객체 해석을 객체 대수로의 변환

이 홍 로[†] · 곽 훈 성[†††] · 류 근 호[††]

요 약

본 논문은 객체 해석을 객체 대수로 변환하는 알고리즘을 제안한다. 이 알고리즘은 해석식을 등가의 대수식으로 변환하고, 이 객체 대수식을 객체 대수 연산자 그래프로 표현한다. 이 변환 알고리즘은 질의의 효율적인 접근 계획을 생성할 뿐만 아니라 등가의 질의식임을 입증할 수 있다.

A Translation of an Object Calculus into an Object Algebra

Hong Ro Lee[†] · Hoon Sung Kwak[†††] · Keun Ho Ryu[††]

ABSTRACT

In this paper, we propose an algorithm to transform an object calculus into an object algebra. The algorithm translates the calculus expression into an equivalent algebra expression, and it maps the object algebra expression to an object algebra operator graph. This translation algorithm not only generates an efficient access plan of queries, but also proves the equivalent expressiveness of queries.

1. Introduction

Object-oriented database systems applied to object-oriented data model can reduce the semantic gap between real world entities and database entities. The database systems have been proposed to the solution that is suitable for providing the data management facilities for applications such as computer aided design, office information system, and multimedia systems. Most of the proposed solutions have focused on complex objects together generation inheritance[1, 2, 3].

† 정 회 원: Department of Computer Science, Chungbuk National University, Korea
†† 종신회원: Department of Computer Science, Chungbuk National University, Korea
††† 종신회원: Department of Computer Engineering, Chonbuk National University, Korea

논문접수: 1995년 9월 5일, 심사완료: 1996년 3월 12일

Bancilhon[4, 5] specified a calculus for a complex object that is based on the orthogonality of a semantic data model, and also Codd[6] divided the first order logical calculus into the relational tuple calculus and the relational domain calculus. Roth[7] investigated the nested relational calculus that was extended to a relational data model. Osborn[8] described an object algebra which views objects as passive data not supporting encapsulation. His algebra allows aggregate or set objects to be disassembled and then reuses in later stages of a query. Shaw[2] developed the object algebra which support encapsulation, but only allowed unary method to be used in qualifying algebra operators. Also, he not only defined object-oriented data model based on ENCORE[9] which supported a tuple constructor such the nested relational data model, but also specified algebra operators "Dup-

Eliminate" and "Coalesce" to remove a redundant objects which are generated newly. Straube[10] defined an object calculus and an algebra expression for the formal query processing which supports encapsulation. He proposed the atom place routine for an algebra operator graph visually and an algebra expression, which translates calculus into algebra. These investigations are restricted to rewriting rules of the algebra expression and the algebra operator graph of simple objects. Therefore, the object-oriented systems need a formal query processing as well as an algebra translation algorithm that should not generate the anomaly of a schema evolution by aggregation inheritance.

In this paper, we specify an object-oriented calculus for the declarative query processing based on object-oriented query model together with aggregation inheritance. Also, we propose a method which translates the object calculus into the object algebra for the procedural query processing. The algebraic operators support object-preserving and object-creating operations, and analyze a prenex normal form of an object calculus, and lead to the corresponding algebra expression. The generating condition of algebra operators is described in order to compute the translated algebra expression. The stages of the translation algorithm are explained.

This paper proceeds as follows. Section 2 specifies the object-oriented query model. Section 3 specifies the object calculus and object algebra. Section 4 presents an algebra translation algorithm of queries. Finally, section 5 contains conclusion and future work.

