VLDB Journal, 3, 53-76 (1994), Fred J. Maryanski, Editor

A Pattern-Based Object Calculus

Nabil Kamel, Ping Wu, and Stanley Y.W. Su

Received March 11, 1992; revised version received July 2, 1993; accepted July 16, 1993.

Abstract. Several object-oriented database management systems have been implemented without an accompanying theoretical foundation for constraint, query specification, and processing. The pattern-based object calculus presented in this article provides such a theoretical foundation for describing and processing object-oriented databases. We view an object-oriented database as a network of interrelated classes (i.e., the intension) and a collection of time-varying object association patterns (i.e., the extension). The object calculus is based on first-order logic. It provides the formalism for interpreting precisely and uniformly the semantics of queries and integrity constraints in object-oriented databases. The power of the object calculus is shown in four aspects. First, associations among objects are expressed explicitly in an object-oriented database. Second, the "nonassociation" operator is included in the object calculus. Third, set-oriented operations can be performed on both homogeneous and heterogeneous object association patterns. Fourth, our approach does not assume a specific form of database schema. A proposed formalism is also applied to the design of high-level object-oriented query and constraint languages.

Key Words. Object-oriented databases, association patterns, semantic constraints, query expressions.

1. Introduction

Many object-oriented (OO) database models (Batory and Kim, 1985; Banerjee et al., 1987; Fishman et al., 1987; Hull and King, 1987; Su et al., 1989) and OO database management systems like Vbase (Ontologic, Inc., 1988), Iris (Fishman et al., 1987), GemStone (Maier et al., 1986), ORION (Banerjee et al., 1987), O_2 (Lecluse et al., 1988), and ObjectStore (Object Design, Inc., 1990) have emerged recently for supporting advanced application areas such as CAD/CAM, office automation, and

Nabil Kamel, Ph.D., is Assistant Professor; Ping Wu, Ph.D., is Research Assistant; and Stanley Y.W. Su, Ph.D., is Professor, Database Systems Research and Development Center, Computer and Information Sciences Department, E470 CSE, University of Florida, Gainesville, FL, 32611–6125.

multi-media databases. Unlike the relational approach to databases which was based on a firm theoretical foundation, that is, the mathematical notion of relations and the accompanying relational algebra and relational calculus, object-oriented databases and object-oriented database management systems (DBMSs) are primarily founded on ideas adopted from object-oriented programming languages. Implementations of object-oriented DBMSs have been carried out without the accompanying theoretical foundation. As a result, three major problems have surfaced in OO database applications.

First, there is no query language for existing OO database systems. Most of the existing OO systems provide their own query languages to manipulate the objects in the database (Shipman, 1981; Zaniolo, 1983; Schaffert et al., 1986; Fishman et al., 1987; Carey et al., 1988).

Second, most of the reported OO data models do not provide formal constraint languages for declaratively specifying various semantic constraints found in different application domains. Like relational databases (Ullman, 1982), object-oriented databases need such languages to define security and integrity constraints so that an application world can be modeled accurately. In most of the implemented OO DBMSs, a rudimentary set of constraints is "hard-coded" to represent the semantics of a data model. Constraints not captured by the data model have to be implemented as procedures in application programs. Several researchers have introduced different constraint representations such as Horn logic clauses (Urban and Delcambre, 1989), constraint equations (Morgenstern, 1984), and rules based on an associative net (Shepherd and Kerschberg, 1984). However, no uniform constraint representation is established for interrelating constraints captured by different data models.

Third, the existing implementations of query processors and query optimizers are not guided by a well-defined mathematical formalism. Efficiency in object-oriented DBMS implementations is difficult to achieve without a solid theoretical foundation.

Several research efforts have been undertaken to establish a theoretical foundation for processing OO databases. Two general approaches have been made: the algebraic approach and the logic approach. The first approach defines the formal semantics of an OO query model based on object algebras (Manola and Dayal, 1986; Osborn, 1988, 1989; Shaw and Zdonik, 1989; Guo et al., 1991; Straube, 1991). Among them, only the algebra in Straube (1991) has an accompanying calculus. The second approach uses logic to formally specify the semantics of the OO paradigm (Maier, 1986; Chen and Warren, 1989; Kifer and Lausen, 1989; Kifer and Wu, 1989). The most complete work of this approach is represented by F-logic (Kifer and Wu, 1989). F-logic is a database logic formalism which provides a clean declaration for most of the "object-oriented" features such as object identity, complex objects, inheritance, methods, etc. In contrast, our model is simpler, (i.e., it contains fewer linguistic constructs) and lower level in the sense that it does not attempt to tie the language too strongly to higher-level concepts, such as inheritance. The language is strongly rooted in first order logic. We rely on the notions of association and non-association among objects exclusively to specify queries and constraints

as patterns of associated (or non-associated) objects. This flexibility allows the language to be used with other navigational models, such as the Entity-Relationship (E-R), network, and hierarchical data models, none of which adopts the notion of inheritance (in their purest form).

In this article, we present a pattern-based object calculus for formally describing and manipulating objects and object association patterns in OO databases. This object calculus is based on first-order logic. It is the accompanying formalism to the association algebra reported by Guo et al. (1991) and Su et al. (1993). Its main feature is the concise yet semantically powerful expressions for specifying queries and constraints in terms of patterns of object associations. It captures the semantics of "association" and "nonassociation" relationships explicitly, that is, it identifies respectively the related and unrelated objects among classes which are associated with one another in the intensional database. The semantics of nonassociation cannot be easily expressed in other calculus-based languages. This calculus can also express set operations on both homogeneous and heterogeneous object association patterns whereas set operations expressible in the relational calculus can only be performed on union-compatible (i.e., homogeneous) relations. A calculus expression specifies a subdatabase that satisfies the expression and the calculus preserves the *closure* property (i.e, the output of an expression is association patterns which can be operated on by another expression). Because of its uniform object association pattern representation, the object calculus can be used to express formally both search queries and semantic constraints. Therefore, unlike some implemented systems in which the query processing subsystem is separated and different from the constraint enforcement subsystem, a DBMS can be built to use the same semantic constructs expressed in the calculus for both query and constraint processing. It is also shown later in this article that the calculus can provide a formal basis for the design of high-level OO query and constraint languages.

The rest of this article is organized as follows. In Section 2, an OO view of databases based on objects, classes, and associations is first described and compared with the relational view of databases. Section 3 presents the main features of the object calculus including its syntax and formal semantics. The object calculus' expressive power in formulating queries is presented in Section 4 by examples. Section 5 shows an application of the object calculus for defining semantic constraints. The status of the system implementation based on the OO view is briefly described in Section 6. Concluding remarks are given in Section 7.

