Moa and the Multi-model Architecture: A New Perspective on NF^2

M. van Keulen[1], J. Vonk[1], A.P. de Vries[2], J. Flokstra[1], and H.E. Blok[1]

[1] Center for Telematics and Information Technology (CTIT)
University of Twente,
Enschede, The Netherlands
{keulen,vonk,flokstra,blok}@eemcs.utwente.nl
[2] Centrum voor Wiskunde en Informatica,
Amsterdam, The Netherlands
arjen@cwi.nl

Abstract. Advanced non-traditional application domains such as geographic information systems and digital library systems demand advanced data management support. In an effort to cope with this demand, we present the concept of a novel multi-model DBMS architecture which provides evaluation of queries on complexly structured data without sacrificing efficiency. A vital role in this architecture is played by the Moa language featuring a nested relational data model based on XNF^2, in which we placed renewed interest. Furthermore, extensibility in Moa avoids optimization obstacles due to black-box treatment of ADTs. The combination of a mapping of queries on complexly structured data to an efficient physical algebra expression via a nested relational algebra, extensibility open to optimization, and the consequently better integration of domain-specific algorithms, makes that the Moa system can efficiently and effectively handle complex queries from non-traditional application domains.

1 Introduction

Advanced non-traditional applications, such as digital library systems (DL) and geographic information systems (GIS), place high demands on their data management components. Data in these areas is intrinsically complex and voluminous in nature and queries are computationally intensive. Researchers have sought to cope with these demands in different directions. In section 2, we explore these directions focussing on data model and DBMS architecture as a motivation for our multi-model DBMS architecture, as well as the particular role the logical algebra Moa plays in this architecture.

The multi-model DBMS architecture consists of three layers each supporting a different data model. In this way, the top conceptual layer provides a data model supporting complexly structured data, while in the logical and physical layers, a query on complexly structured data is gradually transformed to efficient storage-level operations. To be able to bridge the gap between a conceptual-level data model (e.g., a hierarchical or object-oriented data model) and a storage-level data model, we place renewed interest in Non First Normal Form (NF^2) data models, which were popular in the 80s, but proved difficult to support efficiently with then available technology. The Moa data model, used in

V. Mařík et al. (Eds.): DEXA 2003, LNCS 2736, pp. 67–76, 2003.

showed that a pure relational schema performed better in most cases than a schema using OR features such as set-valued attributes.

Furthermore, encapsulation of data and operations inside objects or ADTs affect query evaluation: optimization by the DBMS becomes infeasible, and query processing too often results in object-at-a-time evaluation. Predator's E-ADT concept [16] improves upon this by implementing an optimization interface to facilitate optimization of a query plan using its own algebraic operations. The E-ADT approach adheres to the *open implementation* approach, known from the software engineering field [14]. Instead of building an ADT as a black box, it provides a meta-interface (see Figure 1) allowing a client of the ADT to make certain performance choices. A typical example is a program's advice to the operating system to adjust its caching strategy to a (sequential) memory access pattern.

Multi-model DBMS architecture. To be able to deal with the trade-off between complex data model and performance, [22] introduces the multi-model DBMS architecture with different data models on different layers (see Figure 2) as opposed to ordinary relational systems, which use the relational model throughout the DBMS architecture. The conceptual layer typically has an OO data model or a hierarchical semi-structured one (e.g., XML). We choose other, 'simpler' data models for the logical and physical layers. Obviously, this comes at a cost, namely additional mappings between layers, that map a query expression from one language to another. A typical choice for a data model and algebra on the physical level, is one close to the machine, for example, relational algebra, or, what we have used in some cases, the binary relational data model of main-memory DBMS MonetDB [3].

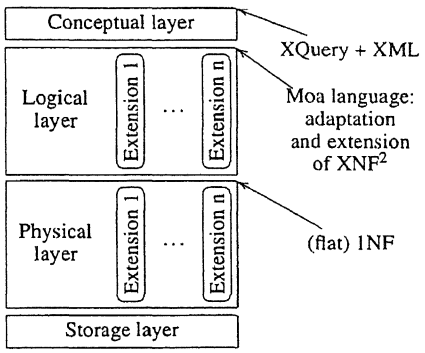

Fig. 2. The multi-model DBMS architecture and our choice for data models.