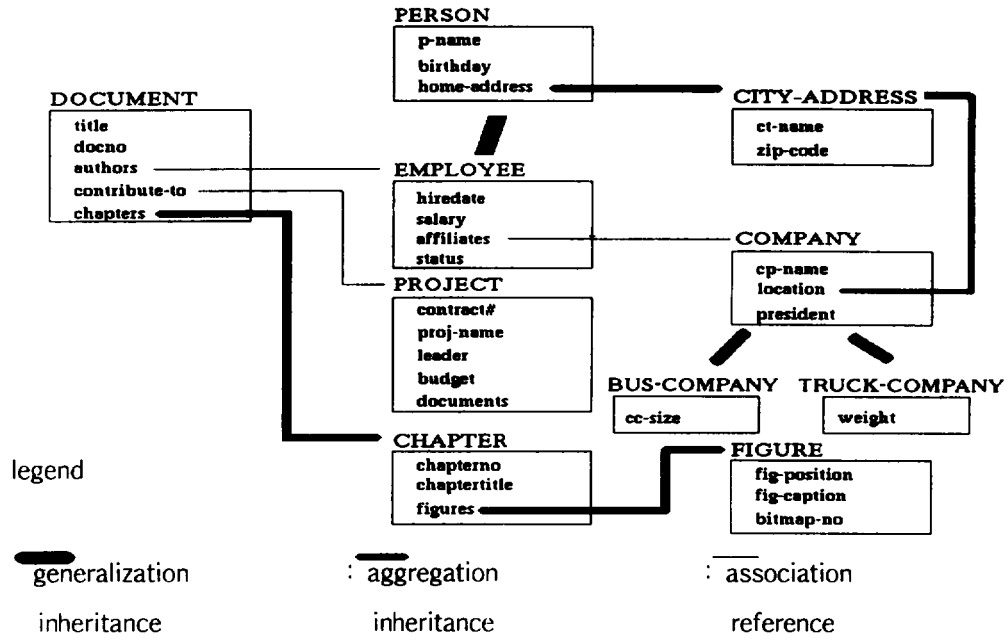
2. Object-oriented query models

For this query model, we not only use the common object-oriented concepts such as the object, class, and inheritance, etc., but also distinguish aggregation relationship from the arbitrary association relationship and incorporate aggregation inheritance in the aggregation class hierarchies.

A complex object is accomplished by the recursive orthogonality of atomic objects and constructed objects, and so on, set, tuple, and list, etc.[4, 11]. All objects that have the same attributes for the data part and the same methods for the operating part are collectively referred as a class. They are arranged in such a way that a class(subclass) inherits the attributes and methods from other class(superclass). In the class hierarchy, the subclasses have a generalization relationship. In the class composition hierarchy, the attribute-domain relationship divides the association relationship into the aggregation relationship. The former is used to model arbitrary associations of objects, where the transitive and antisymmetric features have **"Has-A"** relationship between two independent classes. The latter is used to model an object composition relationship, which carries on the **"Is-part-of"** semantics between individual objects. It is a tightly coupled form of association with some extra semantics such as transitivity and antisymmetric properties. Therefore, the class composition hierarchy is a semi-lattice structure[12].

From the polymorphic point of a view, the subclass inheritance reflects a universal polymorphism whose polymorphic characteristics are inherent from all the circumstances, and the coercion is not necessary. Whereas an aggregation inheritance exhibits an ad-hoc polymorphism which exists only at the syntactic level and disappear at the close range[13].

Consider a database schema for complex objects as a collection of schema rules of the form $C=(A_1 \ldots A_n)$, $(C(A_1 \ldots A_n)$ for short), where C is called a class and A_i is called an attribute name. $A_i(1 \le i \le n)$ is called a higher-order attribute function of the class C (Hfn(C)), if it is an atomic set-valued attribute name or if it appears on a left side of the same schema rule. Otherwise, A_i is called a zero-order attribute function of the class C(Zfn(C)) which returns a single atomic value for each input. In the attribute-domain relationship, a set of all attributes are called Lfn(C) if it is left side attribute function. That is, $Lfn(C)=Zfn(C)$



(Fig. 2.1) A Schema of a Class Composition Hierarchy

+Hfn(C). In the class composition hierarchy, an attribute function space of a given class C is denoted by Fspace(C).

