Aggregation Query Model for OODBMS

J. Wenny Rahayu
Department of Computer Science and Computer Engineering
La Trobe University, Bundoora, Victoria 3083, Australia
wenny@cs.latrobe.edu.au

David Taniar
School of Business Systems, Faculty of Information Technology
Monash University, PO Box 63B, Clayton, Victoria 3800, Australia
David.Taniar@infotech.monash.edu.au

Xiaoyan Lu
Information Technology Services
La Trobe University, Bundoora, Victoria 3083, Australia
A.Lu@latrobe.edu.au

Abstract

Query language and querying facilities are critical factors for wide acceptance of Object-Oriented Database Management Systems (OODBMS) in the market. In this paper, we focus on query model on an aggregation hierarchy. We call this query "Aggregation Query". Query on an aggregation hierarchy is unique and differs from general query on association relationships. The latter is often known as path expression query. The difference is analogous to the distinction between association and aggregation in object modelling. In our proposal, we present three important elements of aggregation query, particularly (*i*) aggregation query hierarchy, (*ii*) shorthand path expression for aggregation query utility, and (*iii*) retrieving aggregation tree. Whilst the first element above is adopted from path expression queries, the second element is an extension to general path expression query, and the third element is unique to aggregation, as aggregation resembles a Part-Of relationship, which is more specialized than association relationships.

Keywords: Object-Oriented Queries, OODBMS, Aggregation, OQL, OMG, Composite Objects, and Path Expressions.

1 Introduction

Object-Oriented Database (OODB) systems have great potential to be used in a wide range of applications (Bertino and Martino, 1993; Kim, 1990). However, although it is more expressive, it is also insufficient to make the OODB technology succeed in the market (Cattell, 1991). To be successful, the OODB system must also perform well. One of the key factors for achieving good performance is the development of a good query processing technique similar to SQL (Bertino et al, 1992; Kim, 1989). In this paper, we propose a query model for composite objects in an aggregation hierarchy.

Our previous work (see Taniar and Rahayu (1999)) has classified object-oriented queries into two major categories: *basic queries* and *complex queries*. Basic queries mainly consist of single-class queries, inheritance queries, path expression queries, and explicit join queries. Complex queries, made up of basic query components, can be classified into cyclic queries, semi-cyclic queries and acyclic-complex queries.

Because of the existence of the composite object (whole-part), we redefine the object-oriented query classification to include the notion of composite objects or aggregation – this is then called an *aggregation query*. The relationship between these types of query is shown in Figure 1. In this paper, the terms composite objects and aggregation are interchangeable.

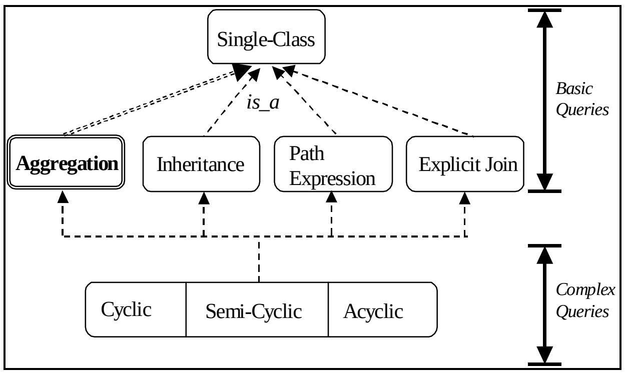

Figure 1. The New Object-Oriented Query Classification incorporating Aggregation Query

In the same way as our original object-oriented query classification (Taniar and Rahayu, 1999), the new classification forms an "is-a" hierarchy. All features of single-class queries, including selection, projection, etc, are applicable to aggregation, inheritance, path expression and explicit join queries; and further complex queries may use the features of the basic queries and single queries.

The rest of this paper is organized as follows: Section 2 describes an aggregation model and its operations on composite objects. Section 3 presents our aggregation query model, including aggregation query hierarchy, shorthand path expression, and retrieving aggregation tree hierarchy. Section 4 compares with existing work in this area. Finally, Section 5 gives the conclusions.

2 Aggregation Model and Operations on Composite Objects: A Foundation

In order to define aggregation queries, it is necessary to clearly express our understanding of composite objects and their operations in OODBMS according to the ODMG standard (Cattell and Barry, 1997).

