A Semantic DBMS Prototype*

Liu Chen and Ting Yu

State Key Lab of Software Engineering
School of Computer
Wuhan University, China

Abstract. The dominant database management systems such as Oracle and DB2 are based on the object-relational model, which grew out of the research in 1990s by extending the relational model with object-oriented features. They provide extended modeling power to users to build complex applications as it has shortened the distance from the conceptual model to the logical model. However there are three main problems with this approach. Firstly, it does not allow object migration so that the application development is unnecessarily complicated, time consuming and difficult to evolve. Secondly, it don't support inverse relationship so that the user has to manually define them and maintain their consistency. Thirdly, as the current implementations simply convert object-oriented features into various flat relations, the object manipulation and query processing are quite inefficient. Information Network Model is a novel conceptual model that can directly represent real-world organizational structures and different kinds of relationships and their inverse relationships between real-world entities and corresponding context-dependent properties so that the design and development of complex data applications is greatly simplified. Over the past three years, we have systematically designed and implemented this semantic database management system based on INM. In this paper, we describe the system.

1 Introduction

In early 1990s, various object-oriented models were proposed to support the direct modeling of real-world entities by means of object identity, complex values, typing, classification, property inheritance, etc. [1,2]. The object-relational model extends the relational model with these key object-oriented features [3,4] and has been adopted by the dominant database management systems such as Oracle and DB2. They do so by adding an object-relational layer on top of the relational engine so that it appears to be object-relational but the data is actually split and stored in various underlying relations. This approach provides users with extended modeling power to build complex applications. However, there are three main problems with this approach. Firstly, it does not allow object migration as an object must have a direct class and cannot change its membership during its life time. This limitation makes the application development unnecessarily

* This work is supported by National Natural Science Funds of China under grant No. 61202100.

complicated, time consuming and difficult to evolve. Secondly, it don't support the representation of the inverse relationship as in ODMG 3.0 [5]. The user has to manually define them and maintain their consistency. Thirdly, as the current implementations simply convert object-oriented features into various flat relations, the object manipulation and query processing are quite inefficient as many underlying relations are involved.

To solve these problems, Information Network Model (INM) has been proposed [6]. It supports two kinds of classes: object classes and role classes, two kinds of attributes: simple and composite, six kinds of relationships: normal, contain, context, role, role-based and composite. With these constructs, we can directly represent real-world organizational structures, various relationships and their inverse relationships between real-world entities and corresponding context-dependent properties. Indeed, INM is a conceptual model that is more expressive than existing ones such as UML [7] and ER [8]. It is quite easy to create the conceptual model of an application with INM. INM has three languages: data definition language (IDL), data manipulation language (IML) and query language (IQL). Based on the object class and relationship definitions in IDL, various role classes for the roles that entities play in the relationships can be generated automatically and objects can evolve to different role classes when needed. In INM, all information regarding a real-world entity is represented as a single object with a unique object identifier and one or more names. The object can belong to several classes to reflect the dynamic, many-faceted and evolutional aspects of the entity and objects are networked through various relationships and their inverse relationships. The object contains attribute/relationship names together with their values so that data in the object is self-describing. Furthermore, we use not only oids but also object names as relationship values so that the user can see what objects this object has relationships with, without jumping to linked objects to find their names.

Over the past three years, we have designed and implemented a semantic database management system based on INM to solve the problems with dominant database management systems. In this paper, we describe our implementation of the system. First, we give a brief overview of the INM modeling languages and then the architecture of INM database management system, which employs a conventional thin client/fat server software architecture. Also, we elaborate the core processing module, schema manager, instance manager, storage manager and query manager in which we use the advantage of object-oriented programming language and techniques, and at last the conclusion.

2 Overview of INM Languages

To introduce INM modeling, let us take university information modeling as an example. A university has a number of departments and locates in a city. Within a department, there are a number of faculty members and students, and each faculty member supervises some students. A person has an attribute birthdate and resides in a city, a country contains some provinces which in turn contain some cities. The following examples show several ways to use Information Definition Language (IDL), Information Manipulation Language (IML) and Information Query Language (IQL) for this application.

