Similarly, we can compute normalized inverse document frequencies from the collection statistics, and use these in combination with normalized term frequencies and document lengths to produce a wide variety of ranking formulas.

User-defined operators in Monet. To support the belief computations in Monet, we extend MIL with new algebraic operations. Monet allows the definition of new operations in two ways: define new procedures in MIL, or add extensions written in C or C++. MIL procedure `ntf`, defined in code example 7, computes normalized term frequency *tf* using equation 1.1. Because there exists not yet a 'best' model for information retrieval, many different ranking formulas may be used in retrieval experiments. Implementing these ranking formulas as MIL procedures is very convenient for experimentation; we do not have to recompile our code every time we want to try a new ranking formula.

Example 7

```
PROC ntf( tf, maxtf ) :=
```

Table 2 Intermediate tables with and without outer join

doc	term	belief
d_1	a	0.56
d_1	b	default
d_2	a	0.67
d_2	b	0.82

doc	term	belief
d_1	a	0.56
d_2	a	0.67
d_2	b	0.82

```
RETURN 0.4 + 0.6*(log10(tf + 0.5)/log10(maxtf + 1.0));
```

Probabilistic reasoning in Monet. After the term-based probability estimates have been calculated, a combined score is computed to express the belief in a document given the query. We use Monet's modular extension framework for the efficient implementation of this belief computation. The extension framework supports the implementation of user-defined data types, user-defined search accelerators, and user-defined functions.

We compute the document belief scores based on the well-known **inference network retrieval model** (Turtle and Croft, 1992). This retrieval model is based on a restricted class of Bayesian inference networks. An efficient implementation of this model, the InQuery system, has been shown very effective in the TREC evaluations (Callan et al., 1995). (Vasanthakumar et al., 1996) expressed the computations in the inference retrieval network model as SQL queries in the DEC Rdb V6.0 relational DBMS. User-defined functions encapsulated the implementation of InQuery's probabilistic operators. Our implementation of this retrieval model resembles their approach. The probabilistic operator `PICEval`, that combines the term beliefs into document beliefs, is based on the algorithms given in (Greiff et al., 1998).

A problem with the database implementation of the inference network given in (Vasanthakumar et al., 1996) is the computation of a **full outer join** between the terms occuring in the query and the terms occuring in the documents. Thus, the query terms *not* occurring in a document, *are* represented physically in the intermediate results. If the query consists of n_q query terms and the document collection consists of N_d documents, then the intermediate result requires space for $n_q \cdot N_d$ records. Because most documents will only match a small portion of the query, the intermediate result is likely to be a long but sparse table. The storage requirements for this table will cause a performance bottleneck. Note that even when the user enters a short query, it is common in IR systems to increase the number of query terms using query expansion techniques.

The full outer join is only used to assign default beliefs to the terms that do not occur in the document. A solution without the full outer join is possible when we handle the assignment of default beliefs locally, *inside* the probabilistic operators. Hereto, we developed the aggregate function `queryPICEval`, a special version of `PICEval` that takes the query terms as an extra operand. Instead of inserting default beliefs for all documents beforehand, these are only inserted

per document during the combination of evidence itself. Table 2 illustrates the difference in processing between the two approaches.

Discussion

In the previous subsections, we gave an overview of our implementation of IR query processing in the MOA/Monet environment, based on a mixture of kernel operators and user-defined functions. Here, we summarize the benefits of the data independence offered in our design.

The combination of MOA and its extensions for IR makes it possible to flexibly combine constraints on the *content* of documents with constraints on their *logical* structure. The nested data model also allows us to model compound documents easily. And, we can easily manage several versions of the same underlying collection, e.g. using different stemming algorithms, or extracting concepts with varying NLP techniques. The separation of tasks between the MIL procedures defined in the database and the structural definition of the document collection in MOA supports the design of IR experiments. The MOA expressions that define the experiment do not change when we tweak the parameters of the retrieval model.

The clear distinction between the specification of a retrieval model in algebraic operations and its physical execution is also a nice property for experimental IR research. The researcher can develop new retrieval models without worrying too much about low-level implementation issues like hash tables, index trees and parallellization. The kernel chooses at runtime from a variety of algorithms to execute the algebraic operations as efficient as possible. Similar to the use of indices in relational databases, we may increase performance by the definition of access structures on the BATs without having to change the code. Furthermore, when executing on a parallel machine, the IR code automatically benefits from the implementation of parallel kernel algorithms.

These benefits can *in principle* also be achieved using just Monet. However, writing MIL programs on the flattened document representation requires a lot of knowledge of the mapping from documents to BATs. For example, the expression given in example 5 is translated into a MIL program consisting of 36 operations, and is already quite complex to comprehend, let alone construct. Clearly, MOA expressions have a great advantage over the low-level BAT manipulation. Also, the MOA expressions can be manipulated in an algebraic query optimizer. A 'push select down' strategy that first evaluates constraints on the logical structure and then constraints on the content should be easily implemented for MOA expressions; for MIL programs however, the search space would be too large to accomplish the equivalent rewrite.