The XNF^2-data model [15] is very suitable as intermediary data model for bridging the gap (logical layer) between a complex data model and a simple relational one. It handles complex data structures as nested relations, but still comes with an algebra that is not much more complex than an ordinary relational one. The idea of a DBMS based on XNF^2 [9] lost interest when it appeared too difficult to build one that performed well. In the sequel, we show our adaptation of the XNF^2 data model and its effective use in our multi-model DBMS prototype, called Moa.

Summarizing, the Moa DBMS prototype with its multi-model architecture has the potential of better meeting the demands of advanced application domains. By using different data models on different layers, it is possible to provide a complex data model at the top and still be able to evaluate queries efficiently. We achieve the latter through several provisions: (1) by utilizing an XNF^2-based algebra as an intermediary, queries on complexly structured data are gradually and effectively translated to efficient storage-level operations, (2) extensibility at all layers allows to better integrate domain-specific algorithms into the DBMS, thus improving the performance of domain-specific opera-

the logical layer of the architecture, is an adaptation and extension of (X)NF^2 for which we developed an approach towards efficient query evaluation. Another key feature of our approach is extensibility on all layers to be able to integrate domain-specific algorithms such that certain optimization obstacles concerning ADTs are avoided.

The strength of the Moa system lies in the combination of architecture, data models, the mappings between them, and the way extensibility is handled. This paper provides an overview of the entire concept necessarily leaving out much detail and focussing on architecture and data models. The more interested reader is referred to [21].

In Section 3, we present Moa's approach to query processing. The Moa language is illustrated by means of an example in Section 4. The ideas behind Moa have been validated and fine-tuned in various advanced application domains, such as GIS and multi-media retrieval, an overview of which is given in Section 5. Finally, we present our conclusions and future work in Section 6.

2 Motivation and Related Research

In the past decade, the following trade-off particularly eluded researchers when trying to make DBMS technology better suitable to non-traditional applications. On the one hand, it concerns *data model expressiveness*. The suitability of a DBMS for an application is closely related to the expressiveness of its data model. The data model of the conceptual level should fit the universe of discourse, since end-users have to understand this model of the real world in order to formulate their queries. To support the inherently complexly structure data of many application areas, advanced data models were proposed, such as the object-oriented and object-relational data models. On the other hand, *performance* is expected from the DBMS, which typically means that it should be able to effectively optimize queries. The more complex the data model, however, the harder it is to develop an effective optimizer, as research on OODBMSs clearly showed.

Garlic [8] is an example of a system that integrates special-purpose data servers using *object wrappers*. Unfortunately, even with its advanced optimizer that uses statistics from the wrappers, performing such processing outside the scope of the database system may cause serious performance degradation, see, e.g., [11]. *Full-fledged OODBMSs* do not meet demands either. Often data independence is not handled well: the application class structure dictates physical layout of data and user-interfaces were non-declarative. As known from RDBMSs, data independence is essential for scalability and data distribution. But even with a declarative query language and the ability to convert OO item-oriented thinking into set-oriented query plans, which O_2 [1] both mastered, OODBMSs never became the success as anticipated.

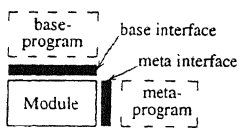

Fig. 1. Open implementation.

In the DBMS market today, the *object-relational (OR) data model* dominates claiming to be simple enough for query optimization, but expressive enough to handle advanced application areas. As Date and Darwen point out in [10], extensibility with user-defined data types does not require a new data model per se. However, as the Bucky benchmark has shown, room for improvement concerning performance exists [6]. Designed to evaluate especially the extra features of the OR data model, it

tions, and (3) extensibility in Moa is defined such that extensions are not black-boxes, but are open to the optimizer, hence, possible optimizations can be better exploited.

In the following section, we present Moa's approach to XNF^2-based query processing within the multi-model architecture.

3 Query Processing in Moa

As early as 1982, Schek and Pistor argued that the relational data model is inconvenient for a domain like information retrieval [15]. To overcome these shortcomings, they propose a NF^2 data model dropping the first normal form (1NF) requirement to allow non-atomic attribute domains such as sets of values. Many theoretical properties of the relational model also hold for NF^2. The main difference between NF^2 and eXtended NF^2 (XNF^2) data models such as that of the AIM DBMS [9], is that XNF^2 supports additional data types such as lists and allows for arbitrary nesting of type constructors.