Example 1. First, we can use a nested relation so that all properties of Univ are nested as composite attributes, such as address, departments, faculty, students, etc. The following is the corresponding IDL statement, where * indicate the attribute is multi-valued.

```
create class Univ [
    @yearfounded: int,
    @category: {public,private},
    @address: [no:int,street:string,city:string,postcode:string],
    @departments*: [
        name:string,
        faculty*: [
            name:string,
            address: [no:int,street:string,city:string,postcode:string],
            birthdate: [day:int,month:int,year:int],
            students*: [name:string,
                address: [no:int,street:string,city:string,postcode:string],
                birthdate: [day:int,month:int,year:int]]]]];
```

Example 2. Different kinds of entities can be represented as INM objects of some classes, such as Univ, Country, Province, City and Person. In addition, faculty and student are the roles persons play in a department, so they are represented using role relationships. While playing the roles, these entities can have properties: a faculty member supervises students. Every relationship has a unique inverse, which can be a user-define one or a system generated default one if the inverse clause is not given. Also, we can represent cardinality constraints on the relationship and its inverse. For example, we can use (1:N) or * express one to many. The following IDL statements define classes and their various relationships.

```
create class Univ [normal address(N:1): City(inverse hasUniv),
    contain depts*: Department(inverse belongTo)];
create class Department [role Student*: Person(inverse studiesIn),
    role Faculty [role_based supervise*: Student(inverse supervisor)]*
            :Person(inverse worksIn)];
create class Person [@birthdate: [day:int,month:int,year:int],
    address(N:1): City(inverse hasPerson)];
create class Country [contain provinces*: Province(inverse belongTo) ];
create class Province [contain cities*: City(inverse belongTo)];
```

Example 3. The following IML statements show how to insert the facts that Department CS has a Student Jack and a Faculty Mike whose birthdate is 1960-1-1 and resides in Beijing, and supervises Student Jack.

```
insert Person Mike [@birthdate: [@day: 1,@month: 1,@year: 1960],address: Beijing];
insert Department CS [role Student: Jack, role Faculty: Mike [supervise: Jack]];
```

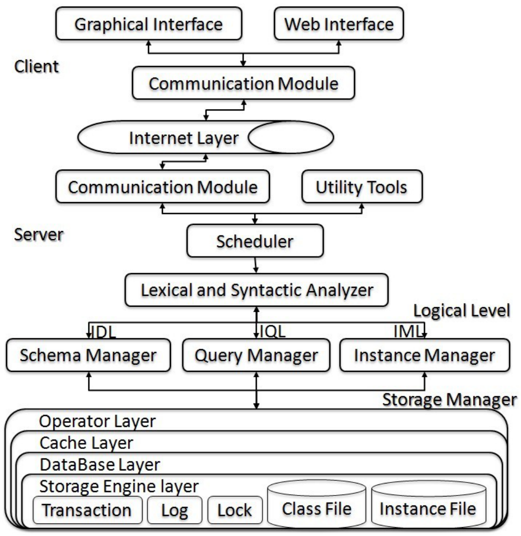

Fig. 1. Diagram of system architecture

Example 4. The following IQL statement finds and displays the birthyear of Mike and the student Mike supervises. Note that unlike SQL which mix querying and result constructing together, IQL strictly separates them so that queries are easier to write and understand.

```
query $x=Mike[birthdate/year:$y,//supervise:$s]
construct Professor $x[birthyear:$y,Student:$s];
```

In the query clause, logical variables $x matches the name Mike, $y and $s in the path expressions match the year 1960 and name Jack respectively. The construct clause specifies how to output the query result.

3 Overview of INM Database Management System

The INM database management system is a full-fledged database management system implemented in C++ on top of a PC running Linux. The lexical and syntactic analyzer is implemented using Flex 2.5.35 and Bison 2.4.1. The implementation employs a conventional thin client/fat server software architecture, as shown in Fig.1.

Client. The client of the database management system is organized into two layers: the *graphical/web interface* and the *communication module*. The graphical/web interface sends user requests to the server via the communication module and takes the query results from the server for proper display. The second layer is the communication module. It sends user requests from the first layer to the server for parsing and execution, and obtains query results from the server and then sends them back to the first layer.

Server. The server is organized into five layers. The first layer is the *communication module* which communicates with the communication modules of clients. It accepts user requests from clients, sends them to the scheduler, and ships the results back to these clients. Also, we provide on the server side a *Utility Tools* and available commands that for administrators to manage database, like batch file, statistics, clearing database, opening database listener, etc.

The second layer is the *scheduler*. It obtains multiple user requests from the communication module and the utility tools, schedules user requests, and ships them to the lexical and syntactical parser.

The third layer is the *lexical and syntactical analyzer* which performs lexical and syntactical analysis of user requests. It filters out invalid requests, transforms strings of valid requests into standard forms, and sends them to the schema manager, instance manager and query manager, which collectively referred to as the logical level.

