In Proceedings of 10th International Conference on Database and Expert Systems Applications (DEXA 2000), Greenwich, UK, September 4-8, 2000. LNCS Vol. ????. pp. ???-???.

Design and Implementation of the OLOG Deductive Object-Oriented Database Management System

Xindong Li and Mengchi Liu

Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada S4S 0A2
Email: {li,mliu}@cs.uregina.ca

Abstract. OLOG is a novel deductive database system for advanced intelligent information system applications. It directly supports effective storage, efficient access and inference of large amount of persistent data with complex structures. It provides a SQL-like data definition language and data manipulation language, and a declarative rule-based query language. It combines the best of the deductive, object-oriented, and object-relational approaches in a uniform framework. This paper describes the design and implementation of the OLOG system.

1 Introduction

Deductive, object-oriented, and object-relational databases are three important extensions of the traditional relational database technology. Deductive databases stem from the integration of logic programming and relational databases. It offers representational and operational uniformity, reasoning capabilities, recursion, declarative querying, efficient secondary storage access, etc. However, deductive databases based on relational databases only allow flat relations and do not support data abstraction. As a result, more powerful deductive languages that support data with complex structures have been proposed, such as LDL [6], LPS [12], COL [1], Hilog [5], Relationlog [16], See [17] for an overview of some of these languages. Also, several deductive database systems that support data with complex structures have been developed, such as LDL [6], CORAL [23], and Relationlog [19].

Object-oriented concepts have evolved in three different disciplines: first in programming languages, then in artificial intelligence, and then in databases since the end of the 60's. Indeed, object orientation offers some of the most promising ways to meet the demands of many advanced database applications. The object-oriented philosophy creates a powerful synergy throughout the development life cycle by combining abstraction, encapsulation, and modularity. In the past decade, various object-oriented data models were developed [3, 7, 8, 13]. Also see an review in [25]. But there is a major problem with the object-oriented approach that is the lack of logical or mathematical foundations that, traditionally, has been playing an important role in database research. Such a foundation is essential for defining the semantics of databases and queries, for database design, and for query optimization.

Object-relational databases combine important object-oriented features with nested relational object databases. It extends relational databases with a richer type system including object orientation and adds constructs to relational query languages, such as SQL to deal with the added data types.

In the past decade, a lot of effort has been made to integrate deductive and object-oriented databases to gain the best of the two approaches, such as O-logic [22], revised O-logic [11], F-logic [10], IQL [2], LOGRES [4], ROL [15], Datalog^{++} [9], and DO2 [14]. As surveyed in [24], many of DOOD models are developed by extending and/or integrating the already existed deductive or object-oriented data models and they either are limited in object-oriented features or lack logical semantics. Few of them are fully implemented as persistent database management systems.

The objective of the OLOG system is to develop techniques for advanced intelligent information systems that directly support effective storage, efficient access and inference of large amount of data with complex structures. The OLOG language [18] is based on IQL [2] and O2 [7]. It overcomes the problem associated with IQL. It effectively integrates useful features in other deductive languages with a well-defined logical semantics.

The OLOG system has been developed in C++ mainly on a SUN SPARC-station running Solaris 2.5. It is implemented as a persistent database system that supports the OLOG query language, an SQL-like data definition language and data manipulation language. The implementation is based on ROL [20] and Relationlog [21].

In this paper, we describe the design and implementation of the OLOG system. In Section 2, we provide a brief overview of the OLOG language. In Section 3, we describe the OLOG system architecture. In Section 4, we explain the file system and storage management. In Section 5, we discuss OLOG kernel. In Section 6, we focus on the query evaluation issues. In Section 7, we conclude and comment on our future plans.

Due to space limitation, the discussion is terse and imcomplete.

2 Overview of OLOG Language

In OLOG, we can have not only classes but also relations with complex data structures such as nested tuples and sets. In this section, we present a brief summary of the OLOG language.

An OLOG database consists of four parts: type, schema, program, and fact. The OLOG language is a typed language for defining, manipulating, and querying OLOG databases.