2. Object-Oriented vs. Relational Databases

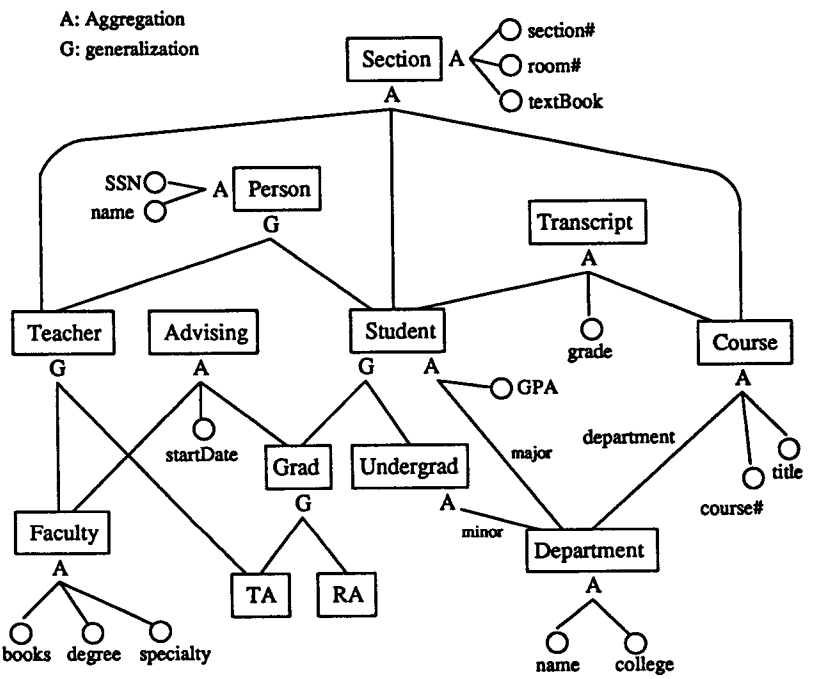
In this section, we point out the main differences between the relational and object-oriented data modeling paradigms as a justification for introducing two important semantic constructs: the *association* and the *nonassociation* operators, in the object calculus.

In the relational model, relations are used to represent entities and associations in the real world. An entity or association instance (or an object) is represented by a tuple, and it is distinguished from others by its unique primary key value. The entity integrity enforces the rule that null values cannot be accepted as the primary key values. To relate one instance of one relation to an instance in a second relation, an attribute, called a *foreign key*, is used in the second relation. Foreign-to-primary-key matching is used in query processing to identify *relationships* between tuples. It is carried out by the expensive (i.e., computationally costly) *join* operation between relations. Referential integrity guarantees such a match between a foreign key value and a primary key value. In a relational query, the matchings of keys and foreign keys must be explicitly specified when traversing a number of relations. We call this type of query specification and processing *attribute-based.*

Unlike the relational view, our object-oriented view of a database is based on the essential characteristics of the object-oriented paradigm. First, an object in the database represents a real world entity, such as a physical object, an abstract thing, an event, a process, or whatever is of interest, and each object is assigned a system-defined, unique object identifier (OID). The OID allows an object to be distinguished from other objects. Second, objects having the same structural and behavioral properties are grouped together to form an object class. Third, a class can associate (interrelate) with other classes to represent the relationships among the real world entities. As a result, objects in a class can associate with objects in other classes. Different association-types between or among object classes have been recognized in different OO data models. Two of the most commonly recognized association types are *aggregation* and *generalization.* Aggregation association captures the semantics of "is part of," "is composed of," or "has properties of" relationship. It allows an object to be defined in terms of other objects. In Figure 1, for example, a Section object can be associated with (or semantically is composed of) a Teacher object, a (set of) Student object(s) (the cardinality mapping between Section and Student can be specified), and a Course object. The generalization association captures the semantics of "is a" or "is a kind of" relationship. Objects in the subclass inherit the structural and the behavioral properties (operations and rules) from one or more superclasses. In our discussion, an object is represented using its OID and it encapsulates all its structural and behavioral properties, such as its inherited attributes, its associations with other objects, and so on.

An association between two objects is represented by bi-directionally linking these two objects using OID references which can be easily mapped to some physical pointers for locating these objects. For this reason, we can simply use the association operator "$*$" to specify the existence of an association between two objects. For example, we use the expression $obj_1 * obj_2$ to denote that an object which is bound to the object variable obj_1 is associated with another object which is bound to the object variable obj_2. Similarly, we can use the nonassociation operator "!" to explicitly specify the absence of an association between two objects, (e.g., $obj_1 ! obj_2$). The notion of nonassociation is not included explicitly in the existing relational languages.

Figure 1. University database schema

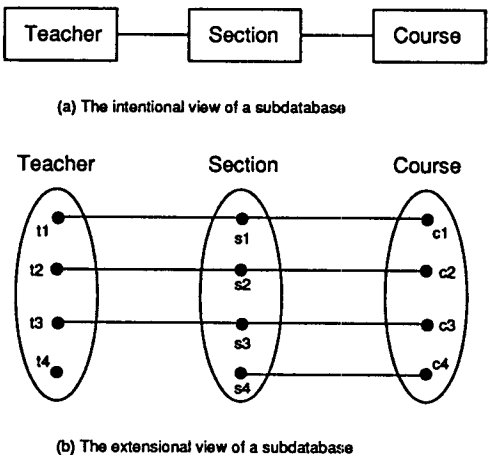


It has to be expressed in a round-about way by a number of other operators.

Different OO data models may capture different association types such as aggregation, generalization, using, composition, etc. The semantics of these association types in a database can be represented by a set of constraints which govern the processing of association-type objects in an object-objected DBMS. For this reason, their semantics do not have to be restated in the expressions of the object calculus. The object calculus presented in this article is developed for expressing and processing the common primitives found in most OO models, (i.e., objects, classes, and association between objects and classes). The semantics of association types are handled by the underlying DBMS.

To conclude our discussion on the OO view of databases, we consider an example database in a university environment. Figure 1 is a schema for the university database, which is modeled using the object-oriented semantic association model (OSAM*) (Su et al., 1989). The OSAM* data model provides five types of system-defined semantic associations, although only two types appear in the schema, namely, Aggregation (A) and Generalization (G). In the schema, class Section has six attributes which are represented by "A" links to domain classes (section#, room#, and textbook) and entity classes (Teacher, Student, and Course). That is, a Section object instance is composed of a section number, a room number, a text book and references to objects of Teacher, Student and Course. Class Person is defined as the superclass of

Figure 2. Object-oriented view of a subdatabase



Teacher and Student. An OO view of a subdatabase consisting of Teacher, Section, and Course classes is shown in Figure 2. Figure 2(a) is the intensional (i.e., schema) view of the subdatabase which shows the classes and their interconnections. Figure 2(b) is the extensional (i.e., instance) view which shows the objects (represented by OID's) in different classes and their associations (represented by edges). Because the nonassociations among objects are the complement of associations among the objects, they are not drawn explicitly in the figure. From the extensional pattern we know that Teacher t1 teaches Section s1, which is opened for Course c1. We also know that Teacher t1 does not teach Section s2.