The forth layer is the *logical level* which is in charge of class definition, object creation and modification and query processing. The *schema manager* validates operations and checks various integrity constrains. The *instance manager* matches objects with classes, and it is in charge of storing, modifying and deleting objects in the database. The *query manager* decides on what evaluation strategies to use according to the nature of the query, and it invokes the schema and the instance manager to handle queries of classes and objects respectively.

The last layer is the *storage manager*. It is responsible for the management of disk-based data structures and moves the data between disk and memory as needed. It is implemented in four layers, operator, cache, database and storage engine and each outer layer encapsulates the next inner layer respectively. It provides rapid access to classes, objects and other meta information about them on the disk.

3.1 Schema Manager

Schema manager is in charge of the logical processing of classes' definitions and updates and responsible for the interfaces to instance manager for objects' validation.

The structure of INM class consists of class identifier, class name, attributes and various relationships. The class identifier is generated automatically by the system as the key for each class. Attributes are prefixed with the symbol @ and there are two kinds of attributes in INM: simple and composite. In Example 1, attributes *yearfounded* and *category* are simple whereas the rest are composite. Also, attributes can either be single valued or multi-valued.

There are six kinds of relationships: role relationship, context relationship, role-based relationship, normal relationship, composite relationship and contain relationship. Schema manager insures that every relationship has a unique corresponding inverse relationship.

All classes are stored in the class primary file, in which the key of each record is the class identifier and the value is a byte array of the class's tree structure.Furthermore, there is a class name index whose key column stores the string

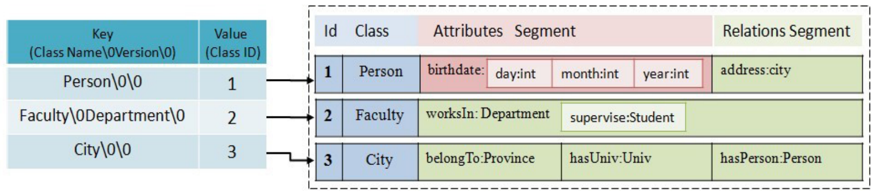

Fig. 2. A sample of the Class Name Index and the Class Primary File

which consists of the class name and the class version and value column stores the ID of the corresponding class.

Fig.2 displays several data records of the class primary file and a part of the class name index of the Example 2. Role class Department.Faculty induced from the corresponding role relationship in the class Department generates the context relationship worksIn and the role-based relationship supervise. In addition, there is no direct definition for the class City in Example 2, but for the inverse relationship mechanism, City has the relationship belongTo with Province and the relationship hasPerson with Person.

3.2 Instance Manager

Instance manager takes care of object generating, object modifying and object removing. It firstly executes validation check of the intermediary object structure and merges the objects confirmed to be the same entity. Three principles established to simplify object creation are partial instantiation, multi-inheritance and consistency maintaining.

The entire structure of an object contains object name, identifier, version, belonging classes, attributes and relationships. Only name, ID and a belonging class are necessary to construct basic objects. So objects can be partially instantiated in two ways: via instance creation command as (1) (3) in Fig.3, and via relationship target as (2) (4) in Fig.3. Segments of the same object are merged and stored as an entire object in the instance primary file.

Target of role relationship usually generates multi-inheritance for the target object already exists as an instance of an object class. The new generated role instance will be merged with the object instance referring to Example 3, as shown in Fig. 3(3).

Whenever a relationship is created with object generation, the inverse relationship with binding attributes and sub-relationships will be generated in the target object. The same thing happens during modifying and removing procedure to maintain the data consistency. Various indexes are created to raise efficiency of data access. The instance primary file is divided into the *object ID index*, the *object file* and the *large object file* to reduce memory fragments. In the object ID index, the key column stores the object ID and the value column keeps a marker bit marking in which file, the object file or the large object file, the object is stored. In both the object file and the large object file, the object ID is

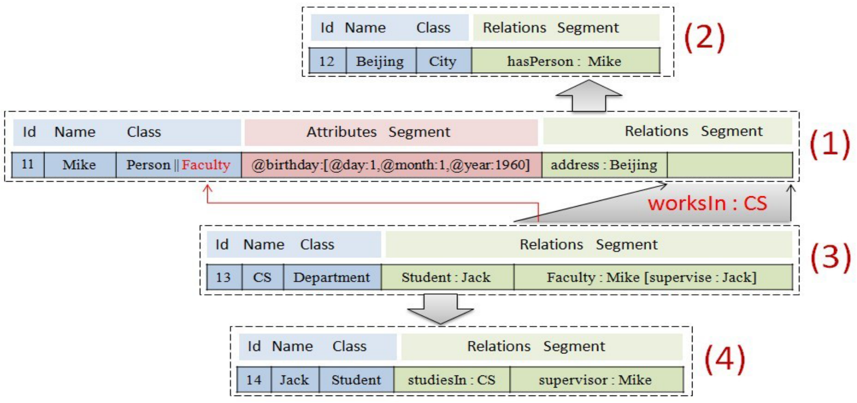
Id Name Class Relations Segment
12 Beijing City hasPerson : Mike
(2)
Id Name Class Attributes Segment Relations Segment
11 Mike Person || Faculty @birthday:[@day:1,@month:1,@year:1960] address : Beijing
(1)
worksIn : CS
Id Name Class Relations Segment
13 CS Department Student : Jack Faculty : Mike [supervise : Jack]
(3)
Id Name Class Relations Segment
14 Jack Student studiesIn : CS supervisor : Mike
(4)