2.1 Aggregation Model

An *aggregation hierarchy* indicates that there is an IS-PART-OF relationship in the hierarchy (Coad and Yourdon, 1991; Rumbaugh et al, 1991). But what is the real structure of an aggregation hierarchy? What is the core of the aggregation hierarchy? The answer is *composite objects* (Bertino et al 1989).

A composite object has a single root object, and the root references multiple children objects, each through an instance variable. Each child object can in turn reference its own children objects, again through instance variables. A parent object may exclusively/not exclusively own children objects, and as such the existence of children objects is predicated on the existence of their parent. Children objects of an object are thus dependent/not dependent objects. The instance that constitutes a composite object belongs to classes that are also organized in a hierarchy. This hierarchical collection of classes is called *aggregation hierarchy* (Liu, 1992). Aggregation hierarchy can be represented by a composite object schema consist of a single root class and a number of dependent classes.

In Figure 2, we illustrate an aggregation hierarchy schema, or composite objects schema for books. The graphical notation for a composite object model is composed of two basic symbols: *nodes* and *arcs* (Taniar and Rahayu, 1998a). Nodes represent classes, whereas arcs represent relationships. The IS-PART-OF relationship is shown with bold arcs from a composite object class to a component object class. The nodes are labelled. The label inside a node is a class name; the important attributes are shown beside the node. The label of an arc represents the domain of relationship between the two nodes, which is not only for the association but also for the composition. In this composite class schema, a collection type is used, and a notation of [] is used to represent lists.

