The Composite OLAP-Object Data Model: Removing an Unnecessary Barrier

Elaheh Pourabbas *
IASI "Antonio Ruberti"
National Research Council
Viale Manzoni, 30 I-00185 Roma, Italy
pourabbas@iasi.cnr.it

Arie Shoshani †
Lawrence Berkeley National Laboratory
Mailstop 50B-3238
1 Cyclotron Road Berkeley, CA 94720 USA
shoshani@lbl.gov

Abstract

OLAP and object data models represent different logical concepts and structures, and therefore separate database systems with different query languages were developed based on these models. We show in this paper that it is desirable and possible to combine these models to represent realistic modeling requirements. We define in this paper an OLAP-Object data model that combines *the main characteristics of OLAP and Object data models in order to represent their functionalities in a common framework. We use three different types of object classes:* primitive, regular *and* composite. *In the OLAP-Object data model, primitive and regular classes which represent object structures can be used for form composite classes that represent OLAP structures. We define a query language that uses path structures to facilitate data navigation and data manipulation. The proposed language uses the concept of an* anchor. *An anchor is an object class (primitive, regular or composite) that is selected as a starting node from which paths structures can be formed to express queries. The power of the proposed query language is illustrated through numerous examples. The syntax and semantics of the proposed language are developed.*

1. Introduction

The OLAP data model [9] was introduced in order to manage multidimensional summary databases. The conventional object data model represents mainly object, attributes, and associations between objects. These two data models have been used to build different specialized database systems: OLAP-based systems for the multidimensional data and mostly relational systems for Object-based data. The design of the object-based data uses one of the familiar object-attribute-association models (such as the Entity-Relationship model, the Unified Modeling Language (UML) or similar Object models).

*This work was supported by a Fulbright Scholarship academic year 2004-2005 while the author was visiting Lawrence Berkeley National Laboratory, U.S.A.

†This work was supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

These systems present to user with different query languages and user interfaces. We argue in this paper that this separation of concepts is unnecessary and limiting. It is desirable and possible to combine the logical structures of these models and devise a single coherent query language for the combined model.

The OLAP data model, which is also referred to as the Statistical Data Model [19] consists of three basic constructs:

1. Dimensions, where each can consist of a multi-level classification hierarchy;

2. A multidimensional object, also referred to as a "cross-product" object;

3. Summary attributes, where each is associated with the multidimensional object.

For example, a database for "average-income by city, race, and sex", can be modeled in an OLAP data model, where the dimensions are *City*, *Race*, and *Sex*, the multidimensional object is the cross-product $(City) \times (Race) \times (Sex)$, and the summary attribute is the "average-income". If, in addition, cities were organized by counties, and counties organized into states, a "classification hierarchy" (also referred to as "category hierarchy") $City \rightarrow County \rightarrow State$ can be associated with the *City* dimension. *City*, *Region*, and *Country* are referred to as category attributes. Summarization over the classification hierarchy can also be specified, such as "average-income" for counties and states.

Various representations of these concepts were proposed, including the graphical representation of Statistical Models [7], Multidimensional OLAP, or MOLAP [1] [10], Relational OLAP, or ROLAP [2] [3], and Data Cubes [4] [12] [14]. However, in many applications the elements of the dimensions as well as the elements of the classification hierarchies can be objects in their own right, such as cities and states in the example above. In such cases, these objects can have their own attributes (e.g. the mayor of a city, or the governor of a state). Furthermore, such objects may be associated with other objects (such as the hospitals in a city). This obviates the need to have concepts from the OLAP modeling domain to be combined with an object-attribute-association model.

1.1. Contribution

In this paper, we define an OLAP-Object data model that permits a uniform definition of concepts, and achieves the functionalities from both domains. Specifically,

- We classify three different object classes: *primitive*, *regular* and *composite*. We investigate well-formed composite objects and we study their summarization semantics.
- In addition to the conventional associations between object classes, we define a *summarizable* association that facilitates the specification of classification hierarchy structures.
- We define a query language which uses the path concept in order to facilitate data navigation and data manipulation. An important concept of the proposed language is an *anchor*. It allows us to fix dynamically an object class (primitive, regular or composite) as the origin of paths over the OLAP-Object data model for expressing queries.
- We investigate the power of the proposed query language through multiple query examples.

1.2. Related work

The desire to bridge the gap between OLAP and object data models has recently motivated several studies. One approach taken by some authors was to extend object database systems to support summary querying. For instance, in [12], the CUBE operator, which is an n-dimensional generalization of GROUP BY in SQL is introduced. This operator computes the aggregations over all subspaces defined by different combinations of category attributes. In [8], an extension of the relational algebra and the SQL syntax is proposed in order to overcome the imitation of SQL to answer queries involving repeated grouping and aggregation over the same groups. In [15], the authors proposed a model for data warehouses with star/snowflake schema which extends the relational data model of SQL to express queries involving dimension hierarchies. Another approach is to develop a new query language for OLAP (e.g., Multi-Dimensional eXpressions- MDX) and translate SQL to that new language and vice versa.