The type part contains all type definitions. OLOG supports atomic data types including Char, String, Integer, and Real and class types for objects. Based on these data types, complex data types can be defined using tuple and set constructors. The following examples show how to define tuple and set types in OLOG:

```
create type Address (Street String, City String)

create type PersonSet {Person}
```

The schema part contains all schemas for both classes and relations. The schema provides the description of the database structures and are the basis for storage structures and query optimization strategies. Moreover, they are essential to the consistency of the database. The following examples show how to define classes and relations in OLOG:

```
create class Person (
    Name String,
    Gender Char,
    LivesIn Address,
    Spouse Person,
    Children PersonSet,
    Ancestors PersonSet
    )

create relation Family (
    Husband Person,
    Wife Person,
    Children PersonSet,
    primary key (Husband, Wife)
    )
```

Syntactically, class and relation definitions differ only in the key words. Semantically, a class definition is for a collection of objects whereas a relation definition is for a collection of relation tuples. Objects in OLOG has object identifiers whereas relation tuples do not. OLOG supports class hierarchy and multiple inheritance. The following examples show how to define subclasses in OLOG:

```
create class Employee isa Person (
    Salary Integer,
    Phone Integer
    )

create class Student isa Person (
    Age Integer
    )
```

The rule part is used to define intentional data that can be derived from the extensional data in fact part. Rules correspond to views of relational databases, but can be defined recursively. The following example show how to define rules to derive intensional information for class Person and relation Family:

```
create rule Person (
    Ancestors <P> :– Parents <P>;
    Ancestors <A> :– Parents <P>, P.Ancestors <A>;
    )
create rule Family (
    Family (Husband H, Wife W, Children <C>) :–
        Person H (Gender 'M', Spouse W, Children <C>);
    Family (Husband H, Wife: W, Children <C>) :–
        Person W (Gender 'F', Spouse H, Children <C>);
    )
```

The fact part contains relation tuples and objects. The following example shows how to create objects in OLOG, where tom and pam are object identifiers:

```
insert Employee tom(Name "Tom",
    Gender 'M',
    LiveIn ("Elm St", "Toronto"),
    Spouse pam,
    Parents {}
    )
insert Student pam(Name "Pam",
    Gender 'F',
    LiveIn ("MacPherson Avenue", "Regina"),
    Spouse tom,
    Parents {}
    )
```

The user can use OLOG queries to query OLOG databases. Queries in OLOG are represented using rules as well, the head of which specifies how to construct the result whereas the body specifies the query conditions. The following are several query examples:

```
query Result (TomsWife S) :– Person (Name "Tom", Spouse.Name S)
query Result (TomsAncs <S>) :– Person (Name "Tom", Ancestors.Name S)
query Result (AllTomsChildren <C>) :–
        Family (Husband.Name "Tom"; Children.Name C)
query Result (Person N, Children <C>) :–
        Family (Husband.Name N, Children.Name C)
```

Like ROL and Relationlog, OLOG supports type, schema, rule and other higher-order queries.

3 System Architecture

The OLOG system has been implemented as a single-user persistent database system in C++ under the Unix environment. The functional components of

OLOG can be roughly divided into three layers: interface, OLOG kernel, and storage management as shown in Figure 1.

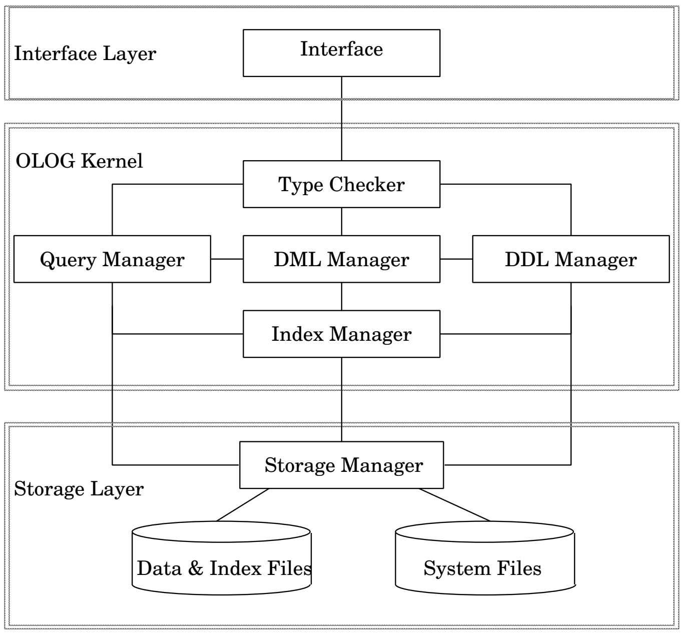

Fig. 1. The Architecture of OLOG System

The interface layer provides a friendly interface to users, takes user requests in different forms, and translates them into internal form to the OLOG kernel. Two kinds of interfaces are provided: textual interface and web interface.

