Toward Resolving Inadequacies in Object-Oriented Data Models

Tok Wang LING and Pit Koon TEO
Department of Information Systems and Computer Science
National University of Singapore

Abstract

The object oriented (OO) paradigm suffers from several inadequacies which are widely recognised, eg. lack of a formal foundation, general disagreement in interpreting OO concepts, lack of a declarative query language, use of a navigational interface, inheritance conflicts in class hierarchies etc. In this paper, a representative list of these inadequacies is presented. Some proposals that have been made to resolve some of these inadequacies are reviewed. Then, we outline and justify an Entity Relationship based three level schema architecture which addresses several OO data modelling inadequacies, viz. the lack of support for the notion of relationship in the OO approach, the lack of support for external schemas, the inability to judge the quality of an OO schema and the lack of a reasonable approach to resolve inheritance conflicts in class hierarchies.

1 Introduction

Relational database management systems (RDBMSs) are suitable for handling applications utilising fixed format records. These applications include traditional business systems such as payroll, accounting, inventory control packages etc. Recent experience[25] has shown that applications requiring complex data structures and complex relationships among structures are not suitable for RDBMSs. Examples of such applications are CAD, knowledge based and multi-media systems. The base types of RDBMSs are not extensible and expressive enough to capture the application semantics. Moreover, the relational joins required to build complex structures from many base relations impose an unacceptable performance overhead.

Object-oriented (OO) DBMSs attempt to exploit the object-oriented paradigm to handle these complex applications. The advantages of the OO approach are well recognised [6,16,17,28,46] and include code reuse, enhanced modularity, extensibility, ease of maintenance among others. The tremendous interest in the OO approach has exposed several

inadequacies which have been widely discussed [9,14,28,38,46] eg. lack of a formal foundation, general disagreement in interpreting OO concepts, lack of a declarative query language, use of a navigational, as opposed to a declarative, interface, inheritance conflicts in class hierarchies etc.

This paper reviews, from a database perspective, a list of inadequacies in the OO paradigm and several solutions that have been proposed so far to resolve them. The list is not complete but it is representative of the major issues facing users of the OO paradigm. Some of these inadequacies cannot be solved by formulating a new model. For instance, the lack of consensus on OO concepts requires the concerted efforts of various standards group before it can be resolved. Other inadequacies such as lack of a standard declarative query language, multi-lingual support etc are still the subject of much research and generally accepted solutions to address these inadequacies may not appear within the next few years. We outline and justify an Entity Relationship (ER) [11] based three level schema architecure which addresses a set of OO data modelling issues, viz. (1) the lack of explicit support for the notion of relationship in the OO approach, (2) the lack of a general and flexible support for external schemas or views, (3) the inability to judge the quality of OO database schema design and (4) the lack of a reasonable approach to resolve inheritance conflicts in class hierarchies.

The list of inadequacies is classified into two parts. Section 2 discusses several widely recognised problems in the OO approach and some of the solutions that have been proposed so far to resolve them. Section 3 provides a list of OO data modelling problems which can be addressed by leveraging research efforts[11,31,32,33] in ER data modelling. Several examples from various OODBMSs, eg. ORION[6], IRIS[17], POSTGRES[39] and O2[16] are used to illustrate some of these problems. Comparisons with the relational model are made. Section 4 describes an ER-based architecture which addresses the data modelling problems mentioned in Section 3. Section 5 concludes the paper. The reader is assumed to be familiar with object-oriented concepts. See [18,28,40,46] for some useful references.

2 Widely Recognised Problems and Some Proposed Solutions

In this section, some of the widely recognised issues about the OO paradigm and several proposals that have been made to resolve them are discussed.

2.1 General Disagreement On OO Concepts

Several OO data models have been proposed [3,6,16,17,37,39] that offer somewhat different interpretations of OO concepts. Although some attempts have been made to forge a common agreement on OO concepts [5,14,46], they have so far been unsuccessful. There is also wide diversity in the implementation of these data models[20]. For example, Gemstone[37] adopts the object/message paradigm in which objects respond to messages sent to them, while Vbase[3] uses an abstract data type paradigm to encapsulate data and operations. IRIS uses a functional approach in which methods and attributes are modelled by mathematical functions while POSTGRES extends the relational model to support OO concepts. None of the proposed data models/implementations has emerged as the pre-eminent model/implementation.

Despite this diversity, a core set of OO concepts is beginning to emerge[28] that is common across these data models. The core concepts cover the notions of objects, attributes, methods, classes, class hierarchy, encapsulation and message passing. In time, the drive for standardisation and the efforts of various standards groups, eg. ANSI standards committee on OODB, Object Management Group etc, may forge a common Reference Model that includes the set of core concepts.

2.2 Navigational Model of Computation

Each object in an OODBMS is associated with a logical, non-reusable and unique object identifier (OID) [21]. The value of an attribute of an object may be an OID of another object, which in turn may reference other objects and so on, leading to a complex, nested structure. The use of explicit reference is reminiscent of the CODAYSL approach. This introduces a navigational component which causes several problems. First, for a navigational interface, access to data is hard-coded and therefore does not enjoy the benefits of a query optimiser[14]. Because of this, new information in the database (eg. new indexes) cannot be utilised automatically to improve retrieval efficiency. Second, the navigational approach does not preserve data independence any better than the hierarchical or network model. Application programs or methods that use navigational links are susceptible to changes that may arise from any schema change.

