OQL: A Query Language for Manipulating Object-oriented Databases

A. M. Alashqur
S.Y.W. Su
H. Lam

Database Systems Research and Development Center
Electrical Engineering Department
University of Florida

Abstract

An essential property which is desirable in a query language designed for a certain data model is that queries issued in that language must produce results that are structured and modeled using the same data model. A consequence of maintaining this property in a query language is that the result of a query can be used as an operand in some other query (or queries) or can be saved as a user's view. Existing query languages that have been designed for the class of object-oriented data models do not posses this property. In this paper, we introduce the object-oriented query language (OQL), which maintains this property. An OQL query is considered as a function, which when applied to a database, returns a subdatabase whose structure consists of some selected object classes and their associations. The objects that satisfy the search conditions and participate in the patterns of object associations specified in the query constitute the extension of the resulting subdatabase. A subdatabase forms a "context" under which system-defined and/or user-defined operations can be specified and performed. Several advanced features such as branching association patterns and set operations on subdatabases are also presented.

1. Introduction

The limitations of the existing record-oriented data models have long been observed [HAM81,HUL87,SU88]. To alleviate these limitations, several object-oriented (OO) and semantic data models have been introduced as the potential alternatives for modeling many advanced database applications such as CAD/CAM, office automation, and multimedia databases [HAM81,KIN84,BAT85,SU86,HUL87]. OO and semantic data models can capture much more of the semantics of these application domains in a "natural" way. They provide a rich variety of modeling constructs, which can model most of the situations that may arise in such application domains.

The term "OO data model" is used to refer to a data model that is a structurally and/or behaviorally object-oriented [DIT86]. A structurally OO data model is one that encompasses at least the following characteristics:

1. It allows for defining aggregation hierarchies.
2. It allows for defining generalization hierarchies.
3. It supports the unique identification of objects, that is, each object is assumed to have a unique object identifier (surrogate).

The OO view of an application world is represented in the form of a network of classes and associations, which can be aggregation or generalization associations.

Object classes can be either **primitive** classes whose instances are of simple data types (e.g., integer, string, real) or **non-primitive** classes whose instances represent real world objects (e.g., Part, Employee). At the extensional level, instances of different classes can be related (associated) with each other forming patterns of object associations. A behaviorally object-oriented data model, on the other hand, is one in which operations that describe the behavior of the objects of a class can be defined and registered with that class.

Several query languages such as DAPLEX [SHI81], GEM [ZAN83], ARIEL [MAC85], and the object-oriented query language described in [BAN88], which are based on the OO view of data as described above (or variations of it), have been introduced in the literature. A query in these languages is performed by choosing one of the non-primitive classes in the schema as a central class (anchor class). The user then specifies some path expressions such that each path expression starts from the central class and ends up at a primitive class. A restriction condition can be specified on the class referenced at the end of a path expression and/or such a class can be specified in the target list (i.e., the list of attributes to be retrieved). The result of a query is defined as a set of tuples (i.e., a relation) each of which corresponds to a single instance of the central class and contains values related to that instance which are collected from the primitive classes specified in the target list.

These languages share the above characteristic even though they are based on different paradigms. For example, DAPLEX is based on the functional paradigm. An attribute of a class in DAPLEX is treated as a function, which when applied to an object of the class, returns its attribute value from the related class. Path expressions starting from a central class can be specified by composing functions. The query language of [BAN88], on the other hand, is based on the message passing paradigm. In this language, an attribute of a class is treated as a message, which when sent to an object of the class, returns its attribute value from the related class. Path expressions starting from a central class can be specified using a series of messages each of which is sent to the object returned by the previous message in the series.

A major drawback of existing data manipulation languages for OO and semantic models is that they do not maintain the closure property because the result of a query does not have the same structural properties as those of the original database. In these languages, the input to a query has an OO representation (i.e., classes and their associations) and its output is a relation. Consequently, the result of a query cannot be further operated on uniformly using the same query language operators to produce a new result. Contrary to these languages, the relational query languages such as SQL, QUEL, or the relational algebra maintain the closure property. A relational query can be a single-relation query or a multi-relation one. In either case, the result of a query is always a relation. In other words, both of the input

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

and output of a query have a relational representation. Thus, the resulting relation can be uniformly operated on by another query in a nested fashion or can be saved as a view definition and manipulated uniformly as any of the base relations.

Motivated by this limitation of the existing OO query languages, we have developed a new OO query language called OQL, which maintains the closure property. A query in this language, can be considered as a function that, when applied to a database, returns a **subdatabase** whose structure is comprised of some selected classes of objects and their associations, i.e., it has the same structural characteristics as those of the original database. For this reason, a resulting subdatabase can be further operated on by another OQL query to produce another subdatabase or it can be saved as a view definition. The instances (objects) that satisfy the search conditions and participate in the **patterns of object associations** specified in the query (see Section 3) form the extension of the resulting subdatabase.

Another shortcoming of existing OO query languages is that they are oriented towards retrieval and storage manipulation operations (i.e., system-defined operations) in the sense that a query specifies data to be manipulated by such operations as Retrieve, Update, and Delete. In a behaviorally OO data model, user-defined operations that describe the behavior of object classes can be defined and registered with these classes. In OQL, a subdatabase represents a "context" in which objects of various classes must exist before being considered for further manipulation by different operations including system-defined operations (e.g., Update, Display) as well as user-defined operations (e.g., Hire-employee, Move). Multiple operations (messages) can be issued against the different classes of a subdatabase in the same query.