The OLOG kernel consists of five components that are type checker, query manager, DML manager, DDL manager, and index manager. The type checker deals with all the type checking requests. The DDL manager is responsible for processing all DDL commands, maintaining system catalogs about domains, classes, relations and rules. The DML manager performs all the updates to the extensional relations and objects. The query manager handles user queries. The index manager is responsible for maintaining the index information, processing all index related operations from DDL and DML managers, and providing the index information for query manager.

The storage management layer is responsible for the management of the disk-based data structures on which databases are stored. It provides rapid access to objects and relations and meta information about them on the disk. The main job of the storage manager is to move the data between disk and memory as needed.

4 File system and storage management

The OLOG file system is built on top of the UNIX file system. Each database corresponds to two disk files: the data file and the index file, which are automatically created by OLOG system when started. The data file stores objects and relation tuples whereas the index file contains key indices to objects and relation tuples. In addition, OLOG maintains four system files for meta data management: database file which records the information of all databases in the system, relation file which records the information of all relations, class file which records the information of all classes, and rule file which records the information of all rules in each database. All data, index and various definition units in OLOG files are stored as byte string.

The internal structure of the OLOG data file supports the following characteristics:

(1) variable-length relation tuples and objects
(2) indices for objects and relation tuples
(3) reuse of deleted disk file space

The data file contains a file header and a dynamic storage area where variable-length blocks of data are stored. The file header is composed of the following fields:

(1) free space – stores a pointer to the first block of deallocated heap space
(2) end of file – stores a pointer to the address of the last byte in the file
(3) start of heap – stores a pointer to the start of the dynamic storage area

When a block is deallocated, the block is marked deleted and left in the file. The size of the file is determined by blocks allocated and will remain the same no matter how many blocks are deleted. The deleted blocks are maintained in a non-contiguous list. Each block in the list points to the next block in the list starting at a specified address. The "free space" pointer in the file header is used to store the file address of the block where the free space list starts.

The dynamic data area always starts out empty and grows when variable data blocks are allocated. The "free space" and "end of file" pointers in the OLOG file header are used to maintain the dynamic data area. The "end of file" pointer marks the end of the file and points to the location where the file can be extended during block allocation. OLOG always tries to allocate blocks from its free space list.

In OLOG, file addresses of block are used as pointers to the objects and relational tuples. File addresses are represented by 32-bit signed integer values. Therefore, the file can grow to a maximum size of 2.1 GB. File addresses have to be translated into exact memory addresses for the accesses in memory. The Storage Manager is in charge of address mapping that translates file addresses into memory addresses

To make efficient use of each block, OLOG adopts a technique similar to that used in Cache-Memory Mapping, called *Set Associative Mapping*. Each relation or class has its own buffer space for its own facts. The buffer space is composed of a lot of memory pages, holding some memory slots. This manager is responsible for managing the space occupied by various entities requested by the applications. This means if an object or a relational tuple is currently in main memory, then its memory address is returned to the application requesting it. Otherwise the Storage Manager employs a LRU algorithm to drop some old objects or relational tuples that have not been used for a long time, and then loads this one into memory. In case that the block to be dropped has been changed or deleted, the Buffer Manager physically changes or deletes the block stored on disk. The format of an object in its page buffer is similar to its disk format.

5 OLOG Kernel

In this section, we briefly describe Query, DML, DDL and Index managers in OLOG Kernel.

Query Manager The Query Manager is responsible for data retrieval and rule evaluation pertaining to the facts stored extensionally in the database and defined intensionally by rules. It firstly translates a query into its internal expressions, optimizes them based on the conditions the query processes and then uses different evaluation strategies such as matching, semi-naive bottom-up evaluation with rule ordering, and magic-set rule rewriting techniques to find the results. We discuss query evaluation in detail in Section 6.