It has been suggested [28] that a navigational component may be unavoidable in an object-oriented system because of the complex, nested structure of objects. There are also claims[28] that there are some applications in AI and CAD that require navigation eg. traversal of graph structures in hypertext systems. In these applications, therefore, performance from navigational access is of primary importance while the need for

associative access through a query language is secondary [38]. We believe, as do others[27], that an OODBMS can be augmented with a declarative query language to complement the navigational access. For instance, through a declarative query facility, a set of object instances satisfying some search predicates can be retrieved. This can make use of the query optimiser. Query optimisation in the presence of methods is discussed later. Each object in the answer set can then be navigated for further processing.

2.3 Lack of a Standard Declarative Query Language

Unlike relational DBMSs which have adopted SQL as the de facto standard, no generally agreed upon standard declarative query language is available for OODBMSs yet. In OODBMSs that lack a declarative query language, methods need to be defined to handle queries. These systems face at least two problems. First, it is impossible to pre-empt all possible queries and provide methods for them. Second, consider the following query on the classic SUPPLIER (S), PART (P), PROJECT (J) database[15] :

Find all suppliers who supply at least one red part to more than one project.

This is not a trivial method to write. Further, this method must be defined at an appropriate level in the schema. For instance, it cannot be defined as a method for a supplier object. A more appropriate level is to either declare it as a class method [28] or as a method at a metaclass level. More complex queries can be found which are not easily translated into methods. To overcome these problems and provide ad hoc associative access to data, a query language is needed.

An OO query language is likely to be more complex than a relational query language because it must consider, among others, the semantics of methods and class hierarchies. We defer a discussion on the presence of methods in queries to the next section. A class hierarchy affects query processing in the sense that queries issued against a class C need to consider the subclasses of C. Depending on the desired semantics, the result of a query on the class C may be a set of instances of C or a set of instances of C and its subclasses. In the latter case, the result is a heterogeneous set of instances. This is different from the result of a relational query which always returns a homogeneous set of tuples [28]. The application which issues the query must be prepared to deal with this heterogeneous mix of instances[28]. An additional processing step, with associated performance implications, is needed to check the type of each instance.

Although recent proposals for query models [1,12,26] should provide a good basis for future developments in query languages, most developments on query languages for OODBMSs are still preliminary. Queries in ORION are restricted to only a class and its subclasses. In [28], the ORION query model was extended to support operations similar to relational join, projection, selection and set operations. O2 has an interpretive query language but the language lacks the declarative approach of SQL. POSTGRES supports POSTQUEL, which is a set oriented, declarative query language with its roots in QUEL. A preliminary proposal for an object SQL is reported in [7] for IRIS.

2.4 Issues in Use of Methods

There are two broad categories of methods: those that update databases (ie those with side effects) and those without side effects (ie only perform database retrievals, computations etc). Optimising queries in the presence of arbitrary methods is an open issue. It may be difficult, if at all possible, to optimise queries that use methods, because of the imperative nature of methods. Further, it may be undesirable to use methods with side-effects in queries. Some systems (eg. IRIS) only allow methods without side-effects to participate in queries. Systems like POSTGRES restrict methods to contain only data manipulation commands that can be optimised. Many other systems do not permit methods in queries.

Note that unlike attributes, it may not be possible to index methods to provide an alternative path for method retrieval and subsequent invocation. It has been suggested[46] that an index can be maintained on the (single) value returned by a method provided that the method does not go beyond the boundary of the object for which it is defined. Whenever there is any database state change that affects the result of the method, a trigger automatically maintains the consistency of the index. This proposal is likely to be inefficient and impractical. For instance, consider an index defined for method that computes a value that is dependent on the exchange rate between the US$ and some other currency. Whenever the rate moves, the index on that method must be maintained. In a volatile market, the index maintenance will be very significant.

The message passing mechanism that is used to trigger methods in OO systems presents some problems. First, message passing is binary. To use the message passing mechanism, it may be required to re-define an n-ary relationship among objects, eg. an SPJ relationship[15], in terms of binary relationships. This will result in loss of information. Second, the OID of the receiving object must be known before a message can be sent. The OID may sometimes not be readily available. For instance, consider the query:

Find the course-name for course-id = 'CS101'

The OID of the object holding the answer is unknown and must be found before a message can be sent to it. If an index is maintained on course-id, the OID can be retrieved from the index. Otherwise, all course objects in the database must be scanned before the answer is found. Note that unlike the relational approach where data retrieval using indexes is transparent, methods in OODBMSs that lack a query language/optimiser must explicitly use the indexes.

Of course, the use of methods still provides positive, tangible benefits. Methods are useful for defining object semantics. For example, when an employee is fired, a series of actions need to be taken, eg. deleting the employee information from all project files, inserting employee information into a history file and deleting the employee information from the employee file. A method 'fire-employee', incorporating the above sequence of events, conveys more meaning compared to the sequence of events itself. Further, some methods shield applications from implementation or policy changes. For instance, consider a tax computation method for an EMPLOYEE class. Whenever the tax computation policy changes, the method needs to be changed. However, applications that use this method need not be changed, provided the signature of the method remains unchanged.

2.5 Access Methods and Data Type Extensibility

A notable objective of providing type extensibility in OODBMSs is that new, specialised, types can be defined at schema definition time by using the data definition/manipulation languages of the DBMS, rather than only at database generation time[38]. One major problem in supporting type extensibility is the need to provide efficient access to instances of newly created types. Access methods provided by conventional DBMSs (eg. B-trees, hash tables etc) may not be suitable for these new, specialised types. Therefore, OODBMSs must provide access methods beyond those provided by conventional DBMSs. It has been proposed [42] that users define their own access methods to support new types. However, supporting these user-definable access methods is difficult[42]. To overcome this problem, we envisage that off-the-shelf reusable libraries of access methods may become readily available in future. These libraries can then be linked into the standard DBMS code to support user-defined types when needed.