The main part of the work on NF^2 concerns the *definition* of algebras, not their function: facilitate efficient query evaluation. With the latter, problems are encountered including inefficient nested-loop processing, data redundancy, restructuring overhead and the infamous *COUNT-bug* [18], since in the presence of empty subsets, unnest is not the inverse of nest. [17] has made a large contribution to the area of query optimization of nested relational algebras with, e.g., the introduction of a special *nestjoin*-operator.

Moa is an extension of XNF^2, where not only type constructors can be arbitrarily nested, but also new type constructors can be added. Moa additionally incorporates a solution to the mentioned NF^2 query processing problems. It does this by keeping an explicit structure definition in the form of type constructors connected to unnested (flat) data, and by having both nest/unnest operators as well as navigators, such as *map*. Furthermore, it deals with the COUNT-bug by explicitly generating counteracting operations where needed.

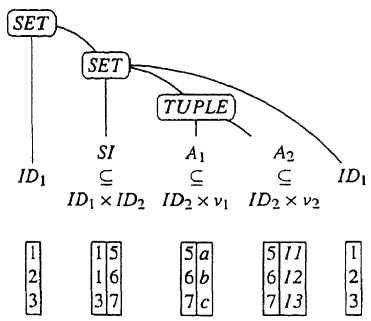

Fig. 3. Nested data and flat data.

Example (see Figure 3). Suppose, our database db is structured as a set of sets of n-tuples: $db \in \mathcal{PP}\, V_1 \times \ldots \times V_n$ where $\mathcal{P}$ is the powerset operator and V_i are domains of atomic values. Choosing subsets of V_i as $v_1 = \{a, b, c\}$ and $v_2 = \{11, 12, 13\}$, a concrete db can look like $db = \{\{(a, 11), (b, 12)\}, \{\}, \{(c, 13)\}\}$. Our generic mapping represents db as type constructors (rounded rectangles) connected to the following flat data:

$$ID_1 = \text{set of as many unique id's as there are subsets in the database.} \quad (1)$$

$$ID_2 = \text{set of as many unique id's as there are } n\text{-tuples in the database.} \quad (2)$$

$$SI = \text{subset index as a set of pairs} \subseteq ID_1 \times ID_2 \quad (3)$$

$$A_i = \text{columns as a set of pairs} \subseteq ID_2 \times v_i \ (i \in \{1, 2\}) \quad (4)$$

The figure furthermore illustrates the distinction between a *value* and an *identified value set* (or *ivs*). A column does not represent one atomic value, but a *set* of atomic values. The TUPLE structure constructed from A_1 and A_2 represents a set of tuples, called

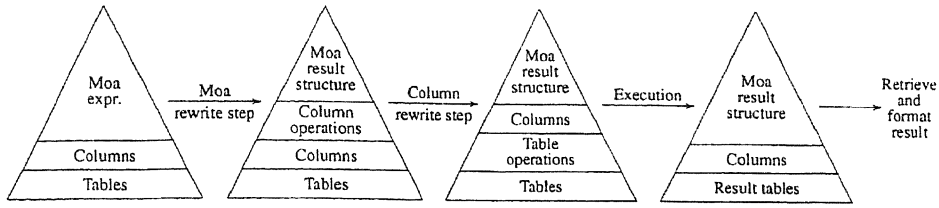

Fig. 4. Moa query evaluation steps.

an ivs, rather than one tuple value. The SET structure above it introduces a partitioning of this set of tuples according to SI, hence representing a set of sets, rather than one set, so it too is an ivs. The top-most SET doesn't introduce another partitioning, but only wraps things in a proper (nested) value and, therefore, is a value. All structures in Moa have a value and an ivs form.

Note that, based on SI alone, it is impossible to determine that one or more empty subsets exists. Part of solving the COUNT-bug is the representation of an empty subset as an occurrence in the third argument of SET ivs (ID_1) and no occurrence in the first (SI).

Each operator in our language is defined on structure and data level. For example, the *count* has the effect of converting a set-of-set-structured argument to a set-of-atomic-structured result (structure level), while at the same time generating a grouped count operation on the flat data connected to that result structure (data level). In other words, a query in Moa is translated into both a physical algebra expression on flat data, and an explicit conversion from argument to result structure.