Let A be a member of the Fspace and Path(A[C]) denoting the path expression of an attribute function A of an class C. Then, the aggregation path expression of the attribute function A with respect to the class C is defined as Path(A[C]) = C.A without intermediate attributes by an aggregation inheritance. And it traverses an acyclic query graph. The association path expression denotes "$C.A = C_i$. **self**" if C_i is a referenced class and **"self"** is a keyword used by Smalltalk, and it traverses cyclic query graph[12, 14].

A schema of a class composition hierarchy is shown in Figure 2.1. The major benefits of the aggregation inheritance are as follows. First, it is increasing of the reusability of the schema specifications and database programs using the combination of aggregation inheritance with the subclass inheritance. Second, the queries with aggregation inheritance do not require to provide the explicit navigational operators. Finally, it does not generate the anomaly of schema evolution.

3. Object calculus and Object algebra

For the formal query processing, a representation of a suitable query has to include an expressiveness of queries and also specifies an object calculus and an object algebra for the internal representation of queries.

3.1 Object calculus

The object calculus defines a query result through the description of its properties. In order to define a complex object calculus that supports the properties of a class composition hierarchy, the forms of calculus are restricted to the domain of a database schema [12].

The used object calculus is similar to the tuple calculus of Codd[6]. The notation of the object calculus

is $\{O|\Phi(O)\}$, where O is a complex object variable within a database, and Φ is a formula that is composed of atoms. The representation of the complex object calculus consists of two parts: a target list and a selection expression. The target list defines the free variables in the predicate and it specifies the structure of the resulting class. The selection expression specifies the contents of the class resulting from the query.

The atomic formula is a predicate which returns a true or false object. As comparison operators compare with two terms in the selection expression, they are boolean expressions that carry out the logical comparison between two attributes and/or the attribute and constant. The comparison operators are scalar operators(==, =, $\neq$, $<$, $\leq$, $>$, $\geq$), set inclusion operators(==, =, $\neq$, $\subset$, $\subseteq$, $\supset$, $\supseteq$), and set membership operators($\in$, $\notin$, $\ni$, $\not\ni$).

A formula for selecting a collection of objects from a database, denoted by Φ, is recursively defined as follows.

【definition 3.1】 WFF(Well-Formed Formula)

i) Φ = true(or false) is formula.

ii) $\Phi = A\,u\,B$, $A\,u\,c$ and $c\,u\,A$ is formula, where,

(a) A and B are comparable attributes returning atomic values, and u is in $\{==, =, \neq, <, \leq, >, \geq\}$.

(b) A and B are comparable attributes returning set values, c is an instance of the attribute A, and u is in $\{\in, \notin, ==, =, \neq, \subset, \subseteq, \supset, \supseteq\}$.

iii) If Φ, Φ_1 and Φ_2 are formulas, then the disjunction $\Phi_1 \vee \Phi_2$, the conjunction $\Phi_1 \wedge \Phi_2$ and the negation $]\Phi$ are formulas.

iv) If Φ is a formula and x is free variable in Φ, then the $(\exists x)\Phi[x]$ and $(\forall x)\Phi[x]$ are formulas.

v) Formula is closed to parenthesis. If there is no parenthesis, then the priority of operators is in the order of $\exists/\forall$,], $\wedge$, and $\vee$.

vi) Nothing else is a formula. □

The formulas in the forms of i) and ii) are so-called boolean expressions(atoms). Besides, all the variables used in the formulas are loosely typed, i.e., bound to a database schema class. When Φ is empty, we treat it the same as a true formula.

Then, here is examples of the object calculus expressions. By means of user's query language, this work introduce NxOQL(Next Object Query Language) which is SQL-like syntax proposed in [12]. Example 3.1, 3.2 and 3.3 represent the queries for the aggregation inheritance, association reference and returning sub-objects, respectively.

【Example 3.1】 "Find documents with a figure under the caption "Multimedia diagram"."

Its NxOQL syntax and object calculus are expressed as follows:

SELECT d
FROM DOCUMENT:d
WHERE d. fig_caption = "Multimedia diagram"

$\{t|t \in DOCUMENT(d) \wedge d.fig\_caption = \text{"Multimedia diagram"}\}$

【Example 3.2】 "Find all the employees who are also the president of the company."