Fig. 3. Partial instantiation and multi-inheritance of object in the instance file

stored as the key and the byte array of the object's tree structure as the value. Objects with the size smaller than 1 KB are stored in the object file, while others stored in the large object file. The page size of the large object file is 32 KB, 8 times larger than the object file. There are four other indexes: the *instance name index*, the *class identifier index*, the *attribute/relationship name index* and the *attribute value index*, which will be introduced in the query processing example in Section 3.4.

3.3 Storage Manager

Storage manager is responsible for the interfaces for the logical level. We divide storage module into four layers, the *operator layer*, the *cache layer*, the *database layer* and the *storage engine layer*, as shown in Fig.1.

Operator Layer. According to the needs of the logical level, the operator layer consists of two parts: schema operator and instance operator. The operator layer schedules the cache layer and the database layer and provides interfaces of data access for the logical level. For example, the instance operator encapsulates methods that getting an object by object name or oid and methods that removing an object from database, etc. The logical level cannot directly access the other layers, but the operator layer.

Cache Layer. Corresponding to the two parts of the operator layer, the cache layer also contains two aspects, and each aspect packs a cache table. The cache table is used to buffer the deserialized data.

Database Layer. Like the cache layer, the database layer also has two parts in accordance with the operator layer. It creates indexes, serializes data, and fetches data by means provided by the storage engine layer.

Storage Engine Layer. The storage engine layer is the Berkeley DB [9]. It provides the fundamental database management service, including the access to

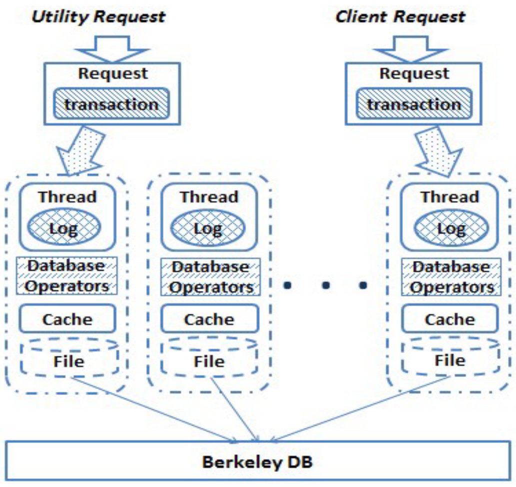

Fig. 4. Process diagram

the record in the class file and the instance file, transaction management, log, lock, buffer, etc.

As in Fig.4, when a process is created, firstly the database will be opened, in which the class file, the class index file, the instance file, and the instance index files are opened by calling methods of the storage engine level, and we call those data collectively *Global Data*. When the listener service is open, eight threads will be created and each thread possesses a pointer to the Global Data, a schema operator, a instance operator and a log file. When a thread receives any data from the client, a request and a transaction will be created, and the transaction is encapsulated in the request.

For example, when a new class is created and needs to be stored into the database, the logical level will invoke the class-inserting method in the schema operator, then the schema operator will pass the data to the schema cache level, and the data will be marked dirty. For the dirty records are not written to database instantly, the class name index is updated here to make sure this newly created class can be searched if following operations need. After the request finishes, the data marked dirty will be serialized and stored into the database through the storage engine level. In addition, the cache level will be cleared as the request's end, and if there occur any errors during the procedure, the transaction will roll back, otherwise, the transaction will be committed.

3.4 Query Manager

Query Manager includes query optimization, return pattern extraction, result construction and query output handler. Parser processing generates a query tree with variables/values bindings, which is used to store the variables, values as well as the hierarchy fetched from query expression.