This two-level approach to query evaluation is illustrated in Figure 4. The general form of a query is a *Moa expression* which uses columns from underlying tables[1] (the leftmost pyramid). In the first rewrite step, Moa operations are mapped onto their respective *column operations* and result structure (second pyramid). This step converts a query on a nested structure to operations on flat data, which, being a rewrite operation independent of data volume, causes only minimal overhead. In the third pyramid, the column operations have been translated to the *table operations* of the physical layer. The third step performs the actual execution of the table operations producing result tables connected to a Moa result structure. An example of the rewrite process is given at the end of the next section.

4 Moa Logical Language

As explained, the Moa logical language is based on XNF^2 and is described here by means of the well-known example of an organisation that has departments and employees that work in those departments. In terms of XNF^2, an organisation consists of a set of departments, which in turn consist of sets of employees.

[1] Until now, we have used relational tables as physical storage representation of a column, but this is theoretically not obligatory.

A specific instantiation of such an organisation is shown in the organisation diagram presented in Figure 5. It contains three departments, one with three, another with two employees, and the technical healtcare department has no employees (yet), as it is a newly founded department. Each department has a name and address attribute. Each employee has a name and a salary attribute.

In traditional 1NF relational terms, the department entity has a multi-valued attribute (i.e., the employees), which requires a transformation into a seperate entity and relationship if stored in a strict 1NF relational database. In the Moa system however, the nested structure of the schema can be preserved. Figure 6 shows the entire structure specification of the example organisation in the Moa logical language. At first glance, without knowing the Moa language, one sees that the organisation is modeled more naturally, since the nesting is preserved and not flattened as in 1NF.

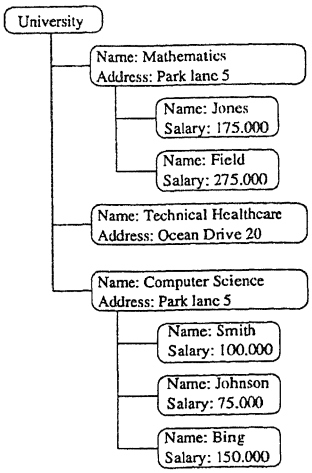

Fig. 5. Example organisation structure.

The structures Atomic, TUPLE, and SET in Figure 6 together form the kernel structures available in the Moa system. They constitute the NF^2 data model. Moa is open to the definition of additional structures, called extensions, that may support NF^2's arbitrary nesting. An example is the FV structure representing a feature vector intended for multi-media retrieval applications.

```
SET( |Department : _key|,
    TUPLE( Atomic( |Department : DName| ) : dname,
           Atomic( |Department : DAddress| ) : daddress,
           SET( |DepEmp : _match|,
                TUPLE( Atomic( |Employee : EName| ) : ename,
                       Atomic( |Employee : Salary| ) : salary
                     ) : Employee,
                |Department : _key|
                ) : Employees
           ) : Department
    ) : University
```

Fig. 6. Schema of University.

The syntax for type construction is a structure identifier, followed by a number of arguments and an optional label. The argument for Atomic is a column identifier which directly references the data stored in the underlying DBMS (cf., A_1 and A_2 in Figure 3). For example, |Department : DName| refers to the DName column of the Department table in an RDBMS. The label is used as a convenience mechanism to be able to reference the structure.

The **TUPLE** type constructor represents the theoretical notion of a product structure that consists of one or more Moa structures of any type. The **SET** type constructor represents a collection of Moa structures. The unrestricted nesting of type constructors makes it, in particular with this type constructor, possible to support the arbitrary nesting of the NF^2 data model. The arguments of SET include an index[2] mapping, the structure of the elements of this set, and the index of the set that encloses the specified set (cf., SI, TUPLE$\langle A_1, A_2 \rangle$, and ID_1 in Figure 3, respectively).

In the example above, the index mapping and enclosing set index are represented by |DepEmp : _match| and |Department : _key|, respectively. Those are special column identifiers disclosing database schema information. In this case, it maps primary keys of

[2] In relational terms, an index is the same as the primary key of a table.

departments to primary keys of employees. Thus, the |DepEmp : _match| column contains the relationship (CS,Smith), (CS,Johnson), (CS,Bing), (M,Jones), and (M,Field), where CS is 'Computer Science' and M is 'Mathematics'.