3. Object Calculus

In this section, we first introduce the basic features of the object calculus and then describe its syntax and semantics in detail.

3.1 Basic Features of the Object Calculus

The underlying philosophy of the object calculus is to provide a formalism for describing and manipulating objects and object associations. Objects and object associations are the basic constituent elements used to express the object association patterns which may have linear, tree, or general network structures. The object calculus allows object association patterns to be specified by the user as search conditions which are to be matched against the extensional representation of an OO database.

Database objects are referenced by *object variables*. An object variable is defined for a class so that it can be bound to any object in the class. For example, if variable

sect is defined for class Section and we assume the extension of the database is as shown in Figure 2(b), then *sect* can be bound to any one of the objects, s_1, s_2, s_3, or s_4, in class Section.

At the primitive level of OO database representation, two objects can be either associated or not associated with each other. Suppose variables *tchr* and *sect* are defined for classes Teacher and Section, respectively. If a user is interested in those teachers who teach some sections (i.e., the associations between the Teacher instances and the Section instances), then he/she can use the expression *tchr* $*$ *sect* to represent the intended semantics. Here, *tchr* $*$ *sect* is a predicate which is an expression that returns "true" or "false." What the expression means is to: (1) bind the variables to two objects in their respective classes, (2) return "true" if the bound objects satisfy the relationship specified by the association operator $*$ in the predicate, (3) return "false" if the bound objects do not satisfy the relationship as specified. Again, we assume the database of Figure 2(b), object pairs (t_1, s_1), (t_2, s_2), and (t_3, s_3) satisfy the predicate because there are associations between the paired objects in the database, and all other object pairs make the predicate evaluate to false. On the other hand, predicate *tchr* ! *sect* evaluates to true when the object bound to *tchr* is not associated with the object bound to *sect* in the database. For example, the following pairs of objects satisfy the above predicate: (t_1, s_2), (t_1, s_3), (t_1, s_4), (t_2, s_1), etc. Only three pairs of objects make the predicate evaluate to false: (t_1, s_1), (t_2, s_2), and (t_3, s_3).

When bound to the same pair of objects, the predicates $var_1 * var_2$ and var_1 ! var_2 are complements of each other. That is, if the first predicate is true, the second is false; and vice versa. A user-issued query is interpreted as a request to output those database objects that satisfy the specified predicate. In the rest of this section, we present the detailed syntax and semantics of the object calculus.

3.2 Grammar

We first introduce a concrete syntax for the object calculus. This syntax is similar to the tuple-oriented relational calculus with the exception of the introduction of an association pattern into a "well-formed formula" (wff) and the binding of variables to objects. The Backus-Naur Form (BNF) grammar of the language is given in Figure 3.

Several points about the BNF are noted below:

1. The categories "class," "variable," "attribute," and "asso_name" are defined to be *identifiers* (a terminal category with respect to this grammar). They represent a class name, an object variable name, an attribute name, and an association name, respectively.

2. The category "comparison" represents either a simple scalar comparison operation (=, <>, >=, etc.) between two scalar values or an object comparison operation (=, <>) between two object variables. A scalar

Figure 3. BNF grammar for the object calculus

```
range-definition
  ::= RANGE OF variable IS range-item
range-item
  ::= class | ( expression )
expression
  ::= target-item-commalist [ WHERE wff ]
target-item-commalist
  ::= target-item | target-item , target-item-commalist
target-item
  ::= variable . attribute | variable
wff
  ::=  association-pattern
    | comparison
    | ( wff )
    | NOT wff
    | wff AND wff
    | wff OR wff
    | EXISTS variable ( wff )
    | FORALL variable ( wff )
    | IF wff THEN wff
association-pattern
  ::= class *[asso_name] association-pattern
    | class ![asso_name] association-pattern
comparison
  ::= target-item op target-item
op
  ::= < | > | <= | >= | = | <>
```

value, in turn, consists of either an attribute value represented by an attribute reference of the form "variable.attribute" or a scalar literal.

3. The category "association-pattern" represents either an object of the form "*class*" or an association pattern of the form "$class_1$ *op* $class_2$ *op* ... *op* $class_n$," where "*op*" can be either "$*[asso_name]$" or "$![asso_name]$", and "$class_i$" is an object variable for the class. In general, we use the "*asso_name*" to designate a specific association between two classes when there are multiple associations between the classes. To simplify the presentation, we assume that there is only one association between two classes and omit the "*asso_name*" in the rest of this article. An expression of $class_1 * class_2 ! class_3$ is the shorthand for the expression ($class_1 * class_2$) *AND* ($class_2 ! class_3$).

4. The category *wff* represents a "well-formed formula." Wffs are discussed in detail in the following subsections.

5. For simplicity of descriptions, the following abbreviations are used—NOT ($\neg$), AND ($\wedge$), OR ($\vee$), <> ($\neq$), EXISTS ($\exists$), NOT EXISTS ($\nexists$), and FORALL ($\forall$). The logical implication operator IF f THEN g is represented by $f \rightarrow g$.

3.3 Object Variables

An object variable is defined by means of a statement of the form

RANGE OF x IS X

where x is the name of an object variable that ranges over class X. Hence, the object variable x represents an object of class X in the database. When there is no explicit range definition, the default is that if a class name is used as an object variable name, then the object variable represents an object of the class. Multiple object variables can be defined for a class as illustrated by the following example.

RANGE OF *e*1 IS *Employee*
RANGE OF *e*2 IS *Employee*

Each variable represents a separate scan of instances of Employee class.

3.4 Wff Involving Association Patterns

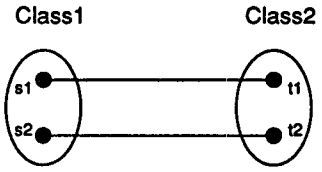
Based on the BNF grammar, a wff takes one of the following forms: a simple comparison, an association pattern, logical operations (i.e., with NOT, AND, OR), quantified expressions (i.e., with the existential quantifier EXISTS and the universal quantifier FORALL), and the logical implication operation (i.e., IF f THEN g).