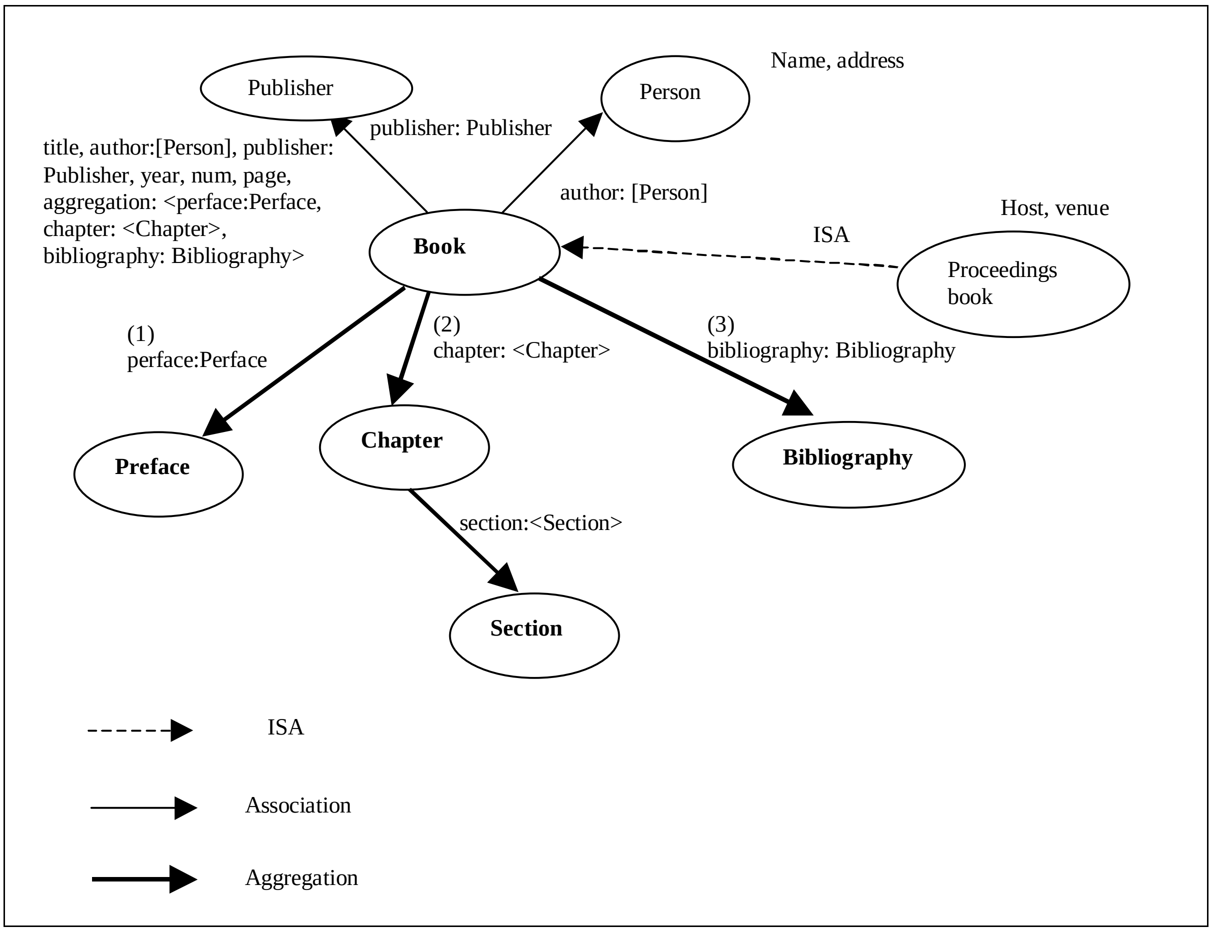

Figure 2. Composite object schema for Book

An ODMG model allows the definition of arbitrarily complex objects as nested objects of arbitrary depth (Cattell and Barry, 1997). An object has a number of attributes; the value of an attribute is itself an object. An object belongs to a class; a class may be a primitive class without any attribute (e.g. integer, string), or it may have any number of attributes. An object with an attribute whose value is an object that belongs to a non-primitive class is a nested object - complex object. An object may have any number of attributes, and any of the attributes may take values from other non-primitive classes. The nested object is a powerful concept. However, it does not imply some special relationships between objects that may be important to a different class of applications. One important relationship that should be superimposed on the nested object is the IS-PART-OF relationship; that is, the notion that an object is a part of another object.

A set of (or a list of) component objects, which form a single logical entity has been called a composite object (Banerjee et al, 1987). Or, a composite object is a collection of related instances that form a hierarchical structure that captures the IS-PART-OF relationship between an object and the parent.

In many applications, the aggregation hierarchy can span an arbitrary number of levels. If a composite object design has component objects that are themselves composite objects, then a two-level aggregation hierarchy is created. This hierarchy could be repeated at several levels of composition/aggregation.

A composite object may be composed of different components in a particular order. The order of occurrence of the component object in the aggregation is significant for the model. For example, a Book is composed of Prefaces, Chapters and Bibliography. In Figure 2, the order of component objects in the book aggregate hierarchy is represented by the numbers with the aggregate label on the arcs.

As we mentioned before, collection type can be shown in the schema; this time the set is represented by the <>. So Book is composed by a preface, a list of chapters and a bibliography, which are in order. (Proceedings book is the subclass of Book, the relationship of which is shown by the dash arcs).

2.2 Applying an Operation to a Composite Object with OQL

In this section, we describe basic operations on composite objects. These operations will become the main kernel of query processing, particularly in retrieving aggregation tree hierarchy (see Section 3.3).

The basic method of extracting an attribute from an object is described as follows. For example, if `e` is an expression of a type (literal or object) having an attribute or a relationship `p` of type `t`, then `e.p` and `e->p` are expressions of type `t`. These are alternate syntaxes for extracting the property `p` of an object `e`. If `e` happens to designate a deleted or a non-existing object, i.e., null, the access to an attribute or to a relationship will return UNDEFINED. For example, `Book_A.title`

Now, we apply an operation with or without parameters to an object according to OQL. If `e` is an expression of a type having a method `f` without parameters and returning a result of type `t`, then `e->f` and `e.f` are expressions of type `t`. These are alternate syntax for applying an operation to an object. The value of the expression is the one returned by the operation or else the object is null, if the operation returns nothing. For example:

```
Book_A->number_of_published
```

This applies the operation `number_of_published` to a book. If `e` happens to designate a deleted or a non-existing object, i.e., null, the use of this method on it will return UNDEFINED. In this way, we have shown how to apply an operation on a composite object with OQL. We treat the aggregation query as querying method. The foundation of that is the defined principle of OQL from ODMG (Cattell and Barry, 1997). For example:

```
Book_A->component [operation criterion]
Book_A->parent [operation criterion]
Book_A->ancestor [operation criterion]
```

The operation criterion is the argument of a method. Applying an operation with parameters can also work on the aggregation query. If `e` is an expression of an object type having a method `f` with parameters of type `t1, t2, …, tn` and returning a result of type `t`, if `e1,e2,……,en` are expressions of type `t1,t2,……,tn`, then `e->f(e1,e2,…,en)` is expression of type `t` that apply operation `f` with parameters `e1,e2,…,en` to object `e`. the value of the expression is the one returned by the operation or else the object is null, if the operation returns nothing. For example:

```
Chapter_A->is-component-of (Book_A)
```

This query calls the operation `is-component-of` on class Chapter for the object `Chapter_A`. It passes on parameter, an object `Book_A` of class Book. The operation will return a Boolean value. For example:

```
Book_A->is-parent-of (Chapter_A)
Book_A->is-ancestor-of (Section_A)
```

The above examples are same as the described previous one.