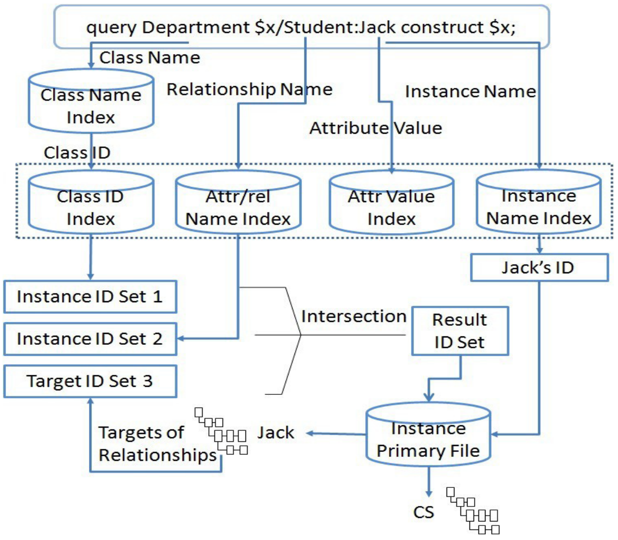

Fig. 5. Example of query processing

Result variables fetched from the construction expression of query command can scatter everywhere in the query tree. According to the position of result variables, appropriate querying strategy is chosen to make an effective querying.

After using querying strategy, values of each variable are binding as the sub-node of the variable in the query tree. The return pattern extraction extracts variables from construction expression in query tree to check if the result variables are in the query expression or not. If there is a result variable not in query expression, query output handler will commit the error information. Otherwise, the result construction will build a result graph to hold query result.

The Fig.5 shows a query processing that finding the department which has the student Jack. To optimize the query processing, we take advantage of indexes of class and instance as we have designed and maintained in schema manager and instance manager. From the left to the right in Fig.5, firstly we get the class identifier of the class Department via the class name index, then we get identifers of instances which belong to the class Department through the class identifier index, and we collects those instance identifiers in set 1. Secondly, we obtain identifiers of instances which have a relationship student via the attribute/relationship name index, and we put those instance identifiers in set 2. At last the value Jack, we firstly treat it as an attribute value for the attribute value index, the instances whose attributes have the value Jack will be returned, and here the result will be empty. We also treat Jack as a relationship target, that is an instance, via the instance name index and the instance primary file, easily we get the complete object Jack. For the inverse relationship mechanism, within the targets of relationships in object Jack, there must be one which is the result. So we get an instance identifer set 3. Finally, the intersection of the three sets will be the result object CS.

4 Conclusion

In this paper, we have described a prototype implementation of a novel semantic database management system based on Information Networking Model [6]. The complete implementation has been completed and the system is available from the svn address: mars.whu.edu.cn/inmproject.

The system has the following novel features. All information regarding a real-world entity is stored in a single object with a unique oid and one or more names. The object can belong to several classes to reflect the dynamic, many-faceted and evolutional aspects of the entity and objects are networked through various relationships and their inverse relationships. The object contains attribute/relationship names together with their values so that data in the object is self-describing, there is no limit on the number of attributes/relationships that objects can have, and space is not reserved for properties that are null. Furthermore, we use not only oids but also object names as relationship values so that the user can see what objects this object has relationships with, without jumping to linked objects to find their names.

Several applications based on the system have been developed such as the human resource management system for Wuhan University. We are also trying to improve the efficiency of object manipulation and query processing. We plan to extend the system to parallel distributed environment.

References

1. Kim, W.: Introduction to object-oriented databases. Computer Systems (1990)
2. Bancilhon, F., Delobel, C., Kanellakis, P.C.: Building an object-oriented database system, the story of o2 (1992)
3. Stonebraker, M., Moore, D.: Object Relational DBMSs: The Next Great Wave. Morgan Kaufmann Publishers Inc., San Francisco (1995)
4. Subramanian, M., Krishnamurthy, V.: Performance challenges in object-relational dbmss. IEEE Data Eng. Bull. 27–31 (1999)
5. Cattell, R., Barry, D., Berler, M., Eastman, J., Jordan, D., Russel, C., Schadow, O., Stanienda, T., Velez, F.: The Object Data Standard: ODMG 3.0. Morgan Kaufmann Publishers (2000)
6. Liu, M., Hu, J.: Information networking model. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 131–144. Springer, Heidelberg (2009)
7. Hamilton, M.: Software Development: Building Reliable Systems, 1st edn. Prentice-Hall (April 1999)
8. Chen, P.P.: The entity-relationship model - toward a unified view of data. ACM Trans. Database Syst., 9–36 (1976)
9. Oracle (Berkeley DB), `http://www.oracle.com/technology/products/berkeley-db/index.html`