2.6 Impedance Mismatch across Multi-Language Platforms

To access a database, a programming language can be embedded with database calls. An impedance mismatch arises because (1) the type systems of a programming language and a DBMS differ, and (2) database calls are declarative and operate on a set-at-a-time basis, while a programming language is imperative and are suited for handling record-at-a-time processing.

To overcome this problem, the notion of a database programming language (DBPL) has been proposed. From a database perspective, a DBPL can be obtained by adding functionalities to a database manipulation language (DML) of the database. Moving functions into databases has been shown to improve performance[14]. From a programming language perspective, a DBPL can be obtained by adding persistence and sharability to variables of the DBPL. For instance, GEMSTONE[37] extends SMALLTALK with support for persistent variables. It is believed[9] that these two approaches will eventually converge, although several difficulties need to be resolved[9], eg. query optimisation, semantics of class, constraints, and triggers etc.

Unfortunately, in an OODBMS supporting multiple languages, impedance mismatch will continue to exist for some of these languages[14]. It is impossible to construct an efficient OODBMS that is seamless across multiple languages.

2.7 Efficient Retrieval of Complex Objects

Many applications need to define and manipulate a set of objects as a single logical complex entity. Arbitrarily complex objects are built using rich data structures such as list, set, records and nested combinations of these. Most OODBMSs eg. O2, ORION etc. support complex objects. In some OODBMSs, eg. ORION, semantic relationships such as IS-PART-OF are assigned to inter-object references within complex objects. A composite object [24] in ORION is a complex object that models the IS-PART-OF (PART-SUBPART) relationship among objects. Orion also supports the concept of an existentially-dependent object, in which the existence of the object depends on the existence of its parent object. The deletion of an object triggers a cascading delete of all objects that are existentially dependent on the deleted object. This adds to the integrity features of the ORION data model. While the use of complex objects has important semantic value, the efficient retrieval of a complex object (and its components) is still an open issue.

Several techniques eg. clustering[27] and indexing[8] have been proposed to improve the performance of complex object retrieval. The suitability of each of these techniques

depends on whether the navigational or query-based approach is used. Clustering is suitable when an object is navigated using inter-object references. In clustering, components of a complex object are stored together on a physical transfer unit (eg. page) and hence can be retrieved efficiently. However, any clustering of objects is optimal for one type of access to the objects, but sub-optimal for most other types of access [27]. In general, it is left to users to specify a preferred clustering strategy[27].

Indexing is used extensively in RDBMSs to support efficient query processing. The idea of using indexes has been extended to OODBMSs. In [8], the notions of a class hierarchy index and nested attribute index were discussed. A class hierarchy index is defined on an attribute of a class, and instances that are indexed belong to the class and its subclasses, if any. Class hierarchy indexes have been implemented in ORION[27,28]. A nested attribute is an attribute of a nested component object of a complex object. Queries on a complex object can be predicated on a nested attribute. Therefore, by defining an index on the nested attribute, the queries can be more efficiently supported. See [28] for an example of nested attribute indexing.

2.8 Object Identity

There has been considerable debate [14,38] on the relative merits of supporting OIDs and using user-defined keys (as in RDBMSs). A somewhat ambivalent position was adopted in [14], which proposed that OIDs should be assigned only if keys are not available. We believe that OIDs are unnecessary and undesirable for the following reasons. First, all keys, eg. social security number, employee number etc, are actually user-created. Indeed, all attributes (key or otherwise) are artificially created by users. Therefore, a key can always be artificially created by the user for a relation that does not possess one. This is not dissimilar to the assignment of OIDs, except that OIDs are automatically system generated and assigned. Second, keys are more natural and human readable compared to OIDs, which are implementation specific (eg. pointer-based OIDs). Key-based data models are also more declarative [45]. Third, while values of keys can change because of changing conditions[38], such changes represent a conscious effort on the part of the user and can be done in a controlled environment. Finally, consider a weak entity[31], whose existence depends on the existence of another entity. The semantics of a weak entity requires that it should be accessed in conjunction with the entity that determines its existence. However, the use of OIDs allows the weak entity to be directly accessed. This weakens the semantics of weak entities.

2.9 Should attributes be directly accessible?

There is some debate [14,38] on whether attributes of an object should be directly accessible. It was advocated in [38] that access to an object's attribute values should be through the object's (public) methods. These methods constitute the external interface of the object class. This approach shields applications from changes in the implementation of attributes and provides data independence. A different position was adopted in [14], which advocated that, subject to a separate authorisation scheme, attributes of objects should be directly accessible. This is because a query language/optimiser needs to access the object's values directly. In [2], the distinction between attributes and methods are blurred, ie attributes can be modelled using methods. In our view, representations of most key attributes eg. part number, employee number, etc do not change once they are implemented. It appears trivial and redundant to generate public access methods (eg. get/set) to these key attributes. These attributes should be made public and directly accessible. In run-time, accesses to these attributes would not generate additional function call overheads.

3 OO Data Modelling Issues

This section discusses some OO data modelling issues which can be resolved by applying concepts and techniques from ER data modelling[11,31,32,33].