The following method call: `Preface->order(Book_A)` will call the operation Order. Assume that class Preface is the first class component of class Book, the above will return an integer value 1.

And the following method call: `Section_A->level(Book_A)` will call the operation Level. Assuming that class Section is declared two levels below class Book, the above will return an integer value of 2.

These composite object operations are useful especially in the retrieval of composite objects, either the whole object or its part objects. Details can be found in our proposed aggregation query model.

3 Aggregation Query Model: The Proposed Model

Most object-oriented database systems provide a declarative database query language (Banerjee et al, 1988; Bertino et al, 1992; Kim, 1989). The use of a query language is still considered very important for writing interactive ad hoc queries and for simplifying the C++ code of application programs, although object-oriented databases can often be accessed through code written in an objected-oriented programming language such as C++, Java, the use of a query language is still considered very important for writing interactive ad hoc queries and for simplifying the C++ code of application programs. Because of the success and popularity of SQL relational query language, most proposed object-oriented database query languages have adopted a syntax similar to that of SQL, called *OQL* (*Object Query Language*) (Alashqur, 1989; Cattell and Barry, 1997).

The aggregation query model is based on OQL principles and assumptions in ODMG, that is the aggregation query relies on the ODMG object model, which has utilized the *ODL* (*Object Definition Language*), *OIF* (*Object Interchange Format*) extension of composite object (Cattell and Barry, 1997). Aggregation query also provides high-level primitives to deal with sets of objects but is not restricted to the collection construct. Aggregation query can provide primitives to deal with structures, lists and arrays and treats such constructs with the same efficiency. The result of an aggregation query has a type that belongs to the ODMG type model and thus can be queried again.

In general, OQL is not computationally complete, so aggregation query is simple to use and provides easy access to an OODBMS. Based on the same type system, OQL can be invoked from within programming languages for which an ODMG binding is defined. Apparently, after the operations on composite object are defined, aggregation query can invoke operations programmed in C++ or Java. Because OQL provides declarative access to objects, the aggregation query can be easily optimized by virtue of this declarative nature.

The details of our aggregation query model are explained in the following three subsections, particularly (*i*) Aggregation Query Hierarchy, (*ii*) Shorthand Path Expression, and (*iii*) Retrieving Aggregation Tree. The first one is very similar to path expression queries generally known in an association relationship, but the rests are unique to aggregation query. We describe the first one to give a complete model for aggregation query.

3.1 Aggregation Query Hierarchy

Aggregation query is a query on an aggregation hierarchy. They can be categorized as *forward traversal query* and *reverse traversal query*. Although the terms forward and reverse traversals are borrowed from object-oriented query processing (Bertino and Martino, 1993, Taniar and Rahayu, 1998b), these two queries do not by any means dictate how the query is going to be processed. The direction in this case only refers to the direction of the query hierarchy, in which the direction where we can reach the target class. A *target class* is the class on which the query focuses. A general format of single-class queries is as follows:

```
Select <projection list>
From <var in class>
Where <selection predicates>
```

The main difference between single class query and aggregation query is that the variable `var` may be dynamically bound to a composite objects class/component object class. In the select projection list and selection predicates, we compare the object in the projection list with the object in selection predicates. If the object order from projection list to selection predicates is from component object to composite object, the query is a *forward traversal query*. In the other words, the binding is downward, meaning that the `var` which appears following the `FROM` is searched downward from the composite level to the component level.

Conversely, the opposite order is a *reverse traversal query*. The `var` is searched from the lower level to higher level in the aggregation hierarchy. The binding process in OQL is basically based on the query input and result. An OQL query is a function that delivers an object whose type may be inferred from the operator contributing to the query expression. Every query has an entry point and corresponding exit point. The values of entry-point and exit point decide the distinction of two different aggregation queries. The point is illustrated with the following example (see Figure 3).