The approaches mentioned above are based on extending an existing language. In our work, we address this issue by considering constructs that support both object and OLAP data structures in a single common framework, in order to answer queries that are formulated jointly over both domains. We propose a composite OLAP-Object data model in order to achieve the desired functionality and semantics of both domains, and define a query language over this data model.

Combined queries over OLAP and object database systems were addressed previously, in [17] and [13] in the context of federating databases. The connection between these databases is achieved through links or functions that map categories in an OLAP database to the corresponding entities in a separate object database. A summary query language, called SumQL for querying an OLAP data is defined. The language was based on the SQL language that was augmented with constructs such as measure, dimensions with hierarchies, and automatic aggregation (automatic application of a pre-specified aggregation function, such as SUM). In contrast, the composite data model presented in this paper, can support multiple OLAP databases and uses a common data model framework to represent constructs in both domains (primitive, regular and composite objects). Consequently, our proposed query language can be applied uniformly to both OLAP and Object data constructs.

A similar approach, but from system point of view that was introduced in order to remove the gap between OLAP and current database management systems (DBMSs) is presented in [11]. A list of shortcomings of the current DBMS for OLAP systems is presented, which includes the inadequacy of the relational model, the current client/server architecture and the problem of the correct implementation of summarizability [16] with the GROUP BY statement. The solutions to bridge this gap are defined on top of a commercial OLAP engine and the requirements for an analytic query language to close the gap are defined. The foundation of such a query language is an extension of XQuery with explicit grouping constructs to support aggregation queries as discussed in [5]. Our data model is defined from a logical modeling point of view using well-formed composite objects and the semantic of summarization over composite objects. The foundation of the proposed query language is based on three basic data model constructs, the concept of an anchor, and path expressions originating from the anchor.

The paper is structured as follows. The next section describes the basic constructs of the OLAP-Object data model, and introduces a graphical representation of the model. Section 3 discusses well-formed composite object classes, and defines the summarization semantics. Section 4 discusses the application of paths to the OLAP-Object data model. Section 5 proposes a query language, and investigates the relative power through multiple query examples. Section 6 compares the proposed query language with ODMG's Object Query Language, OQL [6]. Section 7 concludes.

2. Constructs and Graphical Representation

In this section, we first give the basic constructs of the composite OLAP-object data model and then, we present its graphical representation.

2.1. Constructs

We start with the well-known basic concepts of an object-attribute-association model, and then introduce additional concepts for supporting the semantics of OLAP data. The basic concepts include:

- *Object Class* - represents a set of individual objects, each having a globally unique identifier. An object

class has a label (name). An object class must have at least one attribute. Such a class is called a *regular* object class.

- *Attribute* - a property associated with an object class. Each individual object in that class determines the attribute value associated with it. Therefore, there is a functional dependency of the attribute on the object class. An attribute has a label (name).

- *Association* - a way of pairing objects from two object classes. An association has a pair of cardinalities: one-to-one, one-to-many, many-to-many. An association has a label (name).

For instance, let us consider two regular object classes, called Student, and Course. The attributes of the first class are *Name* and *Age*, while the attributes of Course are *Name* and *Start-date*. An association between these classes is *enrolls*.
In addition to the above concepts, we need to have several concepts that will allow OLAP structures to be represented in the data model. The first is a primitive object class (such as "race" or "sex") that can be used for defining dimensions in the OLAP model. The second is a composite object class that supports the concept of a cross-product or data cube. We also need the concept of an association with special properties, called a summarizable association which is used for defining classification hierarchies. We define these next:

- *Primitive object class* - an object class that represents a finite enumerated set of values. The values represent the identifiers of the individual objects in the class. A primitive object class has a label (name).

- *Composite object class* - an object class defined over two or more component object classes, where each individual composite object has a new identity which is independent of the identities of the component objects. The identifier of the composite object is composed of the identifiers of the component object classes. A composite object class must have at least one attribute. A composite object class has a label (name).

- *Summarizable association* - we make a distinction between a simple association and a summarizable association. A summarizable association must be one-to-many and must pass the test of completeness and non-overlap of objects as described in [16]. Example of summarizable associations are *is-in* between City and State, or *offer* between Course and Department. We expand on the summarization semantics in Section 3.

- *Classification Hierarchy* - an ordered list of object classes of a similar type organized by levels that are related by summarizable associations. The mapping from a lower (more detailed) level of objects to a higher (more aggregate) level is specified by a classification function. This introduces a hierarchical relationship between pairs of object classes and the structure is referred to as classification hierarchy. The operation that implements the summarization from a lower (higher) level to a higher (lower) is called roll-up (drill-down).

For instance, let us consider "average-income by city, race, and sex" mentioned in the previous section. This can be modeled as a composite object class named *Income* where *City*, *Race*, and *Sex* are component object classes. *City* is a regular object class and its attributes are *Name* and *Mayor*, while *Race*, and *Sex* are primitive object classes. *City* is a category of the classification hierarchy $City \rightarrow State \rightarrow Counry$. The attribute of the composite object class is *average-income*.