Its NxOQL and object calculus are expressed as follows:

SELECT e
FROM EMPLOYEE:e, COMPANY:c
WHERE e. **self** = c.president

$\{\mu_e | \exists t_1(\mu_e \in EMPLOYEE(e)(e \wedge t_1 \in COMPANY(c))) \wedge e.\textbf{self} = c.president\}$

【Example 3.3】 "Find the name, birthday and salary of employees whose salary is less than $30,000."

Its NxOQL and object calculus are expressed as follows:

SELECT e.p-name, e.birthday, e.salary
FROM EMPLOYEE:e
WHERE e.salary < 30000

{t| ∃t_e∈EMPLOYEE:e∧t=[p-name, birthday, salary]∧e.salary < 30000}

After all, from those examples for object calculus expressions, we can show that the object calculus satisfies the closure properties of query language.

3.2 Object algebra

This section describes an object algebra for the procedural processing of queries. For the closure property of the object algebra, the output classes of each operators are described and the relationship with the input objects is analyzed through the semi-lattice structure of "Is-A" class. As the input and output objects of the operation define a set of objects, each operation can be applied to the set-valued attributes or query results. The operation can be defined formally.

For the procedural processing of the object-oriented queries, the algebra operators which are defined by Ling[14] are used. The algebra operators of complex objects are specified below.

Selection operators are composed of unary selection and multiple-operand selection. **Unary selection** returns objects that satisfy the predicate condition(F). Its inputs are the instances with various attribute function names(i.e., a zero order attribute name and it through the generalization/aggregation relationship) in a class composition hierarchy(R_i). The unary selection is denoted by $\sigma(F, R_i)$. And, **multiple-operand selection** returns objects which satisfy the predicate condition(F). Its inputs are the instances with an attribute function name through the association relationship in a class composition hierarchy(R_1, $\langle R_2, ..., R_n\rangle$). The multiple-operand selection is denoted by $\sigma^+_F(R_1, \langle R_2, ..., R_n\rangle)$.

Projection operator of an object is a class-specific unary operator to the collection of complex objects. The projection returns objects that are only related with the sub-attributes(X) from the all attributes in class hierarchy(R). The projection is denoted by $\Pi(X, R)$.

Join operator extends the nested relational join in order to manipulate queries on the value-based objects. In the classes with association reference, an object identifier use for preserving the reference projection as well as removing the object reference, which specifies the special operator that is called "followup". In two class R_1 and R_2, join predicate condition (F) is t. A$\mathcal{U}$t'. B, where A and B is comparable attribute function name, $t\in R_1$ and $t'\in R_2$. $\mathcal{U}$ is comparison operators. The **join** of object is denoted by $\otimes_F(R_1, R_2)$. **Followup** operator is used to eliminate the attributes of a class by instances which are referred to object identifiers. In two classes R and S together with association relationship, Lfn(R) and Hfn(S) are the zero order attribute function name of class R and the higher order attribute function name of class S, respectively. And, B is the referenced attributes of R. If B refers to the class S, and X is Lfn(R)∪Lfn(S)−{B, oid(S)}Lfn(R), then the followup is denoted by $followup_B(R) =_{def} \Pi(X, \otimes_F(R, S))$.

Set operators are composed of union, intersection, and difference. All of them are related with the universal objects and include the comparison and set membership based on object identifier. And, they are object-preserving operators. In two classes R_1 and R_2, union, intersection, and difference are denoted by $\cup(R_1, R_2) =_{def} \{t|t\in R_1 \vee t\in R_2\}$, $\cap(R_1, R_2) =_{def} \{t|t\in R_1\wedge t\in R_2\}$, and $-(R_1, R_2) =_{def} \{t|t\in R_1\wedge t\notin R_2\}$, respectively.