CONCLUSIONS AND FUTURE RESEARCH

We reported the integration of IR and databases using structural object-orientation and demonstrated the feasibility of our design with a prototype implementation. We explained the benefits of an integrated approach to IR and databases,

and gave several examples illustrating the kind of queries that can be specified in our system. At the top level, we integrated a well known information retrieval model in MOA object algebra. We defined the belief computations as algebraic operations in the database kernel, with a strong emphasis on set-at-a-time operations. This approach provides data independence between the logical and the physical database. As a result, parallellization and the selection of access structures have become orthogonal to the development of an information retrieval model. More importantly, an algebraic approach with an object-oriented interface allows algebraic query optimization.

The next step in our research is going to be an experimental evaluation of the performance of the prototype system on standard IR test collections. Further research goals include the implementation of a multimedia digital library, following the structural object-oriented approach used in this paper. We also aim to implement a variety of text retrieval models, and use Monet's extensibility to add ADTs for proximity information. With respect to optimization, we are especially interested in the role of new structures during query optimization. Also, we plan to study the optimization that can take place when we are only interested in the N-best matches. Finally, we want to scale-up to very large document collections via the exploitation of parallellism. For this purpose, we plan to extend the MOA implementation to generate parallel MIL programs.

Acknowledgements

Jan Flokstra provided us with the infrastructure to make this work possible. We are also grateful to the Monet team in general for their explanations and support, and want to thank Peter Boncz in particular, without whom prototyping our ideas on Monet would have been impossible.

Notes

1. Notice that we do not model proximity information in our current implementation. Using Monet's extensibility of data types, we expect however no problems in providing an efficient implementation of proximity operators. The implementation of an ADT for word location lists bears similarity with the `polygon` ADT in the GIS extensions.

References

Boncz, P. and Kersten, M. (1995). Monet: An impressionist sketch of an advanced database system. In *BIWIT'95: Basque international workshop on information technology.*

Boncz, P., Wilschut, A., and Kersten, M. (1998). Flattening an object algebra to provide performance. In *Fourteenth International Conference on Data Engineering*, pages 568–577, Orlando, Florida.

Callan, J. (1994). Passage-level evidence in document retrieval. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland.

Callan, J., Croft, W., and Broglio, J. (1995). TREC and TIPSTER experiments with INQUERY. *Information Processing and Management*, 31(3):327–343.

Catell, R., Barry, D., Bartels, D., Berler, M., Eastman, J., Gamerman, S., Jordan, D., Springer, A., Strickland, H., and Wade, D. (1997). *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann Publishers Inc.

Copeland, G. and Koshafian, S. (1985). A decomposition storage model. In *Proceedings of the SIGMOD Conference*, pages 268–279.

de Vries, A. and Blanken, H. (1998). The relationship between IR and multimedia databases. In *IRSG'98*, Autrans, France.

Greiff, W., Croft, W., and Turtle, H. (1998). PIC matrices: A computationally tractable class of probabilistic query operators. Technical Report IR-132, The Center for Intelligent Information Retrieval. submitted to ACM TOIS.

Meghini, C., Rabitti, F., and Thanos, C. (1991). Conceptual modeling of multimedia documents. *IEEE Computer*, 24(10):23–30.

Mills, T., Moody, K., and Rodden., K. (1997). Cobra: a new approach to IR system design. In *Proceedings of RIAO'97*, pages 425–449.

Mizzaro, S. (1998). How many relevances in information retrieval? *Interacting With Computers*, 10(3):305–322. In press.

Turtle, H. and Croft, W. (1992). A comparison of text retrieval models. *The computer journal*, 35(3):279–290.

van Rijsbergen, C. (1979). *Information retrieval.* Butterworths, London, 2nd edition.

Vasanthakumar, S., Callan, J., and Croft, W. (1996). Integrating INQUERY with an RDBMS to support text retrieval. *Bulletin of the technical committee on data engineering*, 19(1):24–34.

Wong, S. and Yao, Y. (1995). On modeling information retrieval with probabilistic inference. *ACM Transactions on Information Systems*, 13(1):38–68.

Zobel, J. and Moffat, A. (1998). Exploring the similarity space. *SIGIR Forum*, 32(1).

Arjen P. de Vries is a PhD student at the Centre for Telematics and Information Technology (CTIT) at the University of Twente. Since 1995, Arjen does a multidisciplinary research project between the DOLLS research group of the department of computer science and the department of ergonomics. He is especially interested in the new requirements on the design of database systems to support content-based retrieval in multimedia digital libraries.

Annita N. Wilschut has recently started a job as software architect at the Dutch revenue services. Before, she has been a researcher in the DOLLS research group at the computer science department of the University of Twente since 1987. Her research mainly focuses on architectural aspects of database management systems. She has been involved in the MAGNUM project studying Object-Oriented database technology in the context of GIS-applications. Before MAGNUM, she worked for several years on the PRISMA parallel relational DBMS.