A wff is a predicate. As defined in the predicate calculus, a predicate is an expression that returns a truth value (*true* or *false*). When a wff takes the basic form of a simple comparison, its meaning is apparent.

The meaning of a wff that takes the form of an association pattern is decided by its *domain* and *interpretation*. The domain of an association pattern is the underlying database, which includes the objects of different classes and the object associations. The domain of an object variable defined for a class is all the object instances of the class. An *interpretation* of a wff is the objects bound to the variable occurrences in the wff and the associations among these objects. An object variable occurrence can be bound to any object in its domain. In general, for a given wff, there may be many different interpretations, thus, resulting in different truth values. A wff is said to be *valid* if it yields the value *true* for every interpretation. Thus, a wff is *nonvalid* if and only if there is some interpretation for which the wff yields the value *false*. A wff is said to be *unsatisfiable* if it yields the value *false* for every interpretation.

Assume that two classes $class_i$ and $class_j$ have been defined in the database schema and that there is an association between these two classes. Also assume that object variable $objvar_i$ and $objvar_j$ range over $class_i$ and $class_j$, respectively. The basic association patterns for the two given classes are $objvar_i * objvar_j$ and $objvar_i ! objvar_j$. The notions "*" and "!" can be viewed as the two binary predicates, *Association*(*x*,*y*) and *Nonassociation*(*x*,*y*) respectively, where x and y are object variables. Therefore, we have the following equivalent expressions:

Figure 4. Evaluation of wff expressions



(a) A simple database for the basic association patterns.

Interpretation	Value of the wff	
	objvar1 * objvar2	objvar1 ! objvar2
(s1, t1)	true	false
(s1, t2)	false	true
(s2, t1)	false	true
(s2, t2)	true	false

(b) The values of the wff's under different interpretation.

$$objvar_i * objvar_j \equiv Association(objvar_i, objvar_j)$$
$$objvar_i \; ! \; objvar_j \equiv Nonassociation(objvar_i, objvar_j)$$

As an example, consider a simple database whose extensional view is shown in Figure 4(a). The domains of $objvar_1$ and $objvar_2$ are $\{s_1, s_2\}$ and $\{t_1, t_2\}$, respectively. The truth values of $objvar_1 * objvar_2$ and $objvar_1 ! objvar_2$ under different interpretations are shown in Figure 4(b).

A quantified wff is evaluated by applying the object variables in the wff to all the objects in their domains. If the database objects satisfy the semantics of a quantified wff, then this wff is evaluated to true, and otherwise evaluated to false.

(a) The quantifier FORALL (universal quantifier) stands for the words "for all ... is true." The value of $\forall$ *objvar*(*wff* (*objvar*)) is true if for all objects over *objvar*'s domain, the value of *wff*(*objvar*)) (with the object bound to all occurrences of *objvar*) is true; otherwise, the value of $\forall$ *objvar*(*wff*(*objvar*)) is false. The expression *wff*(*objvar*) means that the wff has free variable *objvar*.

(b) The quantifier EXISTS (existential quantifier) stands for the words "there exists ... such that ... is true." The value of $\exists$ *objvar*(*wff*(*objvar*)) is true if there is an object over *objvar*'s domain, such that the value of *wff*(*objvar*) (with the object bound to all occurrences of *objvar*) is true; otherwise, the value of $\exists$ *objvar*(*wff*(*objvar*)) is false.

We introduce an explicit syntactic form for the *logical implication operator*. If f and g are wffs, then the logical implication expression "IF f THEN g" is also defined to be a wff. As we will see later, this logical implication expression as a wff can be used widely to express various kinds of constraint rules in knowledge bases.

3.5 Expressions

An object calculus expression has the following form:

target-list [WHERE f]

A target list consists of a list of "target items" separated by commas, in which each item is either a simple object variable name such as *O* or an attribute expression of the form *O.a*. The value of the object calculus expression is defined to be all sets of objects in the target-list *and* the associations among these objects for which *f* evaluates to true. In other words, an object calculus expression returns *both* database objects and associations among them.

Objects returned by an object calculus expression may be different from the original objects in the database, dependent on the form of the target item. When the target item is in the form of an object variable name such as *O,* then the qualifying objects are returned as original database objects. When the target item is in the form of an attribute expression such as *O.a,* then all attributes of the returned objects are stripped from them except for the specified attribute. If more than one attribute is specified with the same object variable in different target items, say *O.a, O.b,* then all these specified attributes are retained for the returned objects. In our OO view of databases, an object encapsulates its structural and behavioral properties. Therefore, the object's inherited attributes can also be specified in target items.

Completeness. The notion of completeness for database languages was first defined for the relational model (Codd, 1972). According to that definition, a language is relationally complete if databases definable by relational calculus expressions are retrievable using statements of that language. This notion of completeness was tightened later by requiring that any database definable by a single relational calculus expression be retrievable via a single statement of the language (Date, 1982). The same notion of completeness can be applied to any other OO query language vis-a-vis the object calculus. A more detailed discussion of completeness can be found in Su et al. (1993) which defines this notion in the context of the association algebra.

Safety and Closure of Expressions. A system that allows writing expressions against finite databases that return infinite results is called *unsafe*. In the same manner, an expression written against a finite database that could return an infinite result is called unsafe. It is possible to write safe expressions within unsafe systems. It is, however, not possible to write unsafe expressions within safe systems.

Unsafe Systems: Consider the object calculus expression:

RANGE of T IS Teacher
RANGE of P IS Person
T, P WHERE ¬(T * P)

If the calculus defines the Range statement to range its variable over its entire domain (e.g., all possible teachers in the universe), then this expression is unsafe because it will return all possible non-associated teacher-person pairs, even if they are not in the database. The following is an example of a safe expression written under the same unsafe system:

RANGE of T IS Teacher
RANGE of P IS Person
RANGE of F IS Faculty
RANGE of S IS Student
T, P WHERE ¬(T * P) AND (T * F) AND (P * S)

The above expression is safe because of the further qualification placed on Teachers and Persons (Teachers must be existing faculty and Persons must be existing students).

The object calculus eliminates the problem of unsafe expressions by restricting the meaning of the Range statement to range its variable over only the current class contents inside the database. For example, the statement:

RANGE of T IS Teacher

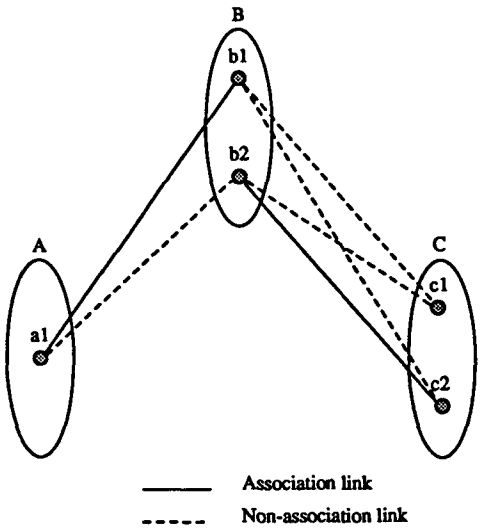
will be interpreted as ranging the object variable T over the existing teacher objects in the database. This interpretation guarantees safety for all object calculus expressions, and consequently results in a safe system.