3.1 No Formal Foundation for OO Paradigm

Relational databases have the relational model[13], which has a mathematical basis in first order logic (FOL). Normalisation and data dependency theories can be applied to a relational database schema to improve the quality of a relational schema design. No equivalent theories (eg. functional and multi-valued dependency theories, normalisation etc) are available for OO database design, and therefore it is difficult to judge whether an OO schema is 'good'. To address this problem, we proposed in [36] that an OO database schema can be treated as a view of a normalised conceptual schema[31] based on the ER model. An ER-based normalised conceptual schema satisfies the 'goodness' criteria for database schema as laid down in [31], and is therefore a good candidate from which to generate OO external schemas (or views). The theoretical foundation of the conceptual schema is provided by ER, normalisation and dependency theories. An outline of our approach is given in Section 4.

Several related efforts have been made to provide a logic-based formalism for the core OO concepts by extending first order logic with the semantics of object identity, inheritance, nested objects and declarative functions [1,22,23]. It has also been suggested that the core OO model is really one type of an extended relational model[28], and therefore the foundation of the relational model carries over naturally to the core OO model. Some models

(eg. O2) have also been given a formal treatment [12,30]. The proposed formalisms are more complex than the relational model and do not appear to enjoy the kind of general acceptance that the relational model has.

3.2 Lack of Support for Explicit Relationships

Most OO data models (eg. O2, ORION) use inter-object references (using OIDs) and the class hierarchy to support relationships among objects. Inter-object references provide only implicit binary relationships between objects. Using this approach, the modelling of m-n, n-ary and recursive relationships is problematic and introduces problems similar to those faced by hierarchical and network models. The class hierarchy allows object classes that are related by the ISA relationship to be organised into a hierarchy. However, special relationship types[31] such as UNION, INTERSECTION, DECOMPOSE etc are not supported. In the following subsections, several problems that may arise from the lack of explicit representation of relationships among object classes are explored. We discuss in Section 4 how the ER approach can be used to resolve the problems mentioned in this section.

3.2.1 Everything as Objects?

Smalltalk[18] has successfully demonstrated the usefulness of a consistent treatment of everything as objects in a programming environment. In our view, it may be less useful to treat everything as objects in a database environment. In database design, it is important, although it may not be easy in certain cases, to distinguish among attributes, entities and relationships. To treat everything as objects will only blur the differences among these notions. In contrast, the ER model represents attributes, entities and different types of relationships (m-n, n-ary, recursive, weak etc) among entities explicitly and therefore captures the semantics of a database application more precisely than the OO approach, as will be discussed in Section 4.

3.2.2 Nested relations

A nested relational model relaxes the constraint that relations should be in first normal form. A relation can then contain another relation as the value of a field of a tuple. Consider the nested relation DEPARTMENT(D#, Dname, EMPLOYEE), in which the EMPLOYEE attribute is itself a relation. Such a nested relation imposes a strictly hierarchical structure which does not facilitate symmetric queries. For example, given a D# value, it is easy to find the employee information associated with it. The converse is difficult without an auxiliary access path (index) to support it. In OODBMSs, there are at least two

approaches to support the DEPARTMENT nested relation. Each of these approaches has its advantages and limitations.

The first approach treats the EMPLOYEE attribute in DEPARTMENT as a multivalued attribute that has, as its value, a list of object identifiers that identifies the employees working in the department. This approach is adopted in O2 and allows object sharing. For example, if an employee works in multiple departments, the employee's OID will appear in the list of employees of each of these departments. Using this approach, however, it is difficult to handle symmetric queries. Note that some object-oriented languages such as C++ [44] do not have direct language support for multi-valued attributes.

The second approach is adopted in POSTGRES. It allows the value of an attribute in a relation to be a relational query. It assumes that there exists two physical tables, viz an EMPLOYEE table with a structure given by EMPLOYEE(E#,Ename, Salary, Deptno) and a DEPARTMENT table with a structure given by DEPARTMENT(D#, Dname, EMPLOYEE). The EMPLOYEE attribute in DEPARTMENT can be defined to hold a query such as:

*Select * from EMPLOYEE*
Where EMPLOYEE.Deptno = D#

This approach is analogous to defining a view of the EMPLOYEE table for each record in the DEPARTMENT table. This view ensures that when a new employee is inserted into the physical EMPLOYEE table, the new employee's information will automatically appear in the view defined by the EMPLOYEE attribute of the DEPARTMENT table. One problem with this approach is that updates on the EMPLOYEE attribute (view) of DEPARTMENT must be translated to updates on the base EMPLOYEE table. This may not always be possible in general and is similar to, if not more difficult than, the classic view update problem. Another problem is the storage of the above query. It would be redundant to store this query against each tuple in the DEPARTMENT table.

3.2.3 M-N, N-ary and Recursive Relationships

Consider the following schema adapted from [30]:

(ob1, <name: "john", spouse: ob2>)
(ob2, <name: "mary", spouse: ob1>)

Each obi is an OID. This example uses an inverse relationship reference (a redundant relationship) in the spouse field that can introduce maintenance problems when there are

changes in the 'spouse' field, eg. if "john" and "mary" are divorced, then references in the 'spouse' field of both "john" and "mary" must be updated. This is a contradiction to the easy maintainability objective of the OO paradigm. To resolve this problem, the relationship between "john" and "mary" should be made explicit. As another example, consider the SUPPLIER (S), PART (P), SUPPLIES (SP) database [15], in which S and P are related by an m-n relationship type SP. Using the OO approach, each supplier object is associated, through inter-object references, with a set of part objects and their respective quantities. Such a structure does not facilitate symmetric queries. To circumvent this deficiency, each part object is then associated with a set of supplier objects and their respective quantities. This introduces redundancy which may lead to updating anomalies. The problems are similar to those of the hierarchical model. These problems are amplified when n-ary (n>2) relationships are considered. There is no feasible solution for modelling an n-ary relationship using inter-object binary references in the OO paradigm.