This paper introduces the OQL language and is organized as follows. In Section 2, we describe the OO view of a university database as represented by an OO model. In Section 3, the concept of semantic association pattern is introduced together with the basic operators used for defining a subdatabase. Section 4 presents some advanced features of OQL. The assignment operator of OQL and the closure property are described in Section 5. Finally, some conclusions OQL are summarized in Section 6.

2. The Object-oriented View of Databases

We shall describe first the OO view of a university database as modeled by the OO semantic association model OSAM* [SU88]. The university schema shown in Figure 2.1 is then used in the remainder of this paper to issue some example OQL queries. The concepts introduced in OQL can be applied to any OO data model that encompasses the three structural properties described in Section 1 and are not limited to the OSAM* model.

A database schema is represented in OSAM* as a network of associated (inter-related) object classes. Graphically, object classes are represented as nodes and associations among object classes are represented as links. The resulting diagram is called the Semantic diagram or **S-diagram**. In OSAM*, there are two types of object classes: Entity object classes (**E-class**) and Domain object classes (**D-class**) which are represented in the S-diagram as rectangular and circular nodes, respectively. The sole function of a D-class is to form a domain of possible values (e.g., integers, strings, etc.) from which descriptive attributes of objects draw their values. An E-class, on the other hand, forms a domain of objects which occur in an application's world (e.g., Faculty, Department, etc.). Each object of an E-class is represented by a unique object identifier (OID).

There are five types of links (associations) in OSAM*. Two of these association types appear in Figure 2.1, namely, Aggregation (A) and Generalization (G), which are also recognized in several other semantic and OO models. A class can have several types of links and more than one link of each type emanating from it. In the S-diagram, links of the same type that emanate from a class are grouped together and labeled by the letter that denotes the association type.

As an example, in Figure 2.1, the E-class Person has two types of links: Aggregation links with the D-classes SS# and Name and Generalization links with the E-classes Student and Teacher (i.e., Student and Teacher are subclasses of the superclass Person). An aggregation link represents an attribute and has the same name as the class it connects to unless specified otherwise (e.g., the link labeled Major that emanates from the class Student). Aggregation links that emanate from an E-class and connect to D-classes represent the descriptive attributes of that class (e.g., the attribute section# of the class Section). A class inherits all the aggregation associations that connect to or emanate from its superclasses. Figure 2.2 shows the actual view of the class RA in which all the associations inherited by RA are explicitly represented. A detailed description of the OSAM* model can be found in [SU88].

3. Object-oriented Query Language OQL

In this section, the general OQL syntax and semantics are described and illustrative examples are given. Throughout this paper, capital letters are used to denote E-classes (A, B,...) and small letters with an integer appended to each letter are used to denote objects (or OIDs) (e.g., a1, a2, ... and b1, b2, ... are OIDs for objects that belong to the classes A and B, respectively).

3.1 Definitions and Overview

An OQL query returns a subdatabase, which is a portion of the operand database (an operand database can be the original database or another subdatabase that has been established by another query). Figure 3.1 shows a certain subdatabase SDB of the original database of Figure 2.1. A subdatabase consists of two parts: an **intensional association pattern** and a set of **extensional association patterns**. An intensional association pattern is represented as a network of E-classes and their associations and descriptive attributes. For example, Figure 3.1a represents the intensional association pattern of a the subdatabase SDB, which, in this simple case, has a linear structure that consists of the classes Teacher, Section, and Course and their associations. By default, all the descriptive attributes of the E-classes of a subdatabase are also present in the subdatabase unless specified otherwise in a Select subclause to be described later. In Figure 3.1, however, the descriptive attributes of the classes of SDB are not shown in order to keep the figure simple.

An extensional association pattern is a network of instances and their associations that belong to the classes and association types of the intensional association pattern. The set of extensional patterns of a subdatabase can be represented in the form of an **extensional diagram**. Figure 3.1b shows a possible extensional diagram for the subdatabase SDB. (We note that Section s3 is related to more than one Course instance and Section s4 is not related to any Course instance. Naturally, there is a constraint on the database that restricts the mapping between Section and Course to N:1 and another Non-null constraint on the aggregation association of Course with Section. We assume