Another important property for database systems is the notion of closure. A database language is called *closed* if the expressions of the language are always guaranteed to return results in a compatible form for further operations. In other words, the expressions of the query language can be nested to any depth. The object calculus assumes databases consisting of four types of primitive entities: objects, classes, association links, and non-association links, as shown in Figure 5. Object calculus expressions operate against databases consisting of only these four primitive constructs and always return finite databases consisting of the same constructs. The closure of the object calculus follows directly from this observation.

4. Using Object Calculus to Express Queries

The object-oriented view of an application world can be represented in the form of a network of classes and associations among these classes. Since an application is modeled in such an object-oriented fashion, the information about the application is therefore stored as objects and associations among objects in the database. Users can query the database by specifying the desired object association patterns in their queries.

Figure 5. Basis of closure property



The object calculus provides a formalism to formally express queries that are to be processed against an object-oriented database. A query expression specified by a predicate is interpreted as identifying those database objects that satisfy the specified predicate. The subdatabase formed by the qualified database objects can be output to a user or subject to other system-defined or user-defined database operations as desired.

In this section, we present several examples on the use of the object calculus in formulating queries. The basic features of the object calculus are shown through these examples. All example queries are issued against the university database modeled in Figure 1. Unless specified otherwise, all object variables share the same names as the class names on which they range.

Query 1. Display the names of those teachers who teach some sections and the section#'s for these sections.

Teacher.name, Section.section#
WHERE Teacher * Section

In this query the *association* operator is used in the WHERE clause. The wff predicate *Teacher * Section* evaluates to true for each pair of Teacher object and Section object that are associated with each other. The calculus expression returns the names of the qualified Teacher objects and the section# of the qualified Section objects.

Query 2. Display the department names for all departments that offer 6000 level courses that have current offerings (sections). Also, display the titles of these courses

and the textbooks used in each section.

```
Department.name, Course.title, Section.textbook
WHERE Department * Course * Section
      AND Course.course# ¿= 6000 AND Course.course# ¡ 7000
```

In this calculus expression, the wff specifies those objects in classes Department, Course, and Section that are associated with one another in the manner that a Department object is associated with a Course object, which is in turn associated with a Section object; also, the Course objects' attribute values of course# must be in the range between 6000 and 7000. The calculus expression returns the attribute values of those database objects that satisfy the wff.

Query 3. Display the names of those graduate students who are TAs but not RAs.

```
Grad.name
WHERE (Grad * TA) AND (FORALL RA (Grad!RA))
```

This query shows the use of *nonassociation* between objects and the universal quantifier. The wff in the WHERE clause identifies those objects in the classes Grad, TA and RA satisfying the condition that those Grad objects associated with TA objects are not associated with any RA object.

Query 4. Display the ssn's of all graduate students (whether they have advisors or not) and for those graduate students who have advisors, display their advisors' names.

```
Grad.ssn, Faculty.name
WHERE Grad * Advising * Faculty OR Grad
```

This query illustrates the calculus' capability in specifying heterogeneous association patterns in an expression. The wff in the WHERE clause specifies two different association patterns: *Grad * Advising * Faculty* and *Grad*. These two association patterns are connected by the logical OR operator to capture the semantics of the outer-join concept introduced by Codd (1979). The calculus allows heterogeneous patterns of object associations to be unioned to retain all Grad objects, whether they are advised by faculty members or not. This is different from the relational set operations which operate only on union-compatible relations. The logical OR and AND operators can operate on potentially very complex heterogeneous patterns.

Query 5. Find the faculty member(s) in the Electrical Engineering department who advise(s) all the EE department's graduate students.

```
RANGE OF Grad1 IS
(Grad WHERE Grad * Student * Department AND Department.name = "EE")
Faculty WHERE FORALL Grad1 (Faculty * Advising * Grad1)
```

This query first defines a variable that ranges over a set of graduate students whose major department is the Electrical Engineering department. Then, it searches for the faculty member(s) who advise(s) all the graduate students in that set. To express the semantics of this query in the calculus, object variable *Grad*1 is ranged over the class Grad to represent the subset of graduate students who are in the EE department. In addition, the universal quantifier is applied to Grad1 to represent the semantics of one faculty member advising all the graduate students in Grad1. This example illustrates the closure property of the object calculus in which Grad1 identifies a subdatabase which in turn becomes an operand of the expression in the second WHERE clause.

The object calculus can also be used to specify complex nonlinear association patterns (i.e., tree or lattice structures) in a database by using the logical AND and OR constructs. For example in Figure 6(a), a complex association pattern is specified in the database schema level. In the figure, the logical AND branching from class B specifies that any object in class B must associate with an object in class C and with an object in class D as well. Likewise, the logical OR branching from class C specifies that any object in class C must associate with either an object in class E or an object in class F. Figure 6(b) further illustrates this association pattern with instantiated objects in the database. For such a complex pattern, it can be expressed in the object calculus as

$$(A * B * C * E \vee A * B * C * F) \wedge (A * B * D * G)$$

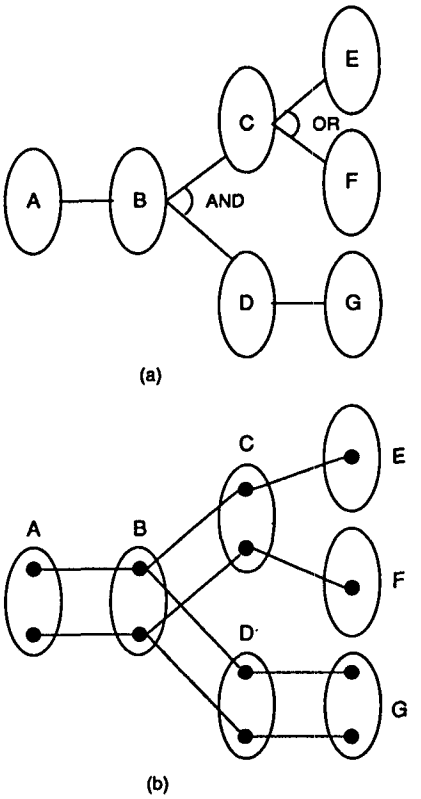
We should clarify one very important point. The primary purpose of the calculus is not merely for data retrieval, though the previous examples may suggest so. The fundamental intent of the calculus is to allow *the writing of expressions* that identify objects satisfying specified patterns of object associations and perform system-defined and/or user-defined database operations (e.g., DELETE, UPDATE, ASSIGN_PROJECT, etc.) on these objects. Therefore, the object calculus provides a theoretical foundation for writing OO query expressions. It can serve as a template when designing for OO query languages. To demonstrate this point, we give the following example to show how an (pseudo) OO query language can be designed based on the object calculus.