Although these operators itself can be applied to the higher order attribute function names repeatedly in other aggregation hierarchy of complex objects, the **nest** and **unnest** operators are still necessary to reconstruct the resulting objects of queries. The nest is to reconstruct a new single tuple class by means of the nested relationship which the attribute of a class

refers to the attributes of other classes. The nest is denoted by nest(A_1, A_2, ..., $A_n \rightarrow$ B, R). The unnest is to reconstruct the tuple object by means of eliminating the attribute of another class which is referenced by object identifiers. The unnest is denoted by unnest ($A \rightarrow$ R).

4. A translation of an object calculus into an object algebra

In this section, the proposed algorithm is discussed. The proposed algorithm translates an object calculus into an object algebra and its algebra operator graph for the procedural query processing. Therefore, the entire stages of the translation algorithm are described with an example.

4.1 Algebra translation algorithm

The algebra translation algorithm analyzes a prenex normal form of calculus, and it translates a calculus into the corresponding algebra and then it generates an object algebra expression as a whole. For the efficient representation and evaluation strategies, the algorithm represents to translate the object algebra expression into an object algebra graph. The concrete stages of the algorithm are expressed as follows. First, range-separable WFFs are applied to a calculus, in order to specify the scope of objects for each object variable. Second, a calculus expression is converted to cartesian product, selection, join, and projection operations according to the kinds of algebra operators about the object variables which are associated with an each of the range clauses.

Consider complex object variables(ov_1, ov_2, ...) whose scopes are in a class composition hierarchy. The j'th class R_j is represented by unary predicate P_j such that $P_j(ov_i)$ is true if the object of the i'th complex object variable is in the j'th class(i.e., $ov_i \in R_j$). The term such as $P_j(ov_i)$ is called a range term.

Also, comparison operators are introduced for predicates with two arguments, and they are used to form join terms. One argument of the join term is a component of the complex object, and it is denoted by $ov_i[N]$, where N is an integer constant which specifies the column number of the tuple including desired attribute object. The other argument of the join term must be another component of the complex object or a constant. The terms that have two components of the complex objects are called join predicates, and the terms that include constants are called selection predicates.

A well-formed formula is formed the join terms and range terms out of "$\wedge$", "$\vee$", and "$\neg$" operators together with the quantifier "$(\exists ov_i)$" and "$(\forall ov_i)$", where ov_i is a range variable. Now let's consider the special class of a well-formed formula which is range-separable. These are of the form

$$U_1 \wedge U_2 \wedge O \ldots \wedge U_n \wedge W$$

, where W and each U_i are well-formed formulas, and they must be at least one $U_i (n > 0)$. W may be empty. Each Ui is a unique range variable and consists of range terms without quantifiers. W may contain both join terms and range terms. If W contains quantifiers, it must be coupled with a range term, and it takes the forms of "$(\exists ov_i)(P_j(ov_i) \wedge W)$". Therefore, every complex object variables in W must have its range given by either U_i or a quantifier.

Two other conditions are imposed on U_i. First, if the same range variable is used for two different predicates, such as in $P_2(ov_1)$ and $P_3(ov_1)$, then, of course, the corresponding classes must be the same type. Second, no U_i may start with a logical negation, although the combination of "$\wedge$" and "$\neg$" is permissible in U_i. This means that the range must be given explicitly.

For translating an object calculus into an object algebra expression, the stages are as follow:

Stage 1. prenex normal form of calculus

The first step converts W to a prenex normal form

with quantifiers at the front of the calculus expression. Remember U_i so that the complex object variable ov_i is arranged in ascending order, i.e., U_2 refers to ov_2, and U_3 refers to ov_3, and so on. Thus, the query takes the form of

$$t_1 : U_1 \wedge U_2 \wedge \ldots \wedge U_n \wedge (Q_1\ Q_2 \ldots Q_q)W$$

, where t_1 is the list of components of the complex object required in the result of query, and the range variable "p" is governed by range terms in the U_i and the variable "q" is governed by guantifiers Q_1 to Q_q. Therefore it contains only join terms.