The query schema in Figure 3 defines class Book and class Chapter. Both of them are organized in an aggregation hierarchy. Book is the parent of Chapter. An example of forward traversal query is given: "Retrieve the book that has a chapter, whose title is 'Advanced Databases'". The query in OQL is as follows:

```
Select C
From C in B.Chapter, B in Book
Where B.title ="Advanced Databases"
```

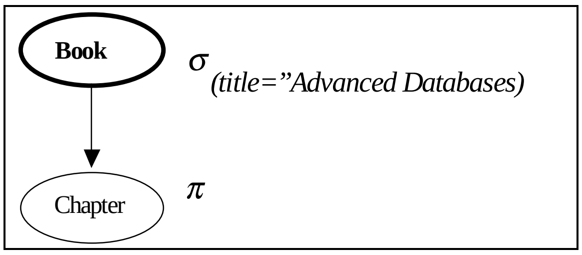

Figure 3. Retrieving “Advanced Database” books that have a chapter

The entry point of this OQL query is in `Where B.title ="Advanced Databases";` the exit point is `Select C.` The graphical notation in Figure 3 is convenient for explaining a forward traversal aggregation query. The bold printed node denotes the entry point, through which the traversal route starts. The target class of the above query is Chapter and the route source is from the Book.

The query scope expansion is a result of a type checking for class Book.

An example of a reverse traversal query is as follows: "retrieve the book which has a chapter which title is 'a taxonomy for object-oriented query'". The OQL is:

```
Select B
From C in B.Chapter, B in Book
Where C.title = "A taxonomy of object-
oriented queries"
```

The entry point of this OQL query is from `Where C.title = "A taxonomy of object-oriented queries";` the exit point is `Select B`. The graphical notation shown in Figure 4 is convenient for explaining a reverse traversal aggregation query. The bold printed node denotes the entry point, through which the traversal route starts. The target class of the above query is Book and the route source is from the Chapter. The query scope expansion is a result of a type checking for class Chapter. The binding in this case is promoted from a component class to a composite class. Since a component class is included in a composite class in a particular hierarchy, through the composite attribute, casting from a composite object to a component object is efficient.

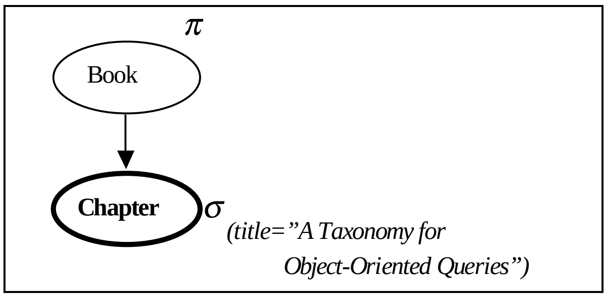

Figure 4. Retrieving books which has a chapter which title is "A Taxonomy for Object-Oriented Queries"

3.2 Shorthand Path Expression with Aggregation Query Utility

As explained above, we can enter a database through a named object. Generally, as long as we get an object, we need a way to navigate from it and reach the right data that we need. To do this, in OQL, a dot (".") notation is commonly used, which enables us to go inside complex objects, as well as to follow simple relationships (Cattell and Barry, 1997). Path expression queries are queries involving multiple classes along the class-domain hierarchies. Class-domain hierarchy is where the domain of an attribute is another class. We can use the select-from-where clause to handle OQL just as in SQL.A general format of path expression expressed in OQL as follows:

```
Select <projection list>
From <var1 in Class,
      var2 in var1.attr1,
      var3 in var2.attr2,
      … …
      varn in varn-1.attrn-1>
Where <selection predicates>
```

The path is explicitly shown in the `From` clause, and projection list is a list of attributes of classes along the path expression. The query starts from the class referenced by var_1. The path grows as attribute $attr_1$, which has a domain of the next class, is pointed by var_2, and so on. In the case of normal query, $attr_1$ is of the first class, $attr_2$ is of the second class, etc, are attributes of a class domain.

So far this is a basic path expression query. In our aggregation query model, we simplify this by introducing a "shorthand" path expression, and the general format is as follows.

```
Select <projection list>
From <var1 in Class,
      varn in var1.attrn>
Where <selection predicates>
```

Here, the $attr_n$ comes from var_n. Based on the composite attribute of a composite object, we can directly jump several levels to the target class. The domain of $attr_n$ of the first class can be a set of objects of its component class or its grandchild class. We define this kind of path expression as a "short-hand" path expression.