DML Manager The DML Manager performs all the updates to the extensional relations and objects. When an object or relation tuple is to be inserted, DML first invokes DDL manager to query the schema definition and check whether it is well-typed with respect to its schema definition. Only well-typed objects and relation tuples can be inserted successfully. When an object or relation tuple is to be deleted, DML first checks whether it is referenced. If not, it then proceeds; otherwise, it rejects the request. Modification of an object or relation tuple is handled similarly. In OLOG, an update may imply a query. In this case, the DML manager will invoke the Query Manager to process the query before performing the update. After extensional relations or objects are updated, it will also request the Query Manager to propagate the updates to the materialized intensional relations and objects that are dependent on the updated ones.

DDL Manager The DDL Manager processes all OLOG DDL commands, maintains system files about domains, class and relation schemas, and rules. It also handles domain, schema, and rule queries and is thus called by DML and Query Managers. As intensional information may be materialized, the DDL manager is also responsible for dropping those materialized intensional information and the corresponding rules when requested.

Index Manager The Index Manager maintains the B-tree indices for the OLOG extensional and intensional relations and objects. and Hash indices for temporal ones. They are chosen because they are efficient and standard, and are used by most database systems. For objects, object identifiers are used as the key. For relations, primary keys are used as the key for B-tree indexing. B-tree indices reside both in memory and on disk and are used for defining and maintaining keys of each relation. They are stored in the index file of each database. Hash indices are applied only in memory for temporal relations, which could be the differential relations of semi-naive evaluation or temporarily created views.

6 Query Evaluation

The OLOG query manager is responsible for data retrieval and rule evaluation pertaining to the facts stored extensionally in the data file and defined intensionally by rules in the rule system file. It combines the query processing methods in ROL [20] and Relationlog [19] and supports the following evaluation strategies:

(1) matching
(2) semi-naive bottom-up evaluation with rule ordering
(3) magic-set rule rewriting technique

Matching In OLOG, the intensional information derived using rules is maintained by the system. It is kept in main memory if the memory space permits and it can be made persistent and maintained current by the system. If a user query concerns only the information in main memory, OLOG can find the query results immediately without any disk operations. If a user query concerns only the information stored on disk, OLOG uses indices if any to find the required information. For example, OLOG will use matching to evaluate the following query:

```
query Result (TomsWife S) :– Person (Name "Tom", Spouse.Name S)
```

Semi-naive bottom-up evaluation This strategy is used to evaluate rules and is the main one for the query processing in deductive database systems. If a user query concerns the intensional information that is not available in main memory or on disk, and the query is so general that no other methods can be used, then OLOG uses this strategy. For example, OLOG will uses semi-naive bottom-up strategy to evaluate the following query:

```
query Result (Person N, Children <C>) :–
      Family (Husband.Name N, Children.Name C)
```

Magic-set rule rewriting technique If a user query contains some constants, then we can only compute intensional information relevant to the query using the magic-set rule rewriting technique as it simulates in semi-naive bottom-up evaluation the pushing of selections that occurs in top-down approaches. Its performance can rival the efficiency of the top-down techniques. For example, OLOG will use magic-set rule rewriting technique to evaluate the following queries:

```
query Result (TomsAncs <S>) :– Person (Name "Tom", Ancestors.Name S)
query Result (AllTomsChildren <C>) :–
        Family (Husband.Name "Tom"; Children.Name C)
```

7 Conclusion

In this paper, we have discussed the design and implementation of the OLOG deductive object-oriented database system. The main novel feature of OLOG is its natural support for persistent classes and relations so that both pure object-oriented and object relational databases can be supported.

A complete implementation as described in this paper has been developed under the UNIX environment. The system will be available over the Internet after further testing, debugging and improving. More information about OLOG system can be found from the web site `http://www.cs.uregina.ca/~mliu/OLOG`.

We are currently extending OLOG into a full-fledged system. We are also developing the interfaces of SQL, OQL, and extended relational algebra and calculus on the top of OLOG in order to make it a useful tool in the database research and teaching.

Acknowledgment

The authors thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for research and equipment grants to M. Liu and the Graduate Studies and Research of the University of Regina for scholarships to X. Li.