Structurally, the (pseudo) query language has two parts: the subdatabase definition part and the operation part. In the subdatabase part, by using object calculus expressions, we specify a subdatabase that satisfies the user-defined semantics. In the operation part we can specify system-defined or user-defined operations that are performed on the specified subdatabase. Consider two examples that respectively employ a system-defined operation and a user-defined operation with respect to the database schema of Figure 1.

Example 1. Delete those Faculty members who are not currently teaching any section of a course.

DELETE

Figure 6. A complex association pattern in the database



Faculty WHERE FORALL Section (Faculty * Teacher ! Section)

In this query, the subdatabase is specified by the object calculus expression:

Faculty WHERE FORALL Section (Faculty * Teacher ! Section)

and the system-defined operation DELETE is performed on the *Faculty* objects that constitute the subdatabase returned by the calculus expression.

Example 2. Assign projects to those students who are in EE department.

ASSIGN_PROJECT
Student WHERE (Student * Department) AND (Department.name = "EE")

In this query, the object calculus expression:

Student WHERE (Student * Department) AND (Department.name = "EE")

specifies the *Student* objects as the subdatabase, and the user-defined operation

ASSIGN_PROJECT is performed on these objects in the subdatabase. Here, we assume that the user-defined operation ASSIGN_PROJECT is implemented as a method of the class Student.

5. Using Object Calculus to Express Semantic Constraints in OO Databases

For a database to be an accurate model of an application world, semantic constraints that identify the invalid or illegal states pertaining to the real world entities and their relationships need to be captured in the database. Commonly, database constraints are specified as integrity and security rules. The semantics of these rules are enforced by the DBMS at all times. If any database object violates a rule, then the database enters into an illegal state.

We can use object calculus wffs to represent rules. However, the semantics of a wff used as a rule are somewhat different from those of a wff used as a search query. In a query expression, a wff is used to identify those database objects which evaluate the wff to true. In a rule expression, it is used to express a constraint such that *all* database objects comply with the constraint. That is, all the concerned database objects must satisfy the rule. We now give the formal definition. A rule is said to be *satisfied* if and only if its object calculus wff expression is evaluated to true under all interpretations in its database domain. A rule is said to be *violated* if and only if its object calculus wff expression is evaluated to false under some interpretation in its database domain.

A database needs to maintain not only the conventional integrity constraints supported by existing commercial systems but also many other application-oriented constraints which can be represented in an object-oriented database by patterns of object associations (Su and Alashqur, 1991). Certain association patterns among objects may need to be maintained in a database at all times. In addition, the existence/non-existence of some association patterns may depend on the existence/non-existence of other association patterns. In the following subsections, we describe various types of constraints that may exist in object-oriented databases and formally specify these constraints by rules using object calculus expressions. The usage of object variables is self-explanatory in the following examples; thus the definitions of these variables are omitted.

5.1 Constraints on Attribute Values

This type of constraint is like the ones in the relational model in that restrictions are specified on the attribute values. They can be represented simply by first-order predicates of the object calculus.

Example 1 (unconditional constraint). Every employee's salary must be higher than $25K.

Rule 1: *Employee.salary* > 25K

Note that this rule is expressed as a wff with a free object variable (i.e., Employee). The number of interpretations for this wff is equal to the number of Employee object instances in a given database. For this rule to be satisfied, the wff must be true in all the interpretations. On the other hand, if the wff is false in any of its interpretations, then the rule is violated. The semantics of this rule can also be expressed by the following logically equivalent wff with a universal quantifier:

$$\forall \textit{Employee}\ (\textit{Employee.salary} > 25\text{K})$$

That is, given a database, these two rules have the same truth value. Their difference is that the second expression does not have a free object variable. In the constraint rules given below, we do not use quantifiers if they are not necessary.

Example 2 (conditional constraint). Employees whose ages are over 45 must have salaries higher than $35K.

Rule 2: IF Employee.age > 45 THEN Employee.salary > 35K

or equivalently

$$\neg\ (\textit{Employee.age} > 45) \vee (\textit{Employee.salary} > 35\text{K})$$

5.2 Constraints on Association Patterns

Some constraints can be specified in terms of the object association patterns. These constraints can be classified into two categories. The first category is constraints that specify nonpermissible extensional pattern(s) of object associations. That is, the constraints enforce the restrictions that certain association patterns of objects in some specified classes are not allowed to exist in the database at any point in time. The second category is constraints that specify that certain object patterns must (or must not) exist if some other object patterns do (or do not) exist. These two categories of constraints are illustrated by the following examples using the university database schema in Figure 1.

5.2.1 Nonpermissible Extensional Patterns. This type of constraint states that, if a database operation results in the formation of nonpermissible extensional pattern, then the stated constraint is violated.

Example 3. A faculty member must have a Ph.D. degree.

Rule 3: $\exists \textit{Faculty}(\textit{Faculty.degree} \neq \textit{"Ph.D."})$

or equivalently

$$\forall \textit{Faculty}(\textit{Faculty.degree} = \textit{"Ph.D."})$$

If a Faculty instance in the database is updated and the attribute degree has a value other than "Ph.D.," or a new instance is created with that value, this rule is violated and some action should be taken by the DBMS to correct the situation.

Example 4. An undergraduate student cannot register in a graduate course (i.e., one with a course number greater than 5000).

$$\text{Rule 4: } \neg\ (\textit{Undergrad} * \textit{Section} * \textit{Course} \wedge \textit{Course.course\#} > 5000)$$

This rule prohibits an object association pattern specifying that an Undergrad instance is associated with a Section instance and the Section instance is associated with a Course instance whose course# is greater than 5000.

5.2.2 Conditional Constraints. This type of constraint explicitly enforces the relationship between object association patterns. That is, the existence of one association pattern implies the existence of another association pattern in the database. The implication construct in the object calculus

$$\text{IF wff THEN wff}$$

represents precisely the semantics of this kind of constraint, where each wff specifies an object association pattern. The following examples are constraints of this type.