Several operations are defined on the kernel structures, of which the most important ones are listed below.

- **select** [m] (op); The select-operation is similar to the select-operation of relational algebra. The modifier m specifies the selection criterion and the operand op specifies the argument on which the selection should be applied.
- **attr** (op, l); The attr-operation is Moa's equivalent of projection, i.e., it evaluates to the attribute referenced by label l of its tuple-valued argument. In a modifier, it can be abbreviated by '%l'.
- **map** [m] (op); The map-operation is a navigational operator: it evaluates modifier m for each of the elements of operand op and collects the results in a SET structure.
- **join** [m_1, m_2] (op_1, op_2); The join-operation joins the two operands op_1 and op_2 based on equality of the modifiers m_1 and m_2, similar to the join-operation in relational algebra.
- **flatten** (op); The flatten-operation converts a set of subsets to one set by taking the union of all subsets, i.e., it removes one level of nesting.
- **count** (op); This is an example of an aggregate function and shown in the example of Figure 7 to illustrate that Moa correctly handles the COUNT-bug as explained in Section 3.

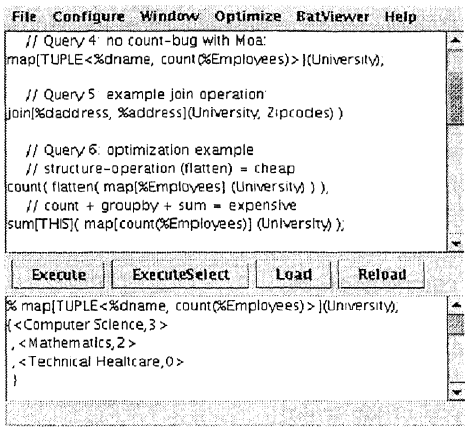

Fig. 7. Screendump of Moa System.

Note that the structure specification of Figure 6 is stored in the data dictionary of the system under the name "University" and can be used in queries directly.

Figure 7 shows a screen dump of the graphical user interface (GUI) of the Moa system. In the figure, the top-part of the GUI represents the input area and shows a number of example queries. The results of a query execution are presented in the bottom-part of the GUI. In this case, Query 4 has been executed. The resulting element $\langle$"Technical Healthcare", 0$\rangle$ illustrates the correct handling of the COUNT-bug, which would otherwise be missing from the result. Although without showing results, query 5 shows the use of the join-operation, and query 6 illustrates two equivalent expressions for the same query, hence offering an optimization opportunity. The underlying database used here is a relational DBMS.[3]

[3] The Moa system currently supports IBM DB2, PostgreSQL, MySQL, and MonetDB.

```
(query)  map[TUPLE( %dname, count(%employees) )](University)
(step 1) SET( C1, TUPLE( Atomic( |Department:DName| ), Atomic( C2 ) ) )
         C1 = |Department:_key|
         C2 = gbCount:(outerJoin(|Department:_key|,|EmpDep:_inverse|))
(step 2) SET( |Department:_key|,
            TUPLE( Atomic( |Department:DName| ), Atomic( |tmp1:_cnt| ) )
          )
         tmp0=outerjoin(Department.DNumber = Employee.DNO)
         tmp1=aggregate(count(tmp0.ENumber),tmp0.DNumber)
```

Fig. 8. Query rewrite steps for Query 4.

As an illustration of the rewrite steps of Figure 4, the rewrite process for example Query 4 is shown in Figure 8. The `map` and `attr` operations rewrite to the result structure representing a set of department tuples identified by C_1. The `count` operation rewrites to the column expression C_2. In the second step, column expression C_2 produces a column that takes the *_cnt* attribute from the result of the extended relational algebra (XRA) expression[4] `tmp1` (table operation).

Besides the operations and structures described in this section, others are available in the Moa system, e.g., the nest and unnest operations. However, due to space limitations, it is infeasible to present an exhaustive list. The reader is referred to [21].