Stage 2. cartesian product of range clause

From the U_i and Q_i, it forms the cartesian product S of the set extensions which give the complex object variable ranges. Inside each U_i, replace $P_j(ov_i)$ by the class R_j, replace "$\vee$" by the union, "$\wedge$" by the intersection, and "$\neg$" by the difference. For each Q_i coupled with a range term $P_k(ov_i)$ write down the class R_k. Thus,

$$P_2(ov_1) \wedge (P_1(ov_2) \vee P_2(ov_2)) \wedge \neg(p_1(ov_3) \wedge P_2(ov_3)) \wedge (\forall ov_4)(P_3(ov_4) \wedge W)$$

the corresponding product of range is

$$S = (\otimes(\otimes(\otimes(R_2, (R_1 + R_2)), (R_1 - R_2)), R_3)$$

Stage 3. selection operations

For each join term in W, connecting the range terms together with Q_i or U_i in a class with the logical operators "$\vee$", "$\wedge$" and "$\neg$" as an appropriate selection condition(F), they are converted to the unary selection operator as follows:

$$\sigma(F, R_i)$$

Also, in class composition hierarchy with association relationship, connecting the range terms together with Q_i or $U_i (1 \le i \le n)$ with the selection condition(F), they are translated to the multiple operand selection operator as follows:

$$\sigma^+(R_1, \langle R_2, \ldots, R_n \rangle)$$

Stage 4. join operations

At the stage 2, two classes R_1 and R_2 are joined for satisfying the predicate condition(F). First, if there is the calculus of attribute function space together with an aggregation relationship, the calculus expression is converted to the object-join operator as follows:

$$\otimes_F(R_1, R_2)$$

Second, at two classes R and S in a class composition hierarchy, if there is the space together with association relationship, the calculus expression is translated into the followup operator as follows:

$$\text{followup}_B(r) =_{def} \Pi(X, \otimes_F(r, s))$$

Stage 5. reconstruction operations

Let R be a class. the unnesting list is {A}. At two classes with an aggregation relationship, if there is the calculus which reconstructs the attribute function space of a class by eliminating the relationship between the two classes, the calculus expression is converted to the unnest operator as follows:

$$\text{unnest}(A \rightarrow R)$$

Let R be a class, the nesting list $\{A_1, \ldots, A_n\}$ and the new attribute function is B. At two classes with an aggregation relationship, if there is the calculus which reconstructs the attribute function space of a class by connecting the relationship between two classes, the calculus expression is translated into the nest operator as follows.

$$\text{nest}(A_1, A_2, \ldots, A_n \rightarrow B, R)$$

Stage 6. projection operation

For each quantifier Q_i(i = 1 to q), apply the projection operation to S. Let the column list for r_i be b_1, and the corresponding class, taken from the range term coupled with the quantifiers, be Rk. Then,

if $Q_i = (\exists R_i)(P_k(R_i) \wedge W)$, S := S discarding b_1

Stage 7. the generation of object algebra expression

Project to the target list t_1, thus,

$S := \Pi(S, t_1)$

Stage 8. the generation of object algebra operator graph

At the query process of object-oriented databases, the algebra operator graph represents visiblely the algebra expression which is generated by stage 8. It also is used to implement an efficient evaluation strategy. Thus, the object algebra operator graph is necessary not only to represent visiblely the generated algebra expression, but also to access efficiently the object about a query. The condition is as follows.

1) The object algebra expressions are represented by the graph. In the graph, nodes are the algebra operators and edges are the set of objects

2) The algebra operators (σ_F, $\otimes_F$) are bound to the predicate of join terms. The predicates consist of conjuncts of atoms. The element of each conjunct($\wedge$) are referred to as several object variables.

3) The element consisting of multiple atoms of the predicates is represented by the section of object-join operator and followup operator, respectively, according to the aggregation relationship and/or the association relationship.

4) According to existence and/or non-existence of the aggregation relationship, two classes use the nest operator in order to reconstruct the aggregation relationship. Similarly, the unnest operator is used to eliminate the aggregation relationship between two classes.

4.2 Comprehensive example of the algebra

We have a comprehensive example to show the correctness of each step of the proposed algorithm. The operators explained in section 3.2 are generated and the object algebra graph is generated after executing final step.