An example of a shorthand path expression of aggregation query is as follows: "retrieve books which have a section titled 'Aggregation query' ". The OQL is given as:

```
Select B
From S in B.Section, B in Book
Where S.title = "Aggregation query"
```

In the corresponding query graph (see Figure 5), we can see the entry class is the Section, and scope of query is expanded to the composite class Book through jumping over the Chapter. This example treats an aggregation relationship. In this way, we can navigate directly from a composite object to any object that belongs to this particular aggregation hierarchy.

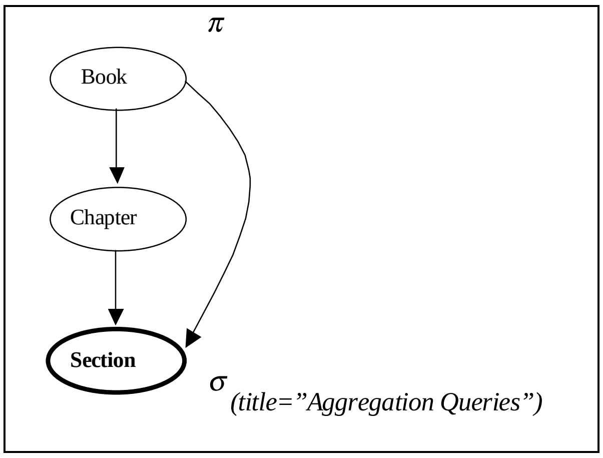

Figure 5. Retrieving books which has a section titled "Aggregation Queries"

3.3 Retrieving Aggregation Tree Hierarchy

An IS-PART-OF relationship normally consists of a root object with multiple components (or parts). Therefore it is critical to provide a mechanism to query the whole composite object tree starting from the root object to its components. In this section, we use the operations of composite objects as previous explained in Section 2.2 to serve this purpose. Assume we would like to retrieve the all/part of the aggregation hierarchy tree as shown in Figure 6.

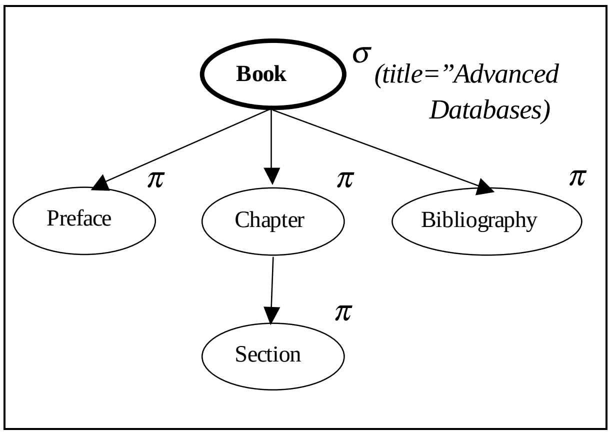

Figure 6. Retrieving “Advanced Databases” book’s components

An example is as follows: "Retrieve the Book's components, the book's name is 'advanced database'". The OQL is following:

```
Select B->component()
From B in Book
Where B.title = "Advanced Databases"
```

By using the operation of composite object, according to the schema, we know the result will retrieve the preface, chapter including sections, and the bibliography, for the book whose title is “Advanced Databases”.

Another example is given to retrieve the part of the aggregation hierarchy, like: "retrieve the publication's title, which has a chapter titled 'A taxonomy of O-O query'". Combined with the operation of composite object, we perform OQL as follows:

```
Select C->parent()
From C in Chapter
Where C.title = "A taxonomy of object-
oriented queries"
```

Or, "retrieve the publication, which has a section named ‘aggregation query’ ". We perform OQL as:

```
Select S->ancestor()
From S in Section
Where S.title = "Aggregation query"
```

So far, we have explained the way to retrieve the whole aggregation hierarchy for the class Book.

```
Select B->component(), B
From B in Book
Where B.title = "Advanced Databases"
```

After inspecting the class Book, apart from the components of `B`, the other attributes of `B` can also be retrieved, like title, author, etc.