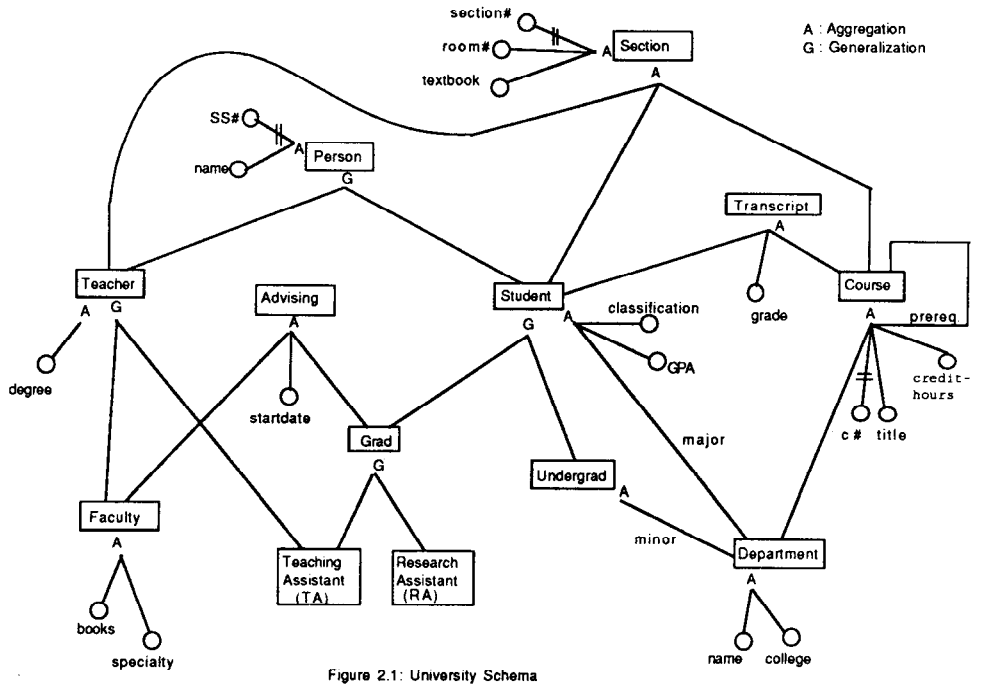


Figure 2.1: University Schema

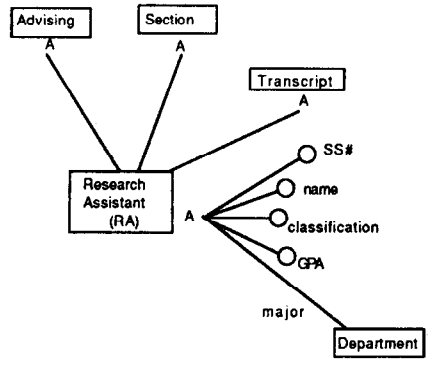


Figure 2.2: Class RA with all the Inherited Associations Explicitly Represented

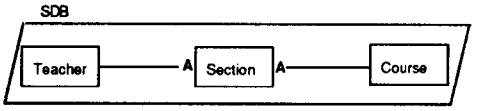


Figure 3.1a: The Intensional Pattern of a Subdatabase SDB

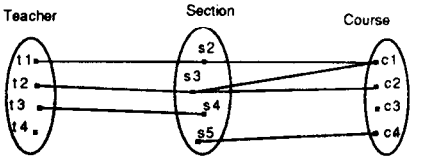


Figure 3.1b: A Possible Extensional Diagram for the Subdatabase SDB

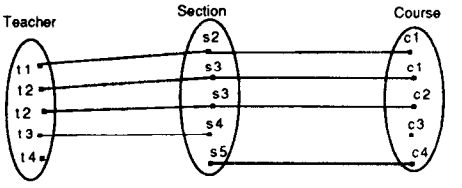


Figure 3.1c: A Normalized Extensional Diagram Corresponding to Fig. 3.1b

Figure 3.1 The Intensional and set of Extensional Patterns of a Subdatabase

that these constraints are waived here in order to be able to describe the most general case.) The interconnection of t3 and s4 in the figure is an example of an extensional pattern, which records the fact that object t3 is associated with object s4 (Teacher t3 is teaching Section s4).

A **normalized extensional diagram** is an extensional diagram in which an object may appear more than once depending on the number of associations it has with the objects of a neighboring class (in case of a multi-valued association) and a separate link is used to connect it to each object of the neighboring class. Figure 3.1c shows the normalized extensional diagram for the same subdatabase. In the remainder of this paper, we shall deal with normalized extensional diagrams only. In addition to the graphical representation, an extensional pattern may be represented as a tuple of OIDs. For example, <t1,s2,c1>, <c3> and <s5,c4> are some of the extensional patterns that appear in the subdatabase SDB (Figure 3.1c).

We define an **extensional pattern type** as the common template that is shared by several extensional patterns in a subdatabase. A pattern type is denoted by a tuple of class names. For example, <Teacher, Section, Course> is one of the extensional pattern types that exist in Figure 3.1c, which has as instances all the extensional patterns that contain Teacher, Section, and Course objects, i.e., the extensional patterns <t1,s2,c1>, <t2,s3,c1>, and <t2,s3,c2>. On the other hand, the extensional pattern <t3,s4> whose Course-component is Null (since the pattern does not contain any Course object) is of the type <Teacher,Section>. The five extensional pattern types present in the extensional diagram of Figure 3.1c are <Teacher,Section, Course>, <Teacher,Section>, <Section,Course>, <Teacher>, and <Course>.

The philosophy underlying OQL is to allow the user to specify, first, the desired subdatabase by specifying its intensional pattern and the set of extensional pattern types that are of interest and then the operation(s) to be performed on the classes of the subdatabase. The search engine of the underlying OO DBMS would establish the subdatabase by identifying all the extensional patterns that belong to the specified types and then perform the operation(s).

A query block in OQL consists of a Context clause and an Operation clause. The Context clause has two optional subclauses: a Where subclause and a Select subclause. This structure is shown below.

context association pattern expression
  where conditions
  select object classes and/or attributes
operation(s) object class(es)

In the context clause, the user specifies a desired subdatabase by specifying its intensional pattern and extensional pattern types of interest (both are specified in the association pattern expression). A linear association pattern expression has the form "A [intra-class conditions] **op** B [intra-class conditions] **op** C [intra-class conditions] ..." where "op" is one of the association pattern operators to be described in Sections 3.2 and 3.3. Each operator separates two E-classes that are directly associated in a schema. More complex association pattern expressions that contain branching are described in Section 4. The intra-class conditions enclosed in brackets following a class name are optional and are expressed in the form of predicates that involve the descriptive attributes of that class.

The Where subclause further causes the extensional patterns that do not satisfy some conditions to be dropped from the Context subdatabase. The conditions that can be specified in the Where subclause are inter-class comparison conditions, which are comparisons between some attributes of two classes if these attributes are type comparable, and/or comparisons between objects (equal '=' or not equal '!=') if these objects are type comparable (i.e., belonging to the same E-class or any of its superclasses or subclasses).