It has been suggested [10] that new 'relationship' object types (classes) must be created to represent ternary (and higher degree) relationships. Objects participating in these relationships are then linked through the 'relationship' object types. Such structures are similar to the modelling of relationships in the relational model, which uses key values instead of pointers. This approach also shows that the notion of relationship is needed.

Consider a recursive relationship [31] such as a course and its pre-requisites. One way to represent this relationship in ORION and O2 is to define a set valued attribute called pre-requisites in a course class; members of the set are pre-requisite courses. Again, this representation does not facilitate symmetric queries. Further, to compute the transitive closure of the course relation, a programmer has to either write a method or rely on query language support, if any. Programmer written methods need to be certified for correctness and is therefore less reliable compared to the use of a query facility. Some OODBMSs provide syntactic enhancements to their query languages to support the computation of the transitive closure of a relation. For instance, in POSTGRES, an '*' can be appended to a query that retrieves instances of a relation. The query then executes repetitively until the closure of the relation is obtained.

The examples of Section 3.2 strongly show that the lack of a semantic support for relationships among objects can cause many problems. The ER-based framework proposed in Section 4 supports the notion of relationship directly and hence does not face the same problems.

3.3 Lack of General View Support

Except for those OODBMSs that are based on the extended relational model (eg. POSTGRES), most OODBMSs do not fit into the 3-level schema architecture framework as spelled out in the ANSI/X3/ SPARC proposal[4] for DBMSs. Users of most OODBMSs are often presented with a large-grained conceptual schema, with little or no facility for defining views or external schemas.

Several proposals have recently been made to incorporate views in OODBMSs, but none of the proposals provides the same generality and flexibility of a declarative relational view mechanism. The focus of [19,41] was on defining multiple interfaces (views) to an object class. Each interface or view defines a set of methods and different views of an object class may share methods. These proposals do not consider views that are based on joins of multiple object classes. Neither do they consider how views can be defined that select a subset of objects from a specific class, based on some search predicate. In [2,29], a query based view mechanism was used to derive subclasses from superclasses. [2] treats views as queries and uses the view mechanism to define virtual classes, which are structured into inheritance lattice. The behavioural aspects of the views defined are generated automatically by the system. Such views are not updatable. In [29], updates apply only to non-recursive views that are based on a join of the primary keys of the base tables. Both [2,29] cannot handle other kinds of relationships eg. m-n, n-ary relationships etc. In [36], we propose an ER-based 3-level schema architecture that treats schemas based on any OO data model as external schemas (or views). While other approaches [2,29] use a query language to derive views, the approach of [36] uses mapping rules to generate external schemas. An outline of the approach in [36] is given in Section 4.

One of the problems with views is the need to support updatable views. It was proposed [38] that OODBMSs can be built to support updatable views by using OIDs. The OIDs of the objects that are used to generate elements of a view are tracked and used to generate OIDs for elements of the view. This generated OIDs can then be used to support view updates. An open issue is whether these OIDs should be transient or persistent. It seems redundant to generate OIDs for something as transient as elements of a view and may slow down view processing considerably. In [43], it was proposed that production rules can be used to propagate updates on views to base predicates. However, using production rules has several problems which is described in [34], eg. possibility of indefinite triggering, lack of theory for triggers, side effects of triggers etc.

3.4 Conflicts in Class Hierarchies with Multiple Inheritance

There can be property name conflicts between a class and its superclasses. In most systems, the property names in the class take precedence over similar property names in superclasses. Property name conflicts can also happen if two or more superclasses of a class have properties with the same name. In this case several proposals have been made for OODBMSs to resolve this conflict. First, the user can explicitly choose which property name to inherit. Second, the system can take a default, which usually is the first in the list of superclasses defined for the class. Third, the system can reject this kind of situation at class definition time. O2 and ORION take the first and second approach respectively while POSTGRES takes the third approach. These techniques are not satisfactory because they operate at a syntactic, rather than a semantic, level. The reasons why the conflicts occur are never explored.

In [35], we provide an algorithm which resolves inheritance conflicts in class hierarchies. The approach of [35] goes beyond most conflict resolution techniques of OODBMSs by examining the reasons why conflicts occur.

4 A Normal Form OOER Model

In this section, a three-level schema architecture is outlined which preserves the advantages of the OO approach at the external schema level and simultaneously resolves, at the conceptual schema level, the OO data modelling problems that have been identified in Section 3, viz: (1) the lack of explicit support for the notion of relationship in the OO approach, (2) the lack of general and flexible support for external schemas or views, (3) the inability to differentiate good OO schema design from bad, and (4) the lack of a reasonable approach to resolve inheritance conflicts in class hierarchies.

The architectural framework comprises an external schema level, a conceptual schema level and an internal storage level (similar to the ANSI/SPARC/X3 proposal[4] for database systems). Any OO database schema (based on any of the proposed OO data models) is treated as an external schema which can be generated from a conceptual schema by applying a set of mapping rules. Distinct sets of mapping rules are needed to handle different OO data models. In other words, this approach treats an OO database schema as one view of a conceptual schema. Any OO schema is then an abstraction layer on top of the conceptual schema. User applications, OO or otherwise, are written based on the external schema. The advantages of the OO approach are therefore preserved at the external schema level.