5 Application Areas and Context

The validity of the Moa concept described so far, can best be seen by looking at the various projects in which Moa played a central role. The ideas for Moa and multi-model architecture originate from the Magnum-project [5,4]. In this project, a structurally object-oriented DBMS was developed for the purpose of efficiently integrating spatial and thematic data in a single data manager. *Decomposition* and *extensibility* were the key features of this project. In terms of the multi-model architecture, the Magnum system consisted of two layers: the main-memory DBMS MonetDB as physical layer and Moa as logical layer. This architecture could be extended in two ways. First, new base types could be defined in MonetDB, e.g., polygon, together with a large set of spatial operators on these primitive base types. Secondly, Moa's structural extensibility was used to support structures like polygonal maps and triangulations, next to the conventional tuple and set. Moa mapped these structures to MonetDB's binary data model, meaning that the structured data was decomposed in binary tables. Experiments showed that the Magnum system performed well on the Sequoia benchmark [19].

The experiences with the Moa/MonetDB combination, especially with the combination of base type and structural extensibility, sparked off new efforts. In the Mirror and AMIS projects, the ideas for an extensible DBMS architecture based on Moa and MonetDB were further developed in the realm of text and multi-media retrieval. Mirror concentrated on a generic multi-media retrieval framework based on belief networks. Experiments showed its feasibility for content-based retrieval for text, images, and music [12,22]. Since parallelisation and fragmentation in the physical layer is orthogonal to the logical layer, the architecture design seems to be better

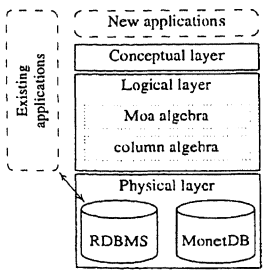

Fig. 9. SUMMER federated architecture.

[4] XRA [13] is a relational algebra extended with aggregates. The `aggregate` operation preforms the aggregate function in the first argument on the groups determined by the second argument.

prepared to scale up. The AMIS-project explored this idea by studying the optimization of top-N IR-queries in a fragmented context [2].

In all three projects described above, the integration of data and algorithms from non-traditional application domains in a single data manager was a central theme. Much thematic (tabular) data related to GIS or multi-media objects, however, resides in RDBMSs with existing applications running on them. Therefore, in the SUMMER-project [20], the multi-model architecture was taken one step further by using Moa as a kind of *data management middleware*, driving both MonetDB and 'normal' RDBMSs (see Figure 9) simultaneously. This allows the 'addition' of, for example, multi-media retrieval functionality to an existing federated information system. Also in SUMMER, we developed an XML-based conceptual layer providing a subset of XQuery [23]. This will make the DBMS suitable to be used in web-based environments, providing a more convenient way of managing large XML data volumes with, among others, integrated and efficient multi-media retrieval.

6 Conclusions and Future Work

In this paper, we presented the concepts behind the Moa system: the multi-model DBMS architecture and the Moa logical algebra which plays an important role therein. In order to support demanding applications like GIS or digital libraries, one needs an expressive conceptual data model supporting complexly structured data. Expressiveness is not the only requirement. Since the managed data is often voluminous and queries complex, performance is an important aspect as well. The multi-model architecture supports extensibility in all three layers thus enabling to integrate domain-specific algorithms in an effective way. Furthermore, the extensibility mechanism of the Moa language used in the logical layer, has been designed in such a way that optimization across extensions is possible. This alleviates the black-box ADT problem, which prohibits the optimizer, for example, to push projections and selections through ADT-operators.

To be able to bridge the gap between an expressive conceptual data model and an efficient simple physical data model, the nested relational intermediary proved effective. We placed renewed interest in XNF^2 algebra, adapted and extended it, and worked on new ways for efficient query evaluation. This resulted in the Moa language. We regard its role to be vital in the success of the multi-model architecture.

In several projects, a prototype DBMS evolved into what is now called the Moa system. The genericity, extensibility, and performance of the system were put to the test in real-life applications.

In current and future projects, the Moa system continues to be used as our experimentation platform, which imposes a ongoing demand for perfecting the extensibility and efficiency of the system. Moreover, we are exploring the realm of category theory in search for ways to fundamentally improve the Moa and column algebra. Further effort on distribution support is aimed at facilitating the construction of federated systems in more advanced ways. Finally, the focus of the CIRQUID-project [7] is the integration of information retrieval and databases to provide support for full text search in XQuery.