Example 5. A graduate student who is an RA must have an advisor, that is, if a Grad instance is associated with an RA instance, it must also be associated with an Advising instance.

$$\text{Rule 5: IF } \textit{Grad} * \textit{RA} \text{ THEN } \exists\ \textit{Advising}(\textit{Grad} * \textit{Advising})$$

or equivalently

$$\neg\ (\textit{Grad} * \textit{RA}) \vee \exists \textit{Advising}(\textit{Grad} * \textit{Advising})$$

Example 6. TAs who are majoring in the department of Electrical Engineering cannot teach courses that belong to other departments.

Rule 6:

RANGE OF Dept_1 IS Department

RANGE OF Dept_2 IS Department

IF $\textit{TA} * \textit{Dept\_1} \wedge \textit{Dept\_1.name} = \textit{"EE"}$
THEN $\neg\ (\textit{TA} * \textit{Teacher} * \textit{Section} * \textit{Course} * \textit{Dept\_2} \wedge \textit{Dept\_2.name} \neq \textit{"EE"})$

In this example, we use the alias variables *Dept_1* and *Dept_2* for class Department so that these two variables can both represent objects of the same class. We also

assume that the underlying data model captures the generalization or superclass-subclass semantics. Therefore, the TA instances inherit the properties (specifically, the associations) of the Student instances. We can use the expression, *TA* $*$ *Dept_1*, as a shorthand for *TA* $*$ *Grad* $*$ *Student* $*$ *Dept_1*.

Example 7. A faculty member who is not an advisor of any graduate student must teach at least one section.

Rule 7: IF $\exists$*Advising*(*Faculty* $*$ *Advising*) THEN $\exists$*Section*(*Faculty* $*$ *Section*)

or equivalently

$$\exists \textit{Advising}(\textit{Faculty} * \textit{Advising}) \vee \exists\ \textit{Section}(\textit{Faculty} * \textit{Section})$$

This rule will be violated if there is a Faculty instance that is associated with neither an Advising instance nor a Section instance. In other words, the pattern of an isolated Faculty instance is not allowed to exist in the consistent database.

From the above examples, one can see that the object calculus can express many types of constraints for OO databases. It can serve as a theoretical basis for the design of high-level constraint languages in OO databases with various syntactic sugar (i.e, syntactic constructs that make the language easier to use).

A constraint language defines a complete rule structure which usually includes the trigger condition, constraint rule body, and/or corrective action (Date, 1981; Stonebraker et al., 1987; Su and Alashqur, 1991). The rule body is the most important part of a constraint language; it specifies the constraint that the database must comply with. As shown in the above examples, our object calculus can be used to express well the constraint rules in the rule body. Because not all the system-defined and user-defined operations will change a database from a legal status to an illegal status, the constraint rules need not be checked and fired all the time. The trigger condition specifies when the rule should be checked and enforced. In addition, some corrective actions can be specified in a rule to cause actions to be taken if a constraint rule (i.e., the rule body) is violated. As a result, a complete structure of a rule in a constraint language can have the following structure:

Trigger_condition (<Trigger_time, operation> pairs)
Rule_body (the object calculus wff)
Corrective_action (procedures/methods)

An example of a complete rule is given below:

Constraint Rule: After update Grad, TA, or RA, a graduate student cannot be both a TA and an RA at the same time. If the rule is violated, print a message to warn the user.

Trigger_condition (After update(Grad), After update(TA), After update(RA))
(IF *Grad* $*$ *TA* THEN $\forall$ *RA*(*Grad* ! *RA*)) $\wedge$

(IF *Grad* $*$ *RA* THEN $\forall$ *TA*(*Grad* ! *TA*))

Corrective_action (Message: ("A graduate student cannot be both a TA and an RA"))

6. System Implementation Status

An object-oriented knowledge base management system (OSAM*.KBMS) has been designed and implemented in the Database Systems Research and Development Center at the University of Florida. In the system, a high-level query language, called OQL (Alashqur et al., 1989), and a high-level constraint language (Su and Alashqur, 1991) have been developed based on the object-oriented view of databases as patterns of object and class associations (although its development had preceded the development of the object calculus). The OQL is a non-procedural OO query language. It is an association- or pattern-based query language instead of a more traditional attribute-based query language. It allows data search conditions of different degrees of complexity to be specified uniformly and simply as patterns of object class associations instead of comparing values of keys and foreign keys.

The high-level constraint language has trigger and rule specification constructs for declaratively specifying semantic constraints. In a constraint rule, the association pattern specification is used so that a complex pattern of object associations can be stated as the condition for activating some operations. In addition to the association pattern specified in the constraint rule, the activation of a rule also can be made subject to various trigger conditions and time constraints as explained in the previous section.

Both OQL and the constraint language have been incorporated in a knowledge programming language called K (Shyy and Su, 1991; Arroyo, 1992). The processing of queries and knowledge rules is supported by persistent knowledge bases. In the present implementation, K programs are translated in C++ code. During execution, the translated programs make calls to the knowledge-based management system to retrieve, store or update the persistent data, to make use of the trigger and rule processing facility, and to make use of other object management functions. The object manager is built on top of the commercial system ONTOS which provides all the needed storage management functions. A prototype system has been demonstrated in an international conference (Su et al., 1993).

7. Conclusion

In this article, we introduced a pattern-based object calculus which incorporates the concept of association pattern specification into the first-order predicate calculus. This incorporation allows complex patterns of object associations to be specified in wffs, thus greatly increasing the expressive power of the calculus. Specifically, the greater expressive power of the object calculus over that of the relational

calculus can be shown in three aspects. First, based on our OO view of databases, relationships between objects (or tuples, in the relational model) are expressed explicitly in the database. Matching of keys and foreign keys in calculus expressions is not required; thus the expressions for complex queries are simplified. Second, the "nonassociation" operator, whose semantics are not easily expressible in relational calculus or other languages, is included explicitly in the object calculus. Third, this calculus allows set-oriented operations to be performed on both homogeneous and heterogeneous object association patterns, whereas set operations in the relational model can only be performed on union-compatible (i.e., homogeneous) relations. Furthermore, the object calculus can be used in a uniform fashion to express search queries involving complex object association patterns and to express various kinds of semantic constraints and deductive rules in OO databases. Expressions of this object calculus are safe, and the closure property is also preserved. One of the potential applications of the proposed formalism is that its precise interpretive semantics can serve as the theoretical foundation for designing high-level OO query languages and constraint languages.

References

Alashqur, A.M., Su, S.Y.W., and Lam, H. OQL: An object-oriented Query Language. *Proceedings of the Fifth International Conference on Very Large Databases,* Amsterdam, 1989.