The OO data modelling issues mentioned in Section 3 are resolved at the conceptual schema level. The following steps define our approach:

(Step 1) *Represent a database schema diagrammatically using an ER diagram.*

There are several advantages in using the ER approach. First, an ER diagram preserves application semantics by explicitly capturing entity types, relationship types (m-n, n-ary, recursive, ISA, etc) and their attributes. Therefore, as opposed to the approach adopted by some OO systems, not everything is treated homogeneously as objects (see Section 3.2.1). Second, nested relations are modelled naturally by using the ER concepts of multi-valued attributes, composite attributes and weak entities. The problems of modelling nested relations in the OO approach (eg. supporting symmetric queries through redundancy, updatability of nested relations, as discussed in Section 3.2.2) do not exist in the ER approach. Third, the cardinalities of entity types participating in a relationship type are displayable on the ER diagram. This is a cardinality constraint which is absent from most OO data models. Fourth, by providing strong support for association among entity types, the ER approach eliminates the need to use a navigational approach to link related entity types. Therefore, the problems that the OO approach faces in representing the relationship between two 'spouses', or in representing the relationship between suppliers and parts of Section 3.2.3 (eg. using redundant inverse pointers, supporting symmetric queries through redundancy etc) are resolved.

However, note that it is difficult to determine whether an ER diagram is the best representation for a given database. To overcome this deficiency, a normal form for ER diagram has been proposed in [31].

(Step 2) *Derive a normal form ER diagram from the ER diagram obtained in step (1).*

This can be done by applying the techniques of [31]. A normal form ER diagram is based on a strong, theoretical foundation provided by ER, normalisation and dependency theories. It is therefore possible to judge the quality of a schema based on a normal form ER diagram. In contrast, an OO schema has no formal foundation and lacks 'goodness' criteria (as discussed in Section 3.1).

Note that all the structural properties of the OO approach can be generated from a normal form ER diagram. For example the ER concepts of ISA, entity type, IS-PART-OF, weak entity type, and attribute aggregation correspond to the OO concepts of inheritance,

class, composite object, existentially dependent object, aggregation of object properties respectively. Therefore, a normal form ER diagram is a good base from which to generate OO external schemas.

(Step 3) *Imbue the normal form ER diagram of Step (2) with methods, derived attributes, deductive rules and triggers, and then eliminate all inheritance conflicts in ISA hierarchies.*

This step produces a normal form OOER diagram. The inclusion of methods, derived attributes, deductive rules and triggers (a form of production rules) provides useful features from the domains of OO programming, deductive and active databases. Methods are categorised into system generated methods, database administrator (DBA) definable methods and programmer-definable methods. System generated methods exist in both the conceptual schema and the external schema level and include code to browse instances of a class, or to retrieve or update an object's attribute values. These methods are, in general, trivial and can be more efficiently optimised by the system. DBA-definable methods are accessible by all authorised users of the database and can be defined at both the conceptual and external schema levels. For instance, the DBA can define generic methods to provide computations such as calculation of interest, commission based on generic formulas, tax expenses etc. Unlike DBA-definable methods, which are accessible by all users, programmer-definable methods, eg. hire an employee, print an entity etc are application specific. The proper place for programmer-definable methods is at the external, rather than conceptual, schema level. This is no different from conventional practice, in which a database programmer writes application programs based on an external schema available to him. Note that at the external schema level, programmer-definable methods are given meaningful names which reflect a higher level of abstraction, eg. hire_employee, fire_employee etc.

Deductive rules and derived attributes can be treated as being comparable to methods in the OO sense. They are used to derive information that are not physically stored. They are therefore intensional. In contrast, methods may have side effects, which cause changes to the extensional database. Most systems allow derived attributes to participate in queries but in some systems (eg. IRIS[17]), methods with side effects cannot be used in queries.

Triggers are typically used in active databases to enforce simple integrity constraints. In our approach, triggers can either be system generated based on the integrity constraints known to the system or explicitly coded by DBA and programmers at both the conceptual and external schema levels. For example, consider a schema in which projects sponsored by

departments are related by a relationship SPONSOR. Using the notation for triggers used in VAX Rdb/VMS, a simple trigger to enforce the referential constraint that exists between PROJECT and SPONSOR is:

AFTER DELETE ON PROJECT
REFERENCING OLD AS OLD_PROJECT
DELETE FROM SPONSOR S WHERE S.P# = OLD_PROJECT.P#

To avoid a possible misinterpretation of the roles of derived attributes, deductive rules and triggers, we have established a framework [34] which differentiates between deductive rules, triggers (production rules), integrity constraints and authorisation rules.

Note that the addition of methods, derived attributes, deductive rules and triggers to a normal form ER diagram is not sufficient to produce a normal form OOER diagram. All inheritance conflicts in ISA hierarchies must also be eliminated in order to produce a normal form OOER diagram. In [35], we provide a complete algorithm that resolves inheritance conflicts in ISA hierarchies. Most conflict resolution techniques[6,16,17,28,39] operate at a syntactic level and are either inflexible, arbitrary or not reasonable. Our approach considers the semantics of conflicting properties, and the reasons why the conflicts occur, and systematically resolves them. Briefly, inheritance conflicts can arise because of (1) redundant ISA relationships, (2) poor design, (3) superclass properties with the same name but with different semantics, and (4) superclass properties with the same name and with same semantics. Conflicts are resolved by redesigning the schema, renaming properties, explicitly selecting the inheritance path, removing redundant ISA links, redefining an overloaded property, and factoring properties to a more general class. Note that when a property is renamed, as a consequence of the conflict resolution algorithm, all methods, deductive attributes/rules and triggers that refer to the renamed property must also be updated to reference the new name.