2.2. Graphical Representation

The representation of object classes and relationships are based on nodes and arcs. Specifically, primitive and regular object classes are represented by a simple node (◯), whereas a composite object class is represented by a "circled-X" (⊗) node.

In the graphical representation, each component class is connected to the composite class by a dashed line. Each node contains a label that indicates the name of the class it represents, and the attributes are shown in *italic* text next to the node. See example in Figure 1.

To represent a *simple relationship* (or a simple association) between two object classes, a simple labeled directed arc is used. The directed label indicates the name of relationship and is represented in *italic* text with lowercase letters. The directionality of the label is chosen according to the meaning of the label. For example, a project can "have" people, or people can "work-in" a project, which are represented with opposite label directionality. Note that, the model allows for inverse labels, but we do not show these in the graphical representation. A *classification relationship* (or a summarizable association) between two object classes is illustrated by a directed bold arc.

Example 1 In Figures 1 and 2 graphical representations for an Object schema and an OLAP schema are illustrated, respectively. In our composite OLAP-Object model it is possible to combine the two types of schemas in a straightforward manner since they are based on the same underlying constructs. Examples of combined schemas are provided in the next sections.

3. Characteristics of Composite Object Classes

3.1. Well-Formed Composite Object Classes

Generally, composite object classes are based on the idea of representing many-to-many relationships between pairs of component object classes as classes in their own right. For instance, let us consider the relationship *enrolls* between Student and Course, which associates students and the courses they attend (see Figure 3-(A)). If *enrolls* is not defined by any attributes then in the database schema design

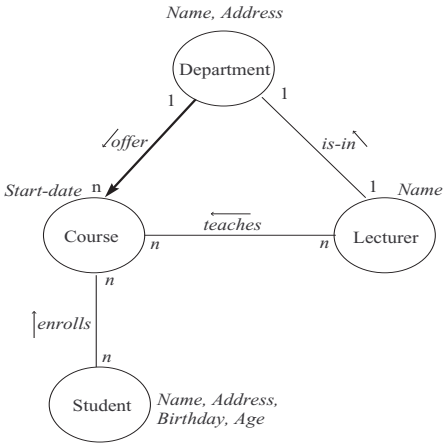

Figure 1. Example of a database with regular object classes

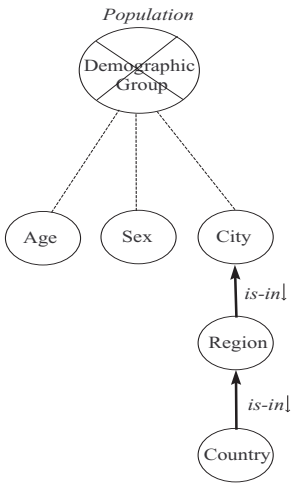

Figure 2. Example of a composite object class

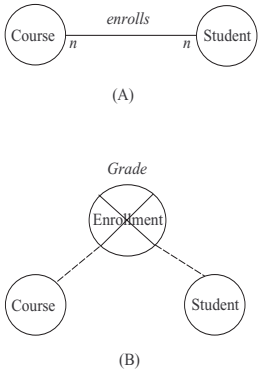

Figure 3. Example of a binary many-to-many relationship (A) and a composite object class (B)

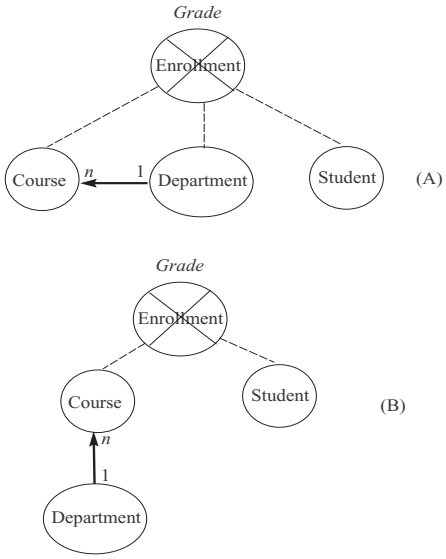

Figure 4. Example of a non well-formed (A) and a well-formed composite object class (B)

it can be represented by a binary many-to-many relationship. However if the relationship has an attribute associated with it, then a composite object is used. For instance, if *Grade* is associated with *enrolls*, then we need to use a composite object class, named *Enrollment* and define it over the original object classes, i.e. Student, and Course (see Figure 3-(B)). Thus, a composite object class without any attribute is equivalent to a simple many-to-many association between the component object classes.

Similarly, ternary and higher degree relationships are captured by well-formed composite object classes as follows.

Definition 3.1 *A composite object class is well-formed if and only if between any pair of its component classes there is no functional dependency.*

Explanation: Let us consider a composite object class defined by three object classes named X, Y, and Z. Suppose that a functional dependency exists between at least one pair of object classes, say X and Y. Thus for each instance of X there are n instances of Y. This dependency represents the binary one-to-many relationship between X and Y, which defines a partial order on instances X and Y. Therefore, these two component classes are related by a hierarchical relationship. This contradicts the definition of a composite object that has an identity which is independent of the identities of the component objects.