Database users always want to make a query with a prerequisite like "whether a particular chapter belongs to one particular book", or "whether one book has one particular section". We included this prerequisite in the selection predicates. For example: "Retrieve the chapters, which belong to the book named 'Advanced Databases'". We give OQL as following:

```
Select C
From C in Chapter, B in Book
Where C->is-component-of(B) = True
And B.title = "Advanced Databases"
```

For different data types in OODBMS, especially like list, array, ..., we can handle the queries on them as well. According to the ODL, OIF specification on data type list, we can query list(). The following query retrieves the second chapter of book “Advanced Databases”.

```
Select C
From C in Chapter, B in Book
Where B->order(C) = 2
And C.title = “Advanced Databases”
```

In this section, we have identified how operations on composite objects in ODMG can be used by aggregation queries to particularly retrieve the whole composite object tree. This facility is unique and very particular to aggregation query only.

4 Comparisons

Most of existing object-oriented query models concentrate on path expression through association relationship. Association is considered different from aggregation (Rahayu et al, 1996), and therefore aggregation query is important. In this section, we present a comparative study with existing works and differences with path expression queries.

4.1 Comparisons with Existing Work

There are many papers on object-oriented queries. The papers by Banerjee et al (1988) and Kim (1989) have been recognized as a pioneer of a model for object-oriented queries, which was based on ORION. It focused on various query models on nested attributes, also known as path expressions. They do not differentiate between association and aggregation, and hence their path expression queries are general, and do not particularly address the IS-PART-OF relationships.

Kifer, Kim and Sagiv (1992) extended Kim’s work (1989) by incorporating complex operations on path expressions (i.e. existential and universal quantifiers), and methods into their query model.

The query model presented in Cluet et al (1990) was influenced by O2 (Lécluse et al 1988). Mainly it covered path expression queries. The paper by Cluet and Delobel (1992) – an extension of their previous work – introduced join queries. The role of join operations was also enhanced as backward traversal of path expression queries, which was implemented in a semi-join. Bertino et al (1992) presented an exhaustive study on object-oriented query languages. Their query models were very much influenced by Kim (1990), where the query is based on attribute-class hierarchy; that is association relationship. No particular distinction between association and aggregation.

The query model by Chan and Trinder (1994) was extensively based on sets. They expressed object-oriented queries in term of object comprehension. Alhajj and Arkun (1992) focused on object algebra.

All of this existing work is based on path expression hierarchy or association. Aggregation has a special semantics (Rumbaugh et al, 1991) and has not been explored its existence in query models.

4.2 Comparisons with Path Expression Queries

Path expression queries are queries whereby traversals are made possible from one class to another through an association relationship (Kim, 1989). This is a general object-oriented query model where join operation is avoided through the use of pointer navigations. This mechanism is regarded as unique and makes object-oriented query processing much faster compared with relational query processing, which purely based on join operations.

From an object-oriented modelling point of view, object relationship exists in various formats, such as inheritance, association, and aggregation (Rumbaugh et al 1991). These relationships are well understood at the design stage. However, in object-oriented queries, the difference between association and aggregation has not been considered critical. In many cases, the terms aggregation and association are used interchangeably. Our proposal in this paper addresses this by introducing a query model for aggregation, as opposed to association.

In Section 3, we have described three extension of aggregation query. The first extension is not purely for aggregation query. It is actually originated in path expression (association) queries. This is included in our aggregation query, simply because some features of aggregation are the same as those of association. We have included the notion of forward and reverse traversal in aggregation query simply for completeness of our proposal.

The second extension in our proposal – that is shorthand path expression for aggregation query utility, has not been explored fully in path expression queries. Like the first extension, this extension is not isolated for aggregation query.

The third extension in our proposal is particularly unique to aggregation query. This is because aggregation has a notion of whole-part (Coad and Yourdon, 1991), which does not exists in the association. Our aggregation query model makes use of operations available for composite objects based on ODMG standard notation. In this extension, we are able to explore the whole aggregation tree or retrieving part components. We found that this extension is critical to expose the notion of whole-part in composite objects.

5 Conclusions

In this paper we have described our query model for aggregation hierarchy. They essentially explored the full capacity of an ODMG extension of a composite object. The examples of aggregation queries have shown that applying aggregation query the semantics of composite objects will be reserved. Our aggregation query covers from forward traversal query to reverse traversal, from retrieving a whole composite object to retrieving a component of a whole composite object.