The Select subclause operates on the subdatabase returned by the Context clause and its optional Where subclause to produce a new subdatabase which results from "projecting" the Context subdatabase over some classes and descriptive attributes. A resulting subdatabase can be either saved for further processing (Section 5) using the same association pattern operators of OQL (since the closure property is maintained) or operated on by the operation(s) specified in an operation clause. We note that the Select operation in OQL does not imply displaying some data to the user as in SQL.

The Operation clause specifies a set of messages (operation names) to be sent to the classes of the subdatabase that is returned by the Context expression. Each message may be followed by one or more arguments that identify the recipient classes. Thus, several operations can be performed over the same or different classes and a single operation can be performed over several classes resulting from a Context clause. An operation can be either a system-defined data manipulation operation (e.g., Display, Update, Print) or a user-defined operation (e.g., Rotate, Order-part, Hire-employee).

If the Display (Print) operation is specified in the operation clause, it causes the values of the descriptive attributes that appear in the subdatabase to be displayed (printed) to the user in the form of a table. If the Display operation is not followed by a class name as an argument, the resulting table will be a first normal form table defined over all identified attributes. Otherwise, the argument class identifies a "viewpoint" based on which the resulting descriptive data is to be organized in the form of a non-normalized table (i.e., a table in which a value can be a nested table or a set). Thus, the descriptive data are structured under the objects of the argument class.

The operators that can be used in the association pattern expression of the Context clause are the **association operator** and the **non-association operator**.

3.2 The Association Operator

When the association operator (*) is applied to two directly associated E-classes A and B in a database (i.e., the expression "A * B"), it returns a subdatabase whose intensional pattern consists of the two classes A and B and their association. The resulting subdatabase contains also the set of extensional patterns drawn from the operand database such that each extensional pattern contains objects of both A and B (i.e, extensional patterns that are of the type <A,B>). B objects that are not associated with any A objects and A objects that are not associated with any B objects in the operand database are not retained in the resulting subdatabase. The following example queries illustrate the use of the association operator.

Query 3.1 Display the names of the teachers who teach some sections and the section#'s for these sections.

context Teacher * Section

```
    select name, section#
display Teacher
```

If the Context expression in this query is applied to the subdatabase SDB of Figure 3.1, it returns a new subdatabase whose set of extensional patterns is [<t1,s2>, <t2,s3>, <t3,s4>]. The extensional pattern <t4> (or <t4, Null>) is not included in this set because its Section-component is Null (similarly the pattern <s5> is not included). Figure 3.2a shows the intensional pattern of this subdatabase where all the descriptive attributes of the classes Section and Teacher appear with them (i.e., by default). The Select subclause derives a new subdatabase from the Context subdatabase. The intensional pattern of the new subdatabase, as shown in Figure 3.2b, consists of the classes Teacher and Section and only the two attributes that are referenced in the Select subclause, namely, Name of Teacher and Section# of Section. If all the descriptive attributes of a class are to be retained in the subdatabase the following rule can be applied.

Rule #1 If all the descriptive attributes of a class are to be retained in the subdatabase derived by the Select subclause, the class name can be referenced in the subclause without specifying any of its attributes, i.e., the default is "all attributes."

A Select subclause can also select a subset of E-classes, thus producing a subdatabase that results from dropping the unreferenced E-classes from the operand subdatabase. (A class is considered "unreferenced" in a Select subclause if none of its descriptive attributes is referenced in it.) If a class to be dropped from the operand subdatabase connects two classes that are to be retained, new direct association is created between these two retained classes in the produced subdatabase. At the extensional level, direct links between the instances of the two classes are inferred. For example, the subdatabase that results from selecting the classes Teacher and Course of the subdatabase SDB of Figure 3.1 is shown in Figure 3.3.

The display operation in the above query causes the result to be displayed in a non-normalized tabular form in which each tuple consists of a teacher's name and the set of section#'s for the sections he/she teaches. This is because the argument of the Display operation indicates that the result is to be viewed from the point of view of the class Teacher. We note that the result of a Display operation does not belong to the world of subdatabases and therefore cannot be operated on using the OQL operators. However, the subdatabase that corresponds to a certain displayed result can be further operated on to produce a new subdatabase whose descriptive attributes can be also displayed.

The definition of the association operator can be easily generalized to the case when the association pattern expression contains more than two classes. For example, the expression "A * B * C" returns the extensional patterns that are of the type <A,B,C>. It is noted here that one can define a single extensional pattern type using the association operator. A mechanism for defining a richer variety of extensional pattern types in a single expression is described in Section 3.4.

Query 3.2 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. In addition, print the results.

```
context Department * Course [6000 <= c# < 7000] *
        Section
    select name, title, textbook
display
print
```

Two operations are specified in the Operation clause of this query, namely, Display and Print. These operations are to be performed on the subdatabase returned by the Select subclause. Also, the intra-class condition on the C# attribute of Course is enclosed in brackets following the class name in the Context expression. The result of the Display or Print operation is a normalized table since neither of the two operations is followed by a viewpoint class.