For instance, the schema of the composite object shown in Figure 4-(A) is not well-formed. As stated in Definition 3.1, there is a functional dependency between Course and Department. The well-formed composite object is shown in Figure 4-(B).

3.2. Summarization Semantics

Composite objects must satisfy the conditions of *summarizability* discussed in [16]. These conditions are: a) disjointness of objects in the dimension hierarchies. This implies that instances of category attributes in dimensions form disjoint subsets of the elements of each level; b) completeness is a condition that means that all the categories of each dimension exist, and every category is assigned to some category in the level above it in the hierarchy; c) correct use of measures is a conditions that states that each summary attribute must be associated with a aggregation function (COUNT, SUM, MIN, MAX, and AVERAGE), and only that function can be applied to it.

Definition 3.2 *Performing summarization over summary attributes associated with composite objects have certain semantic behavior. Therefore, queries expressed over composite object classes satisfy the following semantic conditions:*

1. *if a dimension (or component object class) is not specified, then the summarization must take place over all values of its individual objects.*
2. *if a set of values of a certain dimension is specified, then the restriction must take place over this set only and the remaining object values are not included. In the result of query, the composite object contains only this set of objects for this dimension.*
3. *if a single or a range of values of a certain dimension is specified, then the summary attribute of the composite object must be evaluated over this single/range of values.*
4. *if no dimension is specified, then summarization must take place over all values of the dimensions. This corresponds to obtaining only a single value for the summary attribute (grand-total).*
5. *if a dimension is specified at a higher (lower) level of the classification hierarchy with respect to the level of the object class, which is a component of the composite class , then the summary attribute must be aggregated (dis-aggregated) to the desired level through roll-up (drill-down) operation.*

Example 2 Consider Figure 2, and the query "Find city and population in the $20 \div 40$ age-groups". Sex is not specified, then according to the Definition 3.2-*(1)* it is summarized. Since a sequence of values (i.e. $20 \div 40$) for Age is specified, condition *(2)* of the definition above is applied. If instead, the query also specified a condition for Sex, e.g. getting the population for *females*, then according to Definition 3.2-*(3)*, the value *male* of Sex is not included and the result contains only *Population* of *female* by city and age in the $20 \div 40$ range. If the query specifies "population of the $20 \div 40$ age-groups", then according to Definition 3.2-*(3)*, *Population* should be aggregated over values of Age from 20 to 40. If only total population of demographic group is specified, then Age, Sex, and City are summarized to obtain a single total population value (see Definition 3.2-*(4)*). Finally, if the query specifies "population by region", then according to Definition 3.2-*(5)*, *Population* should be aggregated from City to Region level by roll-up operation.

4. Paths over the OLAP-Object Data Model

A path expression is defined from a starting object class toward the object class to be accessed in a query without having to express explicit join conditions. It is possible to associate with the *same* path, one or more predicate conditions. For instance, let us consider Figure 1, and the query "students who are enrolled in the Mathematics course offered by the Electrical Engineering department". If we insist on writing this query as a single path expression, it will require an expression as follows:

Student.Course:Name="Mathematics".Department:Name= "Electrical Engineering"

We find the above notation cumbersome, and not easily extendible to queries with additional predicate conditions that can cause forks in the path expressions. Instead, in our query language we support multiple *simple* path expressions, where each path expression consists of only a single predicate. It is expressed by a sequence of object classes, which are separated by dot notation. The sequence ends up with a colon, after which a predicate is expressed in order to restrict the objects selected. Thus, a query with multiple predicate conditions is split up into a conjunction of several simple paths. Accordingly, the previous query is expressed as follows:

Student.Course:Name="Mathematics" **AND** Student.Course.Department:Name= "Electrical Engineering"

We allow multiple values of an attribute to be specified in a single path. For instance, if the query is "students who are enrolled in courses Mathematics and Logic", the path is defined as follows:

Student.Course:(Name="Mathematics" **AND** Name="Logic")

A path can be usually expressed as a simple dot notation. However, in the case that more than one relationship hold between two objects, then only one relationship should be specified. For instance, suppose that the object Student is related to Course by an additional relationship, named audits. To specify students who are enrolled in the mathematics course (rather than audits the class), the relationship enrolls should be specified between Student and Course in the path expression. We use the notation of specifying the selected relationship in parenthesis as shown below:

Student(enrolls)Course:Name="Mathematics"

If a path goes through a composite object class that does not involve any attribute of this class, then it has the same structure as having only regular classes. For instance, suppose that in Figure 4-(B) an object class named City is related to Student by a *lives* relationship. The query: "Find Cities of students who are enrolled in a Computer Science course" is expressed as shown below.

City.Student.Enrollment.Course:Name="Computer Science"

Note that this query does not specify the *Grade* attribute. Queries that specify attributes of composite objects have a different structure, which will be discussed in the next section, after the concept of an *anchor* is introduced.