6 References

ALASHQUR, M. (1989), “OQL: A Query Language for Manipulating Object-oriented Databases” *Proceedings of the Fifteenth International Conference on Very Large Data*, Amsterdam, The Netherlands. Morgan Kaufmann.

ALHAJJ, R. and ARKUN, M.E. (1992), "Queries in Object-Oriented Database Systems", *Proceedings of the First International Conference on Information and Knowledge Management CIKM*, pp. 36-52.

BANERJEE, J., KIM, W. and KIM, K-C. (1988), "Queries in Object-Oriented Databases", *Proceedings of the Fourth International Conference on Data Engineering*, pp. 31-38.

BANERJEE, J., KIM, W., KIM, H-J. and KORTH, H.F. (1987), “Semantics and Implementation of Schema Evolution in Object-Oriented Database”, *Proceedings of the ACM Special Interest Group on Management of Data 1987 Annual Conference*, San Francisco, California, May 27-29, 1987. SIGMOD Record 16(3).

BERTINO, E. and MARTINO, L. (1993), *Object-Oriented Database Systems: Concepts and Architectures*, Addison-Wesley.

BERTINO, E., et al. (1992), "Object-Oriented Query Languages: The Notion and The Issues", *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 3, pp. 223-237.

BERTINO, E, KIM, W and GARZA. J.F. (1989), “Composite Object Revisit”, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, May 31 - June 2.

CATTELL, R.G.G. and BARRY, D.K. (1997), *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publisher.

CATTELL, R.G.G. (1991), *Object Data Management: Object-Oriented and Extended Relational Database Systems*, Addison-Wesley.

CHAN, D.K.C. and TRINDER, P.W. (1994), "Object Comprehensions: A Query Notation for Object-Oriented Databases", *Proceedings of the British National Conference on Databases BNCOD 12*, pp. 55-72.

CLUET, S. and DELOBEL, C. (1992), "A General Framework for the Optimization of Object-Oriented Queries", *Proceedings of the ACM SIGMOD Conference*, pp. 383-392.

CLUET, S., et al. (1990), "Reloop, an Algebra Based Query Language for an Object-Oriented Database System", *Deductive and Object-Oriented Databases DOOD Conference*, W.Kim, et al. (eds.), Elsevier Science Publishers, pp. 313-332.

COAD, P. and YOURDON, E. (1991), *Object-Oriented Analysis*, second edition, Prentice Hall.

KIFER, M., KIM, W. and SAGIV, Y. (1992), "Querying Object-Oriented Databases", *Proceedings of the ACM SIGMOD Conference*, pp. 393-402.

KIM, W. (1989), "A Model of Queries for Object-Oriented Databases", *Proceedings of the Fifteenth International Conference on Very Large Data Bases VLDB*, pp. 423-432, Amsterdam.

KIM, W. (1990), *Introduction to Object-Oriented Databases*, The MIT Press.

LÉCLUSE, C., RICHARD, P. and VÉLEZ, F. (1988), “O2, an Object-Oriented Data Model”, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois, June 1-3, 1988. SIGMOD Record 17(3).

LIU, L. (1992), “Exploring Semantics in Aggregation Hierarchies for Object-Oriented Database”, INFOLAB, Tilburg University, The Netherlands.

RAHAYU, W., CHANG, E., DILLON, T.S., and TANIAR, D. (1996), "Aggregation versus Association in Object Modelling and Databases", *Proceedings of the Seventh Australasian Conference on Information Systems ACIS'96*, Hobart.

RUMBAUGH, J. et.al. (1991), *Object-Oriented Modeling and Design*, Prentice-Hall.

TANIAR, D. and RAHAYU, J.W. (1999), "Chapter 5: A Taxonomy for Object-Oriented Queries", *Current Trends in Data Management Technology*, A. Dogac, M.T.Ozsu, and O.Ulusoy (eds.), ISBN: 1-878289-51-9, Idea Group Publishing, pp. 69-96.

TANIAR, D., and RAHAYU, J.W. (1998a), "Complex Object-Oriented Queries: A Graph-Based Approach", *Proceedings of the ISCA 13th International Conference on Computers and Their Applications CATA’98*, Hawaii.

TANIAR, D. and RAHAYU, J.W. (1998b), "Query Optimization Primitive for Path Expression Queries via Reversing Path Traversal Direction", *Journal of Computing and Information JCI*, vol. 3, no. 1.