As in Query 3.3 below, an association (or non-association) operator can be used between any two classes whether they are connected by a generalization or an aggregation association. An association pattern expression concerns only with whether some classes and their instances are associated with one another or not. It does not specify what types of associations relate them. This is because the types of the associations connecting them have already been explicitly defined in the schema and restating these association types in queries is unnecessary. The query processor of an OO DBMS can make use of the type information stored in the dictionary to properly interpret the queries and enforce the relevant semantics and constraints. For example, a link that exists between an instance of the class TA and an instance of the class Grad is an identity link. In other words, the semantics implied by the generalization association here is that the two instances are actually two different perspectives of the same real world object.

Before presenting the example query 3.3, we introduce the following two general rules that are relevant to the query.

Rule #2 An attribute that appears in the Select or Where subclause has to be qualified by its class name only if it is not unique among the attributes of the classes that are referenced in the Context clause.

Rule #3 Different aliases (range or iteration variables) of a class can be generated in OQL by appending an Underscore and an integer to the class name in an association pattern expression (e.g., Grad_1 is and alias of Grad).

Query 3.3 Print the names of graduate students who teach other graduate students in some sections. Also, print the names of those graduate students they teach. Organize the result from the point of view of the teaching graduate students.

```
context Grad_1 * TA * Teacher * Section *
        Student * Grad_2
    select Grad_1 [name], Grad_2 [name]
print Grad_1
```

In this query, TA inherits the status of being related to Section from both Teacher and Student with each of them having its distinctive meaning. Thus, using the expression "TA * Teacher * Section" instead of the expression "TA * Section" is to explicitly state that we are interested in TA as playing the role of Teacher rather than the role of Student of a section. Also, in this query, the Select subclause projects over the two classes Grad_1 and Grad_2 since they are the only classes referenced in it. Thus, the intensional pattern of the final subdatabase contains these two classes with a derived aggregation association between them and the attribute Name of each class.

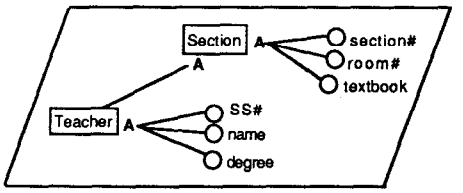


Figure 3.2a: The Result of a Context Expression of a Query

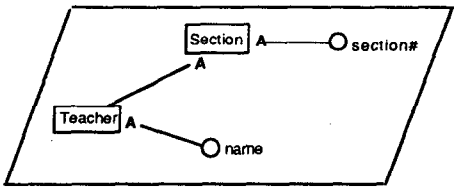


Figure 3.2b: The Projection of the Subdatabase of Fig.3.2a over some Descriptive Attributes

Figure 3.2: The Effect of a Select Subclause

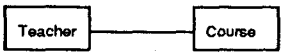


Figure 3.3a: After Projecting the Subdatabase of Figure 3.1 over Teacher and Section

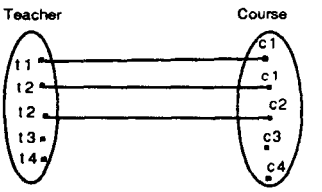


Figure 3.3b: The Extensional Diagram Corresponding to Figure 3.3a

Figure 3.3: The Result of a Projection Operation at the Intensional and Extensional Levels

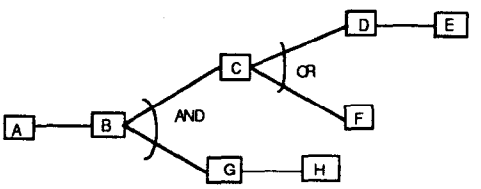


Figure 4.1: A Graphical Representation of a Branching Association Pattern

3.3 The Non-association Operator

We use the exclamation sign (!) to denote this operator. When this operator is applied to two directly associated E-classes A and B in a schema (i.e., the expression "A ! B"), it returns a subdatabase which contains only the instances of A that are not associated with any instances of B and the instances of B that are not associated with any instances of A. For example, the two instances t4 and s5 are returned in the subdatabase that results when the expression "Teacher ! Section" is applied to the subdatabase SDB of Figure 3.1 (i.e., it returns the teachers who are not assigned to any sections and the sections which are not assigned to any teacher).

The association operator has higher precedence than the non-association operator. As an example, when the association pattern expression "Teacher ! Section * Course" is applied to the classes of the subdatabase SDB of Figure 3.1, it produces a new subdatabase that contains the following set of extensional patterns: [⟨s5,c4⟩, ⟨t3⟩, ⟨t4⟩]. The pattern ⟨t3⟩ is retained in this result because of the higher precedence of the association operator over the non-association operator. When the association operator is applied first, object s4 is not retained in the result (since it is not associated with any Course object) causing object t3 to be not associated with any Section object in this result. When the non-association operator is applied next, it causes all the Teacher objects that are associated with any pattern of the type ⟨Section,Course⟩ (i.e., the objects t1 and t2) to be dropped together with these ⟨Section,Course⟩ patterns. Thus, the final result contains the above set of extensional patterns. The precedence of the association operator over the non-association operator can be overridden by parentheses. The following is an example query that uses the non-association operator.

Query 3.4 Display the names of those graduate students who are TA's but not RA's.

```
context TA * Grad ! RA
   select TA [name]
display
```

3.4 Association Pattern Subexpressions

By using only the association operator in an association pattern expression, one can identify a single extensional pattern type. In some situations, extensional patterns of different types may be desired in the resulting subdatabase. This can be performed in OQL by enclosing a subexpression of the association pattern expression inside braces. This subexpression identifies a certain extensional pattern type. For example, the expression "A * {B * C} * D" returns the subdatabase whose intensional pattern consists of these four classes and whose set of extensional patterns includes all patterns that are of the types ⟨A,B,C,D⟩ and ⟨B,C⟩. In other words, this expression means to select both the instances of A,B,C and D classes that are connected (associated) all the way through as well as those instances of B and C that are connected to each other but not necessarily connected to the instances of A and/or B. The braces around B * C capture the semantics of the Outerjoin concept [COD79].