References

1. S. Abiteboul and S. Grumbach. COL: A Logic-Based Language for Complex Objects. *ACM Trans. on Database Systems*, 16(1):1–30, 1991.
2. S. Abiteboul and P. C. Kanellakis. Object Identity as a Query Language. *Journal of ACM*, 45(5):798–842, 1998.
3. P. Butterworth, A. Otis, and J. Stein. The Gemstone Object Database Management System. *Communications of the ACM*, 34(10):64–77, 1991.
4. F. Cacace, S. Ceri, S. Crepi-Reghizzi, L. Tanca, and R. Zicari. Integrating Object-Oriented Data Modelling with a Rule-Based Programming Paradigm. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 225–236, 1990.
5. Q. Chen and W. Chu. HILOG: A High-Order Logic Programming Language for Non-1NF Deductive Databases. In W. Kim, J.M. Nicolas, and S. Nishio, editors, *Proceedings of the International Conference on Deductive and Object-Oriented Databases*, pages 431–452, Kyoto, Japan, 1989. North-Holland.
6. D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, and C. Zaniolo. The LDL System Prototype. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):76–90, 1990.

7. O. Deux and others. The Story of O_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, 1990.
8. D. H. Fishman, B. Beech, H. P. Cate, E. C. Chow, T. Connors, J. W. Davis, N. Derrett, C. G. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M. A. Neimat, T. A. Ryan, and M. C. Shan. Iris: An object-Oriented Database Management System. *ACM Trans. on Office Information Systems*, 5(1):48–69, 1987.
9. H. M. Jamil. Implementing Abstract Objects with Inheritance in Datalog^{neg}. In *Proceedings of the International Conference on Very Large Data Bases*, pages 46–65, Athens, Greece, 1997. Morgan Kaufmann Publishers, Inc.
10. M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of ACM*, 42(4):741–843, 1995.
11. M. Kifer and J. Wu. A Logic for Programming with Complex Objects. *J. Computer and System Sciences*, 47(1):77–120, 1993.
12. G. M. Kuper. Logic Programming with Sets. *J. Computer and System Sciences*, 41(1):44–64, 1990.
13. C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore System. *Communications of the ACM*, 34(10):50–63, 1991.
14. T. W. Ling and W. B. T. Lee. DO2: A Deductive Object-Oriented Database System. In *Proceedings of the 9th International Conference on Database and Expert System Applications (DEXA '98)*, pages 50–59, Vienna, Austria, 1998. Springer-Verlag LNCS 1460.
15. M. Liu. ROL: A Deductive Object Base Language. *Information Systems*, 21(5):431 – 457, 1996.
16. M. Liu. Relationlog: A Typed Extension to Datalog with Sets and Tuples. *Journal of Logic Programming*, 36(3):271–299, 1998.
17. M. Liu. Deductive Database Languages: Problems and Solutions. *ACM Computing Surveys*, 30(1):27–62, 1999.
18. M. Liu. OLOG: A Deductive Object Database Language. In *Proceedings of the Workshop on Next Generation Information Technologies and Systems (NGITS '99)*, pages 120–137, Zikhron-Yaakov, Israel, July 5-7 1999. Springer-Verlag LNCS 1649.
19. M. Liu. Query Processing in Relationlog. In *Proceedings of the 10th International Conference on Database and Expert System Applications (DEXA '99)*, pages 342–351, Florence, Italy, August 30-September 3 1999. Springer-Verlag LNCS 1677.
20. M. Liu. The Design and Implementation of the ROL System. *Journal of Intelligent Information System*, 15(2):41–68, 2000.
21. M. Liu and R. Shan. The Design and Implementation of the Relationlog Deductive Database System. In *Proceedings of the 9th International Workshop on Database and Expert System Applications (DEXA Workshop '98)*, pages 856–863, Vienna, Austria, August 24-28 1998. IEEE-CS Press.
22. D. Maier. A logic for objects. Technical Report CS/E-86-012, Oregon Graduate Institute, Beaverton, Oregon, 1986.
23. R. Ramakrishnan, D. Srivastava, S. Sudarshan, and P. Seshadri. The CORAL Deductive System. *The VLDB Journal*, 3(2):161–210, 1994.
24. P. R. F. Sampaio and N. W. Paton. Deductive Object-Oriented Database Systems: A Survey. In *Proceedings of the 3rd International Workshop on Rules in Database Systems (RIDS '92)*, pages 1–19. Springer-Verlag LNCS 1312, 1997.
25. V. Soloviev. An Overview of Three Commercial Object-Oriented Database Management Systems: ONTOS, ObjectStore, O2. *SIGMOD Record*, 21(1):93–104, 1992.