【Example 4.1】 "Find the project name, the contract#, and all the documents (including title and authors) that are contributed to the project, in which the budget is grater than $100,000."

The NxOQL syntax and its object calculus are expressed as follows:

SELECT d.title, d.authors, p.proj-name, p.contract#
FROM PROJECT:p, DOCUMENT:d
WHERE d.oid ∈ p.documents AND p.budget > 100000

$\{t \mid \exists t_1 \in PROJECT(p), \exists t_2 \in DOCUMENT(d), \exists t \in t_r:$
$t_r[Lfn(p)] = t_1[Lfn(p)] \wedge t_r[Lfn(d)] = t_2[Lfn(p)]$
$\wedge d.oid \in p.documents \wedge p.budget > 100000$
$t = [title, authors, proj\text{-}name, contract\#]\}$

Let these expressions be applied to the algebra translation algorithm. According to the stage 1 and the stage 2, the cartesian product that exists in the prenex normal form and range clause is expressed as follows:

$\otimes(PROJECT, DOCUMENT) = \exists t_1 \in PROJECT(p)$
$\wedge \exists t_2 \in DOCUMENT(d)$

The cartesian product of this range clause can use the predicate condition by means of the stage 3. Thus, the selection operator is expressed as follows:

$\sigma(budget > 100000, PROJECT)$
$= \{t \mid t \in PROJECT(p) \wedge t.budget > 100000\}$

Here, if predicate condition F_1 is budget > 100000, then the selection operator is denoted as follows:

$t_1 = \sigma(F_1, PROJECT)$

IF the attribute "DOCUMENT" and the result of selection operation are joined in the stage 4, it is expressed as follows:

$$\otimes d.oid \in p.documents(t_1, DOCUMENT) = \{t \mid \exists t_r \in t_1 \wedge \exists t_2 \in DOCUMENT(d) : t[p] = t_r[p] \wedge t[d] = t_2[d] \wedge d.oid \in p.documents\}$$

Here, F_2 is d.oid ∈ p.documents. Thus, the join operation is denoted as follows:

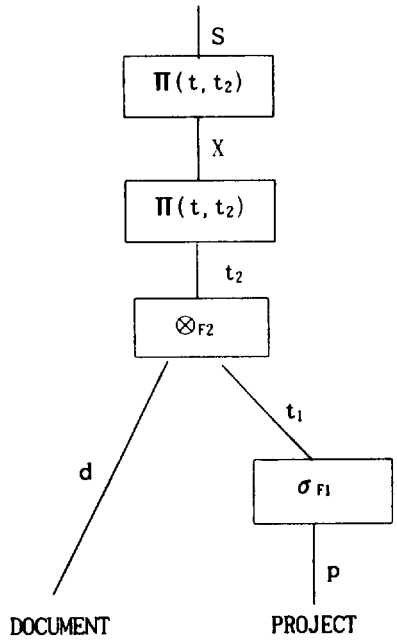
$$t_2 = \otimes_{F2}(t_1, DOCUMENT)$$

As the result of this join operator can be obtained to the association relationship, it must be converted to followup operator as follows:

$$followup_{documents}(t_1) = (X, t_2)$$

Where X is Lfn(p) ∪ Lfn(d)-d.oid.

The result of the target list is expressed as follows:



(Fig. 4.1) An object algebra operator graph

$$S = \Pi(t, t_2)$$

Where t is [title, authors, proj-name, contract#]

According to the stage 7, the algebra expression which is generated by calculus expression is denoted as follows:

$$S = \Pi(t, \Pi(X, \otimes_{F2}(\sigma(F_1, PROJECT), DOCUMENT)))$$

By means of the stage 8, the algebra operator graph, which is generated by algebra expression, is denoted by Figure 4.1.

After all, the object calculus based on the object-oriented query model together with the aggregations relationship can be translated into the algebra expression defined by Ling[14], and the entire stages of the translation algorithm have been shown in the previous example. Therefore, it has proved the equivalence