If, in the above expression, an extensional pattern of the type ⟨B,C⟩ appears in the resulting subdatabase as part of a larger pattern of the type ⟨A,B,C,D⟩, it will not appear independently in that resulting subdatabase. For example, if the original database contains only the two patterns ⟨a1,b5,c5,d5⟩ and ⟨a3,b2,c2⟩, then the expression "A * {B * C} * D" returns the extensional patterns ⟨a1,b5,c5,d5⟩ and ⟨b2,c2⟩. The extensional pattern ⟨b5,c5⟩ will not appear independently in the result since it already appears as a part of the extensional pattern ⟨a1,b5,c5,d5⟩. In general, an extensional pattern of a certain specified type will not appear independently in the result, if it is part of a larger extensional pattern.

Subexpressions can be nested to several levels. For example, the expression "{{{A} * B} * C} * D" identifies the extensional pattern types ⟨A⟩, ⟨A,B⟩, ⟨A,B,C⟩, and ⟨A,B,C,D⟩.

Query 3.5 Display the SS#'s of all graduate students (whether they have advisors or not) and for those graduate students who have advisors, display their advisors' names.

```
context [ Grad ] * Advising * Faculty
   select Grad [SS#], Faculty [name]
display
```

The attribute Name in the Select subclause of this query is qualified by its class name because it is not unique among the classes referenced in the Context clause (see Rule #2 above) and similarly the attribute SS#.

4. Advanced Features of OQL

4.1 Branching Association Patterns

An association pattern expression may contain branches expressed by an AND or an OR operator. There can be several nested levels of branching. For example, the expression "A * B * **AND** (C * **OR** (D * E, F), G * H)" is a branching association pattern expression, which corresponds to the intensional pattern shown in Figure 4.1. A class at which the branching occurs is called a **fork class** (e.g., B and C in the above expression). An AND operator means that, in the result, an instance from the fork class must be associated with instances from both of the branches, while an OR operator means that an instance from the fork class must be associated with an instance from at least one of the two branches. Figure 4.2 shows some association pattern expressions that represent networks of classes and associations together with a graphical representation of the extensional pattern types they define.

Query 4.1 Print the name of any faculty member who is teaching any section of a course that is offered by the 'EE' department, provided that the section is taken by at least one graduate student who is an RA. Also, print the c# value(s).

```
context Faculty * Section *
            and (Course * Department [name = 'EE'], RA)
   select Faculty [name], c#
Print
```

OQL makes full use of the inheritance property of the generalization association. In this query, Faculty and RA inherit the association to Section from their superclasses. Hence, using the expression "Faculty * Section" instead of "Faculty * Teacher * Section" and the expression "Section * RA" instead of "Section * Student * Grad * RA" is legal.

4.2 Set Operators

The set operators Union, Intersection, and Difference can be applied to any two union-compatible subdatabases to produce a new subdatabase. Two subdatabases are said to be **union-compatible** if both of them have the same intensional association pattern defined in terms of E-classes only and irrespective of the D-classes that may

exist in the subdatabases. Thus, the following is a legal format for a query.

```
context A * (B * C * D)
        union
        (A * B * C) * D
   select <classes and attributes>
```

The first argument to the Union operator returns a subdatabase whose extensional patterns are of the types <A,B,C,D> and <B,C,D> while the second argument returns a subdatabase whose extensional patterns are of the types <A,B,C,D> and <A,B,C>. However, both subdatabases have the same intensional pattern that consists of the four classes A, B, C, and D and their associations. The result of the Union operation is a subdatabase that contains extensional patterns of the three types <A,B,C,D>, <B,C,D>, and <A,B,C>, i.e., it contains the set of all extensional patterns that belong to either the first or second subdatabase. The Select subclause is then applied to the resulting subdatabase. The Difference operator returns a subdatabase that contains the set of all extensional patterns that belong to the first subdatabase but not to the second subdatabase. The subdatabase returned by the Intersection operator contains the set of extensional patterns that belong to both subdatabases.

Set operators can be used to create views in which some descriptive attribute values of objects do not appear with them in the view unless these objects participate in certain patterns of associations (e.g., for security reasons). This is illustrated by the following query, which produces a subdatabase that can be saved as a user's view (Section 5 describes how views can be defined and saved).

```
context Teacher * Section * Course [c# >= 5000]
   select name, degree, section#
union
context Teacher * Section * Student * Grad
   select Teacher [name, SS#], section#
```

The two arguments of the Union operator are two OQL queries that return two union-compatible subdatabases (since the Select subclause in each query projects over the two classes Teacher and Section, i.e., both subdatabases have the same intensional pattern). This means that each of the two queries returns extensional patterns of the type <Teacher, Section> but each query derives these patterns based on different conditions. The class Teacher in the first subdatabase has the descriptive attributes Name and Degree, while in the second subdatabase it has the descriptive attributes Name and SS#. The intensional patterns of these two subdatabases are shown in Figure 4.3a. In the final subdatabase, i.e., the result of the Union operation, only the Teacher instances that appeared in both operand subdatabases will have values for all the three attributes Name, Degree, and SS#. Those Teacher instances that appeared in the first subdatabase but not in the second subdatabase shall have Null values for the SS# attribute in the final subdatabase. The same goes for those Teacher instances that appeared in the second subdatabase but not in the first subdatabase with respect to Degree values. Figure 4.3b shows the intensional pattern of the final subdatabase in which the generic class Teacher has the attribute Name and each of the two subclasses Teacher_1 and Teacher_2 has one of the other two attributes. The instances of the Teacher_1 and Teacher_2 subclasses are those derived from the first and second operands of the Union operator, respectively.