Arroyo, J. The design and implementation of K.1: A next-generation knowledge base programming language. Master's Thesis, Department of Electrical Engineering, University of Florida, Gainesville, 1992.

Banerjee, J., Chou, H.-T., Garza, J.F., Kim, W., Woelk, D., Ballon, N., and Kim, H.-J. Data model issues for object-oriented applications. *ACM Transactions on Office Information Systems,* 5(1):3-26, 1987.

Batory, D. and Kim, W. Modeling concepts for VLSI CAD objects. *ACM Transactions on Database Systems,* 10(3):322-346, 1985.

Carey, M.J., DeWitt, D.J., and Vandenberg, S.L. A data model and query language for EXODUS. *Proceedings of the ACM SIGMOD Conference,* Chicago, 1988.

Chen, W. and Warren, D.S. C-logic for complex objects. *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,* Philadelphia, PA, 1989.

Codd, E. Relational completeness of database sublanguages," In: Rystin, R., ed., *Database Systems, Courant Computer Science Symposia Series,* Vol. 6, Englewood Cliffs, NJ: Prentice-Hall, 1972.

Codd, E. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems,* 4(4):397-434, 1979.

Date, C. Referential integrity. *Proceedings of the Seventh International Conference on Very Large Data Bases,* Cannes, France, 1981.

Date, C. *An Introduction to Database Systems.* 3rd ed. Reading, MA: Addison-Wesley Publishing Company, 1982.

Fishman, D.H., Beach, D., Cate, H.P., Chow, E.C., Connors, T., Davis, J.W., Derrett, N., Hoch, C.G., Kent, W., Lyngback, P., Mahbod, B., Neimat, M.A., Ryan, T.A., and Shan, M.C. Iris: An object-oriented database management system. *ACM Transactions on Office Information Systems,* 5(1):48-69, 1987.

Guo, M., Su, S.Y.W., and Lam, H. An aAssociation algebra for processing object-oriented databases. *Proceedings of the Seventh International Conference on Data Engineering,* Kobe, Japan, 1991.

Hull, R. and King, R. Semantic database modeling: Survey, application, and research issues. *ACM Computing Surveys,* 19(3):201-260, 1987.

Kifer, M. and Lausen, G. F-Logic: A higher-order language for reasoning about objects, inheritance, and scheme. *ACM SIGMOD Record,* 18(2):134-146, 1989.

Kifer, M. and Wu, J. A logic for object-oriented logic programming (Maier's O-logic revisited). *Proceedings of the ACM-SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,* Philadelphia, PA, 1989.

Lecluse, C., Richard, P., and Velez, F. O_2, an object-oriented data model. *Proceedings of the ACM SIGMOD Conference,* Chicago, 1988.

Maier, D. A logic for objects. *Proceedings of the Workshop on Foundations of Deductive Databases and Logic Programming,* Washington D.C., 1986.

Maier, D., Stein, J., Otis, A., and Purdy, A. Development of an object-oriented DBMS. *Proceedings of the First OOPSLA Conference,* Portland, OR, 1986.

Manola, F. and Dayal, U. PDM: An object-oriented data model. *Proceedings of the First International Workshop on Object-Oriented Database Systems,* Asilomar, CA, 1986.

Morgenstern, M. Constraint equations: Declarative expression of constraints with automatic enforcement. *Proceedings of the Tenth International Conference on Very Large Data Bases,* Singapore, 1984.

Object Design, Inc. *ObjectStore Reference Manual,* Object Design, Inc., Burlington, MA, October, 1990.

Ontologic, Inc., "Vbase, Integrated Object Database," A set of user manuals, Ontologic, Inc., Billerica, MA., 1988.

Osborn, S.L. Identity, Equality and Query Optimization. In: Dittrich, K.R., ed., *Advances in Object-Oriented Database Systems,* New York: Springer-Verlag, 1988, pp. 346–351.

Osborn, S.L. Algebraic query optimization for an object algebra. *Technical Report No. 251,* University of Western Ontario, 1989.

Schaffert, C., Cooper, T., Bullis, B., Kilian, M., and Wilpolt, C. An introduction to Trellis/Owl. *Proceedings of the First OOPSLA Conference,* Portland, OR, 1986.

Shaw, G.M. and Zdonik, S.B. An object-oriented query algebra. *Bulletin of the IEEE Technical Committee on Data Engineering,* 12(3):29-36, 1989.

Shepherd, A. and Kerschberg, L. PRISM: A knowledge-based system for semantic integrity specification and enforcement in database systems. *Proceedings of the*

ACM SIGMOD Conference, Boston, MA., 1984.

Shipman, D. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems,* 6(1):140-173, 1981.

Shyy, Y.-M. and Su, S.Y.W. K: A high-level knowledge-based programming language for advanced database application. *Proceedings of the ACM SIGMOD Conference,* Denver, Colorado, 1991.

Stonebraker, M., Hanson, E., and Hong, C. The design of the POSTGRESS rule system. *Proceedings of the Third International Conference on Data Engineering,* Los Angeles, 1987.

Straube, D.D. Query and query processing in object-oriented database systems. Ph.D. Thesis, Dept. of Computer Science, University of Alberta, 1991.

Su, S.Y.W. and Alashqur, A.M. A pattern-based constraint specification language for object-oriented databases. *Proceedings of COMPCON,* San Francisco, 1991.

Su, S.Y.W., Krishnamurthy, V., and Lam, H. An object-oriented semantic association model (OSAM*). In: Kumara, S., Soyster, A., and Kashyap, R., eds., *Artificial Intelligence: Manufacturing and Practice,* Norcross, GA: The Institute of Industrial Engineers, Industrial Engineering and Management Press, 1989, pp. 463-494.

Su, S.Y.W., Guo, M., and Lam, H. Association algebra: A mathematical foundation for object-oriented databases. *IEEE Transactions on Knowledge and Data Engineering,* 5(5):775-798, 1993.

Su, S.Y.W., Lam, H., Eddula, S., Arroyo, J., Prasad, N., and Zhuang, R. OSAM*. KBMS: An object-oriented knowledge base management system for supporting advanced applications. *Proceedings of ACM SIGMOD Conference,* Washington D.C., 1993.

Ullman, J.D. *Principles of Database Systems,* Reading, MA: Computer Science Press, 1982.

Urban, S.D. and Delcambre, L.M.L. Constraint analysis for specifying perspectives of class objects. *Proceedings of the Fifth International Conference on Data Engineering,* Los Angeles, CA., 1989.

Zaniolo, C. The Database Language GEM. *Proceedings of ACM SIGMOD Conference,* San Jose, CA, 1983.