References

1. F. Bancilhon, C. Delobel, and P. Kanellakis, editors. *Building an Object-Oriented Database System : The Story of O_2*. Morgan Kaufmann Publishers, 1992.
2. H.E. Blok, A.P. de Vries, H.M. Blanken, and P.M.G. Apers. Experiences with IR Top N optimization in a main memory DBMS: Applying 'the database approach' in new domains. In *Procs of BNCOD 18*, LNCS 2097, pages 126–151, July 2001.
3. P.A. Boncz and M.L. Kersten. MIL primitives for querying a fragmented world. *VLDB Journal*, 8(2):101–119, 1999.
4. P.A. Boncz, W. Quak, and M.L. Kersten. Monet and its geographic extensions: A novel approach to high performance GIS processing. In *Procs of EDBT'96, Avignon, France*, LNCS 1057, pages 147–166, March 1996.
5. P.A. Boncz, A.N. Wilschut, and M.L. Kersten. Flattening an object algebra to provide performance. In *Procs of ICDE'98, Orlando, USA*, pages 568–577, February 1998.
6. M.J. Carey, D. J. DeWitt, J.F. Naughton, M. Asgarian, and et al. The BUCKY object-relational benchmark (experience paper). In *Procs of ACM SIGMOD'97, Tucson, Arizona, USA*, pages 135–146, May 1997.
7. The CIRQUID project website, 2003. `http://www.cs.utwente.nl/~cirquid`.
8. W.F. Cody, L.M. Haas, W. Niblack, M. Arya, and et al. Querying multimedia data from multiple repositories by content: the Garlic project. In *Procs of VDB 3, Lausanne, Switzerland*, IFIP 34, pages 17–35, March 1995.
9. P. Dadam, K. Küspert, F. Andersen, H.M. Blanken, and et al. A DBMS prototype to support extended NF^2 relations: An integrated view on flat tables and hierarchies. In *Procs of ACM SIGMOD'98, Washington, D.C., USA.*, pages 356–367, May 1986.
10. C.J. Date and H. Darwen. *Foundation for Object/Relational Databases: the Third Manifesto*. Addison-Wesley, 1998.
11. A.P. de Vries, B. Eberman, and D.E. Kovalcin. The design and implementation of an infrastructure for multimedia digital libraries. In *Procs of IDEAS'98, Cardiff, U.K.*, pages 103–120, July 1998.
12. A.P. de Vries, M.G.L.M. van Doorn, H.M. Blanken, and P.M.G. Apers. The Mirror MMDBMS architecture. In *Procs of VLDB'99, Edinburgh, U.K.*, pages 758–761, Sep. 1999.
13. Paul W. P. J. Grefen and Rolf A. de By. A multi-set extended relational algebra – a formal approach to a practical issue. In *Procs of ICDE'94, Houston, USA*, pages 80–88, Feb. 1994.
14. G. Kiczales, J. Lamping, C. Videira Lopes, C. Maeda, and et al. Open implementation design guidelines. In *Procs of ICSE'97, Boston, USA.*, pages 481–490, 1997.
15. H.-J. Schek and P. Pistor. Data structures for an integrated data base management and information retrieval system. In *Procs of VLDB'82, Mexico City*, pages 197–207, Sep. 1982.
16. P. Seshadri and M. Paskin. PREDATOR: An OR-DBMS with enhanced data types. In *Procs of ACM SIGMOD'97, Tucson, USA*, pages 568–571, May 1997.
17. H. Steenhagen. *Optimization of Object Query Languages*. PhD thesis, Uni. of Twente, 1995.
18. H.J. Steenhagen, P.M.G. Apers, and H.M. Blanken. Optimization of nested queries in a complex object model. In *Procs of EDBT'94, Cambridge, U.K.*, pages 337–350, 1994.
19. M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The Sequoia 2000 benchmark. In *Procs of ACM SIGMOD'93, Washington, D.C., USA.*, pages 2–11, May 1993.
20. The SUMMER project website, 2003. `http://www.cs.utwente.nl/~summer`.
21. M. van Keulen, J. Vonk, A.P. de Vries, J. Flokstra, and H.E. Blok. Moa: extensibility and efficiency in querying nested data. Technical Report 02–19, Centre for Telematics and Information Technology, University of Twente, The Netherlands, 2002.
22. A.P. de Vries. *Content and Multimedia Database Management Systems*. PhD thesis, University of Twente, 1999.
23. XQuery 1.0: An XML query language, 2003. `http://www.w3.org/TR/xquery/`.