Note that the non-association operator of OQL can be defined in terms of the Association and Difference operators and using braces as follows.

```
A ! B = (A) * (B) - A * B
```

Where "-" stands for the set difference operator described above. The following equation shows the equivalent expression to an expression that contains both an association operator and a non-association operator.

```
A ! B * C = (A) * (B * C) - A * B * C
```

Though the non-association operator can be defined in terms of other OQL operators, it is provided as a shorter notation for ease of use.

4.3 Queries with Multiple Expressions

In OQL, one can specify comparison conditions between attribute values or between objects of two different classes that appear in two different association pattern expressions. In this case, the two expressions shall be separated by a comma in the Context clause. Before presenting Query 4.2 which contains multiple association pattern expressions, we give the following rule that is relevant to the query.

Rule #4 If the association operator is used between two classes that are connected by more than one association (in this case the association links have to be distinctively named in the schema), then the name of the intended association needs to be specified after the association operator.

For example, there are two associations between the classes Undergrad and Department (Figure 2.1): the association labeled Minor and the inherited association labeled Major. Hence, the expression "Undergrad *Major Department" is used to refer to undergraduate students and their major departments.

Query 4.2 Display the names of all the undergraduate students minoring in the major department of the undergraduate student whose SS# = xxx. Display also the name of the department.

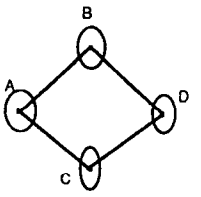
```
context  Undergrad_1 *Minor Department_1,
         Undergrad_2 [SS# = xxx] *Major Department_2
   where Department_1 = Department_2
   select Undergrad_1 [name], Department_1 [name]
display
```

The two association pattern expressions of the Context clause in the above query create two subdatabases. The two subdatabases are then linked by the condition stated in the Where subclause to produce a new subdatabase to which the Select subclause is applied. Also, in this query, the comparison in the Where subclause is performed between two type-comparable objects (i.e., objects that belong to the same class or to two different classes of the same generalization hierarchy).

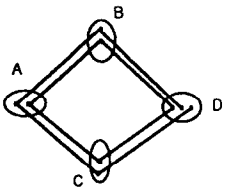
5. Assignment Operator and Closure Property

The assignment operator ":=" can be used to save the subdatabase that is returned by a query. For example, the expression "X := context A * B * C" creates the subdatabase X whose intensional pattern consists of the classes A, B, and C, their associations, and their descriptive attributes and whose extensional patterns are the ones that are of the type <A,B,C>. To save a subdatabase permanently (i.e, not only for the duration of the query session), the key word save is used before the subdatabase name (e.g., save X := Context A * B * C).

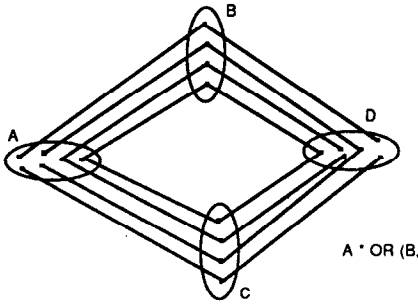
A class that appears in a subdatabase can be referenced in the context of that subdatabase (i.e., only the instances of that class that appear in the subdatabase are considered) by qualifying its name with the subdatabase name using a colon (e.g., X:C). If not qualified by a subdatabase name the "base" class is



A * AND (B, C) AND * D



A * AND (B, C) OR* D



A * OR (B, C) OR* D

Figure 4.2: Extentional Pattern Types Specified by Context Expressions that Form Networks

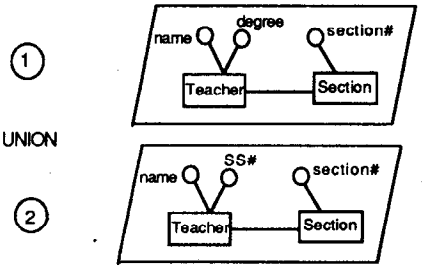


Figure 4.3a Two Union-compatible Subdatabases

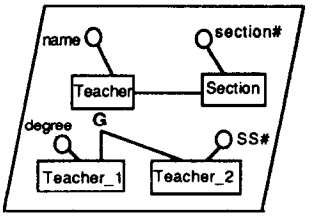


Figure 4.3b: The Result of applying the UNION Operation to the two subdatabases represented in Figure 4.3a

Figure 4.3: An example of applying a set operator to two union compatible subdatabases

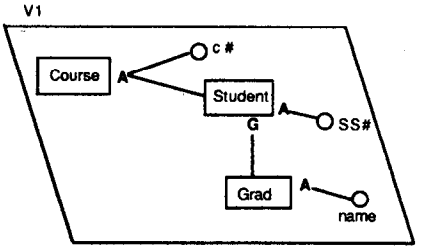


Figure 5.1: The Intensional Pattern of the View V1 Defined by an OQL Query

assumed. For example, if the class C in the original database is associated with the class D and D is associated with E, then the following query creates the subdatabase Y that consists of the class C derived from the subdatabase X and the class E derived from the original database together with all of their descriptive attributes (refer to Rule #1 in Section 3.2).

```
save Y := context X:C * D * E
          select X:C, E
```

This query operates on two databases, the subdatabase X and the original database and produces a single subdatabase (Y) whose intensional pattern consists of the classes C and E and a derived association between them. The subdatabase Y can be used by another query to produce yet another subdatabase.