(Step 4) *Apply a set of mapping rules to generate appropriate OO external schemas from the conceptual schema that is represented by a normal form OOER diagram.*

In [36], a set of mapping rules is provided to generate OO external schemas based specifically on the O2 data model. Similar mapping rules can be defined for other OO data models. This approach is very powerful, expressive and flexible. For instance, if a user wishes to adopt an OO perspective and construct the 'spouse' or supplier classes of Section 3.2.3, he can do so by generating the OO schema from the conceptual schema using the rules given in [36]. As another example, given a recursive relationship such as course and its pre-

requisites, a user can define a view of the transitive closure of the course relation. This closure can be computed using, say, the semi-naive method or magic sets, but the exact implementation is transparent at the external schema level.

To generalise this approach, sets of mapping rules can be defined to generate external schemas based on other data models (eg. hierarchical, network, nested relations, relational, not necessarily normalised ER diagrams, etc) from the conceptual schema. In [32,33], for example, mapping rules have been defined to generate external schemas based on hierarchical, network, relational, nested relations and ER data models. These rules are still valid for a normal form OOER diagram. The additions of methods, derived attributes, deductive rules, and triggers to the normal form OOER diagram do not invalidate any of the guidelines in [32,33]. This is because the guidelines pertain to the structural properties of a normal form ER diagram, whereas methods, derived attributes, deductive rules and triggers provide an orthogonal behavioural perspective. Therefore, depending on the needs of users, the appropriate external schemas can be defined. External schemas may not necessarily be normalised, and therefore users may perceive some form of 'redundancy'. However, this redundancy is virtual in the sense that no redundancy exists at the conceptual schema level and data is not stored redundantly.

5 Conclusion

The tremendous interest in the OO approach has exposed a number of inadequacies, which are now widely recognised. In this paper, we have presented a representative list of these inadequacies and reviewed some of the solutions that have been proposed to resolve them. After reviewing the list of inadequacies, we then focused on a set of OO data modelling issues which, we feel, can be resolved by leveraging techniques developed in ER data modelling. The OO data modelling issues that are addressed include (1) the lack of explicit support for the notion of relationship, (2) the lack of support for views, (3) the inability to judge the quality of an OO schema and (4) the lack of a reasonable approach to resolve inheritance conflicts in class hierarchies.

We outline an ER-based three level schema architecture which resolves the OO data modelling inadequacies at the conceptual schema level and preserves the advantages of the OO approach at the external schema level. Our approach allows an OO database schema, based on any OO data model, to be generated from an ER-based conceptual schema, by using a set of mapping rules. Distinct sets of mapping rules are needed for different OO data models. This approach treats an OO schema as a view of the conceptual schema. To

represent the conceptual schema, the notion of a normal form OOER diagram is introduced. We justify this choice as follows: (1) A normal form OOER diagram captures semantic information in the form of entities, relationships (m-n, n-ary, recursive, ISA etc), attributes and constraints specified by functional and multi-valued dependencies. Inheritance conflicts in the ISA hierarchies of a normal form OOER diagram are systematically removed by applying the algorithm proposed in [35]. (2) All the structural properties of the OO approach (eg. class, class hierarchy, set/tuple valued attributes, composite objects etc) can be generated from a normal form OOER diagram. The definition of mapping rules to generate OO schemas is therefore facilitated. (3) The quality of a database organised according to a normal form OOER diagram can be judged by applying classical normalisation and dependency theories.

As future work, we are investigating the updatability of the constructed entities and relationships in object-oriented external schemas. It is interesting to investigate how the presence of methods, deductive rules and derived attributes at both the conceptual and external schema level will affect the updatability problem.

References

[1] Abiteboul S., Kanellakis P., Object Identity as a Query Language Primitive, Proc Intl Conf on Management of Data, Portland Oregon, May 1989.

[2] Abiteboul S., Bonner, A., Objects and Views, Proc Intl. Conf on Management of Data, 1991.

[3] Andrews T. and Harris C., Combining Language and Database Advances in an Object-Oriented Development Environment, Proc OOPSLA '87, Orlando, Florida, Oct 87.

[4] ANSI/X3/SPARC Study Group on Data Base Management Systems, Interim Report, FDT (ACM Sigmod bulletin) Vol 7 No 2,1975.

[5] Atkinson M., et al, The Object Oriented Database System Manifesto, Proc 1st Intl. Conf. on DOOD, Kyoto, Japan, Dec 1989.

[6] Banerjee J. et al, Data Model Issues for Object-oriented Applications, ACM Transactions on Office Information Systems, Vol 5 No 1, Jan 87.

[7] Beech D., OSQL: A Language for Migrating from SQL to Object Databases, Proc. Intl. Conf. on Extending Data Base Technology, Venice, Italy, Mar 1988.

[8] Bertino E. and Kim W., Indexing Techniques for Queries on Nested Objects, IEEE Trans. on Knowledge and Data Engineering, Oct 1989.

[9] Bloom T. and Zdonik S., Issues in the Design of Object-Oriented Database Programming Languages, OOPSLA 87, Orlando, Florida, Oct 1989.