5. The Query Language

An important feature of the proposed query language is an *anchor*. It allows us to select an object class (primitive, regular or composite) as the query focus. We call this class the *anchor-class*. The path condition expressions can be formulated starting from the anchor-object class using paths over the OLAP-Object data model described in Section 4. In expressing the desired output, the results of a query also refer to the anchor-class and are obtained based on the evaluation of the condition expressions. The general form of a query over OLAP-Object data model is given below.

OLAP-Object query::=
ANCHOR <anchor-class>
FROM <anchor-class-composite>
CONDITION <conditional-expressions>
OUTPUT <anchor-path-items>

The ANCHOR clause contains either one composite object class or one or more primitive and regular classes. In the latter case, the (primitive or regular) classes belong to a composite object class, which should be specified in the FROM clause. In this case, the ANCHOR clause specifies the dimensions of the composite object for which the summary attributes are to be computed according to semantics introduced in Section 3.2. The CONDITION clause contains simple and path conditional expressions, according to the structure discussed in Section 4. Finally, the OUTPUT clause specifies the desired results based on the evaluation of the conditional expressions or the application of aggregate functions on summary attributes. We use the notation, "*" and "ID" to represent all attributes and instances of a given class, respectively.

The formal syntax and semantics are given in [18]. A summary of the main clauses is given below.

<query-item> ::= <anchor-statement><from-statement>
<conditions-statement> <output-statement>

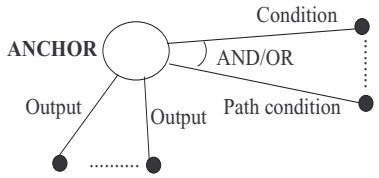

Figure 5. Illustration of Object query

<anchor-statement> ::= **ANCHOR** <simple-class>
{',' <simple-class>} | <composite class>
<simple-class>::= <primitive class> | <regular class>
| <regular class> ISA <regular class>

<from-statement>::=**FROM** <composite class>

<conditions-statement>::=**CONDITION**
<conditions> | <path-conditions> | <null>

<output-statement>::=**OUTPUT** <anchor-paths>
{';' <anchor-paths>}

In the next subsections, we first illustrate how to express queries that are purely object-based and queries that are OLAP-based. We then show how we can express combined OLAP-Object queries without splitting queries into multiple steps. In the last subsection, we discuss composite-composite queries. They are composite queries, in which the evaluation of the conditional expressions depends on a second composite query. We illustrate the power of the proposed query language through multiple query examples. Note that the queries expressed over primitive object classes (e.g., Sex) refer to their domain values (e.g., Female, Male).

5.1. Object Queries

In this case, the anchor is a regular class and the conditional expressions are formulated either on anchor or on regular classes at the end of paths (see Figure 5). In the output clause the result can be the anchor class itself, its attributes, or attributes of paths emanating from the anchor. In the following, some examples of queries formulated on regular classes shown in Figure 1 are given.

Query 1: Find students who are enrolled in Mathematic and Logic courses given by lecturers of the Electrical Engineering department.

ANCHOR	Student
CONDITION	Student.Course: (Name= "Mathematics"
	AND Name="Logic")
	AND Student.Course.Lecturer.Department: Name
	= "Electrical Engineering"
OUTPUT	Student:*

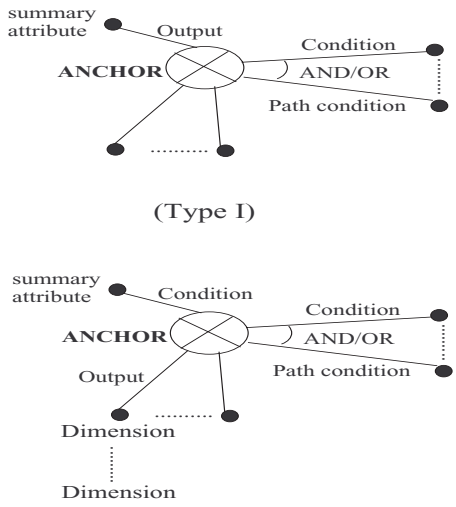

Figure 6. Illustration of Composite query

Explanation: Student is the anchor class. Since no specific property is requested from the query, all attributes of Student are provided.

If in the query above, instead of "students", we wish to get "number of students" then in the OUTPUT clause Student:COUNT is specified. Similarly, Student:AVG(Age) can be specified if "average age of students" is desired.

Query 2: Get name and birthday of students and addresses of departments for students that take courses in at least one department.

ANCHOR	Student
CONDITION	Student.Course.Department: NOT NULL
OUTPUT	Student:(Name, Birthday);
	Student.Course.Department: Address

Explanation: The quantifier *some* is expressed by NOT NULL in the conditional expression. The output paths are constructed starting from the anchor class Student.

5.2. Composite Queries

We classify two types of composite queries depending on whether the conditions are expressed on the summary attribute or not. They are labeled as Type I and Type II in Figure 6. Queries of Type I have condition expressions that are defined on dimensions, and specify the summary attribute in the output. The opposite occurs for queries of Type II.