Another consequence of maintaining the closure property in OQL is that nested association pattern expressions can be used. For example, the following two expressions are equivalent (where X is as defined before and assuming that B is directly associated with each of the classes A, C, J, and K).

```
Z := context J * X:B * K
Z := context J * (A * B * C):B * K
```

In the second expression, "A * B * C" is a nested association pattern expression that identifies a certain subdatabase whose B component is referenced by the outer association pattern expression.

Query 5.1 Define the user's view V1 that consists of the classes Course, Student and Grad such that a Course instance must have been taken by the Student instance who may or may not be a graduate student provided also that the Course instance belongs to the EE department. The attributes C#, SS#, and Name must appear with the classes Course, Student and Grad, respectively.

```
save V1 := context [ Department [name = 'EE'] * Course *
                     Transcript * Student ] * Grad
           select c#, Student [SS#], Grad [name]
```

The intensional pattern of the subdatabase V1 is shown in Figure 5.1, where a generalization association connects Student to Grad. In this subdatabase, only graduate students shall have values for the attribute Name, even though, in the original database, all students may have Name values.

6. Conclusion

In this paper, we have introduced the query language OQL for manipulating OO databases. A query in OQL returns a subdatabase, which has structural properties that are similar to those of the original database (i.e., it contains multiple classes and associations). In other words, the closure property is preserved. Thus, the result of a query can be an operand of another query. The distinguishing features of OQL are summarized as follows.

1. A subdatabase returned by a query represents a "context" under which some operations can be specified and executed. In an OQL query, the specification of the Context subdatabase is separated from the specification of the operations on that subdatabase. This allows different operations to be performed on different object classes in the specified context.
2. Set operations can be performed on Union-compatible subdatabases. The result of a set operation is a new subdatabase that can be further manipulated in the normal way.
3. The association operators and the AND/OR operators allow very complex association patterns to be specified in a simple way. The same functionality would be specified in SQL, for example, by a complex nesting of Select-From-Where blocks.
4. Comparison operators (i.e., '=' and '!=') can be used to compare objects that belong to E-classes directly without referencing their attributes.
5. OQL makes full use of the inheritance property of the generalization association. A class inherits all the associations that emanate from or connect to its superclasses.

OQL is particularly suited for implementation on a graphics system. A query can be specified by browsing the S-diagram of object classes and pointing and traversing object classes to enter qualification conditions and association operators. An implementation of OQL on a SUN workstation is reported in a master's thesis [TY88].

Acknowledgement

This research is supported by the Navy Manufacturing Technology Program through the National Institute of Standards and Technology (formerly the National Bureau of Standards) grant number 60NANB4D0017, the Florida High Technology and Industry Council grant number UPN 85100316, and the National Science Foundation grant number DMC-8814989.

REFERENCES

BAN88 Jay Banerjee, Won Kim and Kyung-Chang Kim, "Queries in Object-Oriented Databases," Proceedings of the Fourth International Conference on Data Engineering, California, 1988, pages 31-38.

BAT85 D. Batory and W. Kim, "Modeling Concepts for VLSI CAD objects," ACM Transactions on Database Systems, vol. 10, No. 3, September 1985, pages 322-346.

COD79 E. Codd, "Extending the Database Relational Model to Capture More Meaning," ACM TODS, Vol. 4, No. 4, December 1979, Pages 397-434.

DIT86 K.R. Dittrich, "Object-oriented Database Systems: the Notion and Issues," Proceedings of the International Workshop on Object-Oriented Database Systems, California, September 1986, Pages 2-4.

HAM81 M. Hammer and D. McLeod, "Database Description with SDM: A Semantic Association Model," ACM TODS, vol. 6, No. 3, September 1981, pages 351-386.

HUL87 R. Hull and R. King, "Semantic Database Modeling: Survey, Aplications, and Research Issues," ACM Computing Surveys, pages 201-260, Vol.19, No.3, September 1987.

KIN84 Roger King, "A Database Management System Based on an Object-Oriented Model," Proceedings of FIWEDS, South Carolina, October 1984.

MAC85 R. MacGregor, "ARIEL—A Semantic Front-End to Relational DBMSs," Proceedings of VLDB 85, pages 305-315.

SHI81 D. Shipman, "The Functional Data Model and the Data Language DAPLEX," ACM TODS, Vol. 6, No. 1, March 1981, pages 140-173.

SU86 S. Su., "Modeling Integrated Manufacturing Data with SAM*," IEEE Comp., January 1986, pages 34-49.

SU88 S. Su, V. Krishnamurthy, and H. Lam, "An Object-oriented Semantic Association Model (OSAM*)," appearing in: A.I. in Industrial Engineering and Manufacturing: Theoretical Issues and Applications, S. Kumara, A. L. Soyster, and R. L. Kashyap (eds.), American Institute of Industrial Engineering, 1988.

TY88 Frederick Ty, "G-OQL: Graphics Interface to the Object-Oriented Query Language OQL," Master thesis, University of Florida, 1988.

XIA89 Daozhong Xia, "The Implementation of OSAM* KBMS Prototype in Vbase," an Internal Report, Database Systems Research and Development Center, University of Florida, 1989.

ZAN83 C. Zaniolo, "The Database Language GEM," Proceedings of the ACM SIGMOD Intl. Con. on the Management of Data, 1983.