[10] Cattell R., Object Data Management: Object-Oriented and Extended Relational Database Systems, Addison Wesley 1991.

[11] Chen P.P., The Entity-Relationship Model: Toward a unified View of Data, ACM TODS, Vol 1, No 1, 1976.

[12] Cluet S., Delobel C., Lecluse C., and Richard P., Reloop: An Algebra Based Query Language for an Object-Oriented Database System, Proc 1st Intl Conf on DOOD, Kyoto, Japan, Dec 1989.

[13] Codd E.F., A relational model of data for large shared data banks, CACM 13, 6, 1970.

[14] -, Third Generation Database System Manifesto, the Committee for Advanced DBMS Function, Memo No. UCB/ERL M90/28, University of California, Berkeley, Apr 1990.

[15] Date C.J., An Introduction to Database Systems Vol I, Addison Wesley, 4th edition, 1986.

[16] Deux et al, The Story of O2, IEEE Transactions on Knowledge and Data Engineering, Vol 2 No 1, Mar 1990.

[17] Fishman et al, IRIS: An object oriented database management system, ACM Trans Office Information Syst., Vol 5 No 1, Jan 87.

[18] Goldberg A. and Robson D., Smalltalk-80, the Language and Implementation, Addison Wesley 1983.

[19] Hailpern B., Ossher H., Extending Objects to Support Multiple Interfaces and Access Controls, IEEE Transactions on Software Engineering, Vol 16, No 11, Nov 1990.

[20] Hong S., Maryanski F., Using a Meta Model to Represent Object-Oriented Data Models, Proc Intl Conf. on Data Engineering, 1990, pp 11-19.

[21] Khoshafian S.N., Copeland G.P., Object Identity, Proc OOPSLA 86, Portland, Oregon, Sept 86.

[22] Kifer M., Wu J., A Logic for Object-Oriented Logic Programming, Proc PODS, Mar 1989.

[23] Kifer M., Lausen G., F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme, Proc Intl. Conf. on Management of Data, Portland, Oregon, May 1989.

[24] Kim W. et al, Composite Object Support in an Object-Oriented Database System, Proc OOPSLA '87, Orlando, Florida, Oct 87.

[25] Kim W. et al, Object-oriented Database Support for CAD, MCC Technical Report ACA-ST-293-87, Sept, 1987.

[26] Kim W., A Model of Queries for Object-Oriented Databases, Proc. Intl. Conf. on VLDB, Amsterdam, Netherlands, Aug 1989.

[27] Kim W., Architectural Issues in Object-Oriented Databases, JOOP, Mar/Apr 1990.

[28] Kim W., An Introduction to Object-oriented Databases, MIT Press, 1990.

[29] Kung ChenHo, Object Subclass Hierarchy in SQL: A Simple Approach, CACM 33, 7, 1990.

[30] Lecluse C., Richard P. and Velez F., O2, an Object-Oriented Data Model, Proc. Intl. Conf. on Management of Data, Chicago, Ill, Jun 1988.

[31] Ling T.W., A Normal Form for Entity-Relationship Diagrams, Proc. 4th International Conference on Entity-Relationship Approach, 1985, pg 24-35.

[32] Ling T.W., A Three Level Schema Architecture ER-Based Data Base Management System, Proc. 6th International Conference on Entity Relationship Approach, 1987, pp 181-196.

[33] Ling T.W., External Schemas of Entity-Relationship Based Data Base Management Systems, in Entity-Relationship Approach, C. Batini (Eds.), Elsevier Science Publishers, 1989.

[34] Ling T.W., Teo P.K., On Rules and Integrity Constraints in Database Systems, Information and Software Technology, Vol 34 No 3, Mar 1992.

[35] Ling T.W., Teo P.K., Resolving Inheritance Conflicts in Object-Oriented Systems, Submitted for Publication, 1993.

[36] Ling T.W., Teo P.K., Yan Y.Y., Generating Object-Oriented Views from an ER-Based Conceptual Schema, Proc. 3rd Intl Symposium on Database Systems for Advanced Applications, Apr 6-8, Taejon, Korea, 1993.

[37] Maier D., Stein J., Otis A., Purdy A., Development of an Object-oriented DBMS, Proc OOPSLA 86, Portland, Oregon, Sept 86.

[38] Maier D., Comments on the "3rd Generation DataBase System Manifesto", Oregon Graduate Institute, 1991.

[39] Rowe L. and Stonebraker M., The Postgres Data Model, in The Postgres Papers, Memo No UCB/ERL M86/85 Jun 87 (Revised), University of California, Berkeley.

[40] Stefik M. and Bobrow D., Object-oriented Programming: Themes and Variations, The AI Magazine, 40-62, Jan 1986.

[41] Shilling J., Sweeney P., Three Steps to Views: Extending the Object-Oriented Paradigm, Proc. OOPSLA 89.

[42] Stonebraker M. et al, The Implementation of POSTGRES, IEEE Transactions on Knowledge and Data Engineering, Vol 2, No 1, Mar 1990.

[43] Stonebraker M. et al, On Rules, Procedures, Caching and Views in Database Systems, Sigmod 1990.

[44] Stroustrup B., The C++ Programming Language, Addison Wesley, 1986.

[45] Ullman J.D., Database Theory - Past and Future, Proc. 6th PODS (San Diego, CA, 1987).

[46] Zdonik S. and Maier D., Fundamentals of Object-Oriented Databases, in Readings in Object-oriented Database Systems, Morgan Kaufman, San Mateo, Ca., 1990.