The following query examples are formulated on the schema of the OLAP database shown in Figure 2.

Query 3: Find population by region and age for demographic groups of females in $20 \div 40$ age-groups, and restrict the result to regions in the USA only.

ANCHOR	Region, Age
FROM	Demographic_Group
CONDITION	Age **EQ** $\{20,40\}$
	AND Region.Country:Name="USA"
	AND Sex="Female"
OUTPUT	[Region, Age](Demographic_Group): Population

Explanation: This is a Type I query. Region and Age are anchor classes. They are two dimensions of Demographic_Group, which is specified in the FROM clause. In this query, the tuples are restricted to a range of values on dimension Age, and the dimension Sex is restricted to "Female". The summary attribute Population is aggregated from City to Region by the roll-up operator. The output is only for regions in the USA.

Query 4: Find regions in the USA with a female population $> 10M$ for females between $20 \div 40$.

ANCHOR	Region
FROM	Demographic_Group
CONDITION	Age **BETWEEN** $\{20,40\}$
	AND Region.Country:Name="USA"
	AND [Region](Demographic_Group):Population $> 10M$
OUTPUT	Region:Name

Explanation: The values of Age is restricted to a range, and specified by the BETWEEN operator in the CONDITION clause. This differs from the previous query because the summary attribute should be aggregated over this range. If the query above would request instead the total population over all the regions selected, then in the OUTPUT clause [Region](Demographic_Group): SUM(Population) has to be specified.

An example ofType II composite query is illustrated next.

Query 5: Find regions where the female population is $>$ 100000 and output the population per city in these regions.

ANCHOR	Region
FROM	Demographic_Group
CONDITION	Sex="Female"
	AND [Region](Demographic_Group):Population > 10000
OUTPUT	Region.City: Population

5.3. Composite-Object Queries

We illustrate in this section queries that involve both object and OLAP Constructs. We refer to the schema shown in Figure 7.

Query 6: Find the name of governors of states where the average income per state is > 20000.

ANCHOR	State
FROM	Demographic_Group
CONDITION	[State](Demographic_Group):Average-Income> 20000
OUTPUT	State.Governor:Name

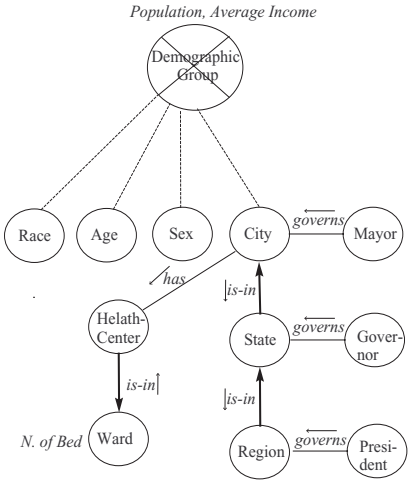

Figure 7. Example of composite-object class

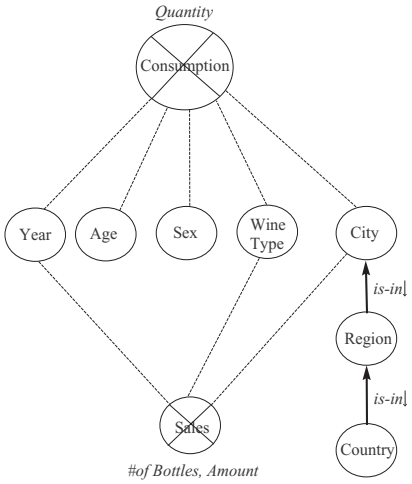

Figure 8. Example of composite-composite classes

Explanation: This query specifies as output an object class from the Object construct. The conditional expression is defined over the Demographic_Group from the OLAP construct, which implies the summary attribute Average-Income should be rolled up from City to Region.

Query 7: Find the N.of Beds in maternity wards of cities where the female population between $20 \div 40$ years old is > 1500.

ANCHOR	City
FROM	Demographic_Group
CONDITION	Age **BETWEEN** $\{20\ 40\}$
	AND Sex="Female"
	AND [City](Demographic_Group): Population > 1500
	AND City.Health-Center.Ward:Name="Maternity"
OUTPUT	City.Health-Center.Ward:N.ofBed

Explanation: This is similar to the previous query but the conditional expressions are defined on both constructs.

5.4. Composite-Composite Queries

These queries are subdivided into query-subquery. The subquery is resolved first and the result is used to give the final answer. The query examples that are given below refer to Figure 8.

Query 8: Get the consumption of wine in cities by sex where sales exceed $10000 in 2001.

Query1	**ANCHOR**	City
	FROM	Sales
	CONDITION	Year="2001"
		AND [City](Sales):Amount> 10000
	OUTPUT	[City](Sales):Amount

ANCHOR	City, Sex
FROM	Consumption
CONDITION	Year="2001" **AND** City **IN** Query1
OUTPUT	[City,Sex](Consumption):Quantity

Explanation: Query 1 is formulated on the composite object Sales. Then, the consumption of wine is calculated for each city retrieved from Query 1.

Query 9: Find sales of red wine in USA regions, where the wine consumption of males in 2003 is ≥ 300 gallon.

Query1	**ANCHOR**	Region
	FROM	Consumption
	CONDITION	Year="2003" **AND** Region.Country="USA"
		AND Sex="Male"
		AND [Region](Consumption):Quantity ≥ 300
	OUTPUT	[Region](Consumption):Quantity

ANCHOR	Region
FROM	Sales
CONDITION	Region **IN** Query1
	AND Year="2003"
	AND Wine_Type="Red"
OUTPUT	[Region](Sales):(#ofBottles, Amount)

The explanation of this query is straightforward.

5.5. Back reference and multiple references

Let us consider queries which express condition caused by ellipsis in natural language (e.g., their, its, etc.). This

class of queries can be answered by a specialized version of paths discussed in Section 4, called *back reference* path. Let us consider Figure 9, and ask the following query.

Query 10: Give the name of books for which the authors live in the same city as a borrower.

To answer this query, city of author should be matched with city of borrower. For a correct formulation of the query, we should retrieve books from the first expression such that on *these books* the second expression can be evaluated. A solution to this is to create a reference for Book as follows:

ANCHOR	Book
FROM	Loan
CONDITION	Book(x).Author.City:Name=
	Name(City.Borrower.Book(x))
OUTPUT	Book:Name

Note that the relationship between Borrower and Book is Loan. Back reference is useful to model *self-reference*, where a class is related to itself by an association. For instance, a class called Person could be related to itself by the relationship Friend. The query

Query 11: Find people who have at least one friend with the same first name, and return full name of person and his friend.

is easily formulated as follows:

ANCHOR	Person
CONDITION	Person(x):First-Name=
	First-Name(Person(y)(friend)Person(x))
OUTPUT	Person(x):(First-Name,Last-Name);
	Person(y):(First-Name,Last-Name)

The query language can also support *isa* relationship construct, since the *isa* relationship is not considered a "distinct" association between regular object classes. In other words, if a class O_i is a specialization of class O_j and is invoked by a query, then the query is evaluated on O_j.

For instance, let us consider $Graduate\_Student$ to be a subclass of $Student$ in Figure 1. Suppose that a query requests the name of graduate students enrolled in a Mathematics course. The effective anchor-class is $Student$, and the query is expressed as follows:

ANCHOR	Graduate_student ISA Student
CONDITION	Student.Course: Name="Mathematics"
OUTPUT	Student:Name

6. Comparison with OQL

OQL is an object-based language version of SQL designed for the Object Data Management Group (ODMG) object model. It uses SQL-like syntax (SELECT, FROM, WHERE) [6]. In order to access data, all queries start from a named object class and navigate through dot notation "."

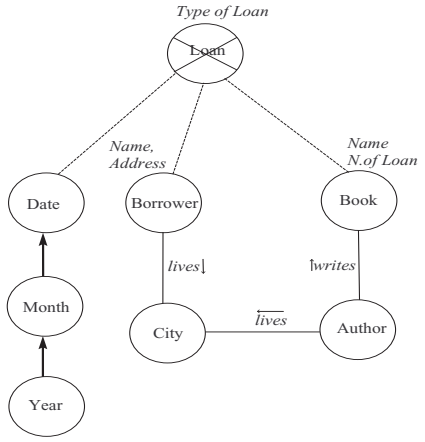

Figure 9. Example of composite-object class

to reach other objects through relationship paths. Path expressions are used to access any attribute of an object, and dot notation is applied only to a single object and not to a collection of objects. When a collection of objects is referenced in OQL queries, iterator variables in the FROM clause are used. For instance, note the variable "s" in the FROM clause of the query "Find the titles of courses taken by Smith" as shown below:

SELECT	s.Courses.Title
FROM	s **in** Student
WHERE	s.name="Smith"

In contrast, our query structure refers to the collection of objects in the object class and the association between object classes is achieved through the dot notation paths.

The above query in our query language is expressed as follows:

ANCHOR	Student
CONDITION	Student:Name="Smith"
OUTPUT	Student.Courses:Title

In the ANCHOR clause of the proposed query language, similar to the FROM clause of OQL, for every object class an *extent* (not the class name) is declared, which is used to refer to the current collection of all objects of that class. Unlike OQL, in our proposed query language, all queries start from an object class declared in the ANCHOR clause, and consequently, path expressions refer to a collection of objects.

Aggregation operations in OQL, similar to SQL, are provided by GROUP BY clause. This provides explicit reference to the collection of objects within each group or partition. The HAVING clause is used to filter the partitioned sets (that is select only some of the groups based on group

conditions). In contrast, in our query language we take advantage of the semantics of a composite object and classification hierarchies to infer aggregation. Specifically, aggregation is inferred over dimensions *not* included in the query. For example, given a composite object representing population by state, race and age, the query *get population by state* infers summarization over all values of race and sex. Similarly, if states are organized by region as a classification hierarchy, then the query *get population by region* will imply aggregation over states because the classification hierarchy semantics are part of our data model.

Consider query 5 as an illustration of the power of our query language to express OLAP queries. This query is expressed in OQL as follows:

```
SELECT  STRUCT (rname, r.City.Population)
FROM  r in
  { SELECT STRUCT (region.Name, pop:SUM(c.Population))
    FROM c in CITY}
    GROUP BY c.Name}
WHERE  region.DemographicGroup.Sex="Female"
         AND  region.Population >10000
```

In our query language, the formulation of this query (see Query 5) is simple and intuitively clear because our query language is designed to use the summarization semantics discussed in Section 3.2. In particular, the roll-up operator is applied automatically to the dimension City of composite class DemographicGroup in order to obtain Population by the higher level Region.

The above query in OQL invokes only one dimension. Complex queries which involve aggregation of multiple dimensions are tedious and require the query to be expressed in multiple steps. In general, it is well-recognized that the GROUP BY and HAVING clauses (used in SQL and OQL) are not efficient to answer OLAP queries [15]. For this reason, several proposals were made to enhance the capability of SQL with some features of OLAP models such as dimension hierarchies (see [15], [8]). The query language proposed in this paper overcome these limitations and allows to perform complex operations through the data model structures that include summarization semantics.

7. Conclusions

In this paper, we propose a data model that combines the characteristics of the OLAP and the Object data models in order to represent their functionalities in a common framework. This is achieved by defining only three basic object classes: primitive, regular and composite. In addition to the conventional associations between object classes, the model includes a *summarizable* association that facilitates the specification of classification hierarchy structures. Based on these concepts and well-formed semantics, we define a query language, which uses the anchor concept to enable declarative, non-procedural, query formulation over the combined OLAP-Object databases. The power of the proposed query language is illustrated with multiple query examples. A possible direction for future research is to apply the proposed data model and query language in the framework of XML data representation.

References

[1] *Arbor Software Corporation. (1996). Arbor Essbase*. http://www.arbosoft.com/essbase.html, 1996.

[2] *MicroStrategy, Inc. MicroStategy's 4.0 Prodct Line*. http://www.strategy.com, 1997.

[3] *Red Brick systems, Inc. Red Brick Warehouse 5.0.* 1997.

[4] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. *Proceedings of the 13th International Conference on Data Engineering - ICDE'97*, pages 232–243, 1997.

[5] K. Beyer, D. Chamberlin, L. S. Colby, F. Ozcan, H. Pirahesh, and Y. Xu. Extending xquery for analytics. *ACM SIGMOD 2005*, pages 503–514, 2005.

[6] R. G. G. Cattell. *The Object Database Standard OMDG-93*. Morgan-Kaufmann, San Mateo, California, 1993.

[7] P. Chan and A. Shoshani. Subject: A directory driven system for organizing and accessing large statistical databases. *Conference on Very Large Data Bases*, pages 553–563, 1981.

[8] D. Chatziantoniou and K. Ross. Querying multiple features of groups in relational databases. *Proceedings of 22th International Conference on Very Large Data Bases-VLDB'96*, pages 295–306, 1996.

[9] E. F. Codd, S. B. Codd, and C. T. Salley. Providing olap (online analytical processing) to user-analysts: An it mandate. *Technical report*, 1993.

[10] G. Colliat. Olap, relational, and multidimensional database systems. *ACM Sigmod Record*, 25(3):64–69, 1996.

[11] J.-P. Dittrich, D. Kossmann, and A. Kreutz. Bridging the gap between olap and sql. *Proceedings of the 31st VLDB Conference*, pages 1031–1042, 2005.

[12] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: a relational aggregation operator generalizing group-by, cross-tabs and subtotals. *Proceedings of 12th IEEE International Conference on Data Engineering*, pages 152–159, 1996.

[13] J. Gu, T. B. Pedersen, and A. Shoshani. Olap++: Powerful and easy-to-use federations of olap and object databases. *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 599–602, 2000.

[14] M. Gyssens and L. Lakshmanan. A foundation for multidimensional databases. *Conference on Very Large Data Bases - VLDB'97*, pages 106–115, 1997.

[15] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can hierarchies do for data warehouses. *Proceedings of 25th International Conference on Very Large Data Bases*, pages 530–541, 1999.

[16] H.-J. Lenz and A. Shoshani. Summarizability in olap and statistical data bases. *Proceedings of Ninth International Conference on Scientific and Statistical Database Management-SSDBM'97*, pages 132–143, 1997.

[17] T. B. Pedersen, A. Shoshani, J. Gu, and C. S. Jensen. Extending olap querying to external object databases. *Conference on Information and Knowledge Management-CIKM 2000*, pages 405–413, 2000.

[18] E. Pourabbas and A. Shoshani. The composite olap-object data model. *Technical report, http://www-library.lbl.gov/docs/LBNL/592/29/PDF/LBNL-59229.pdf*, 2005.

[19] A. Shoshani. Olap and statistical databases: Similarities and differences. *Proceedings of 16th ACM Symposium on Principles of Database Systems*, pages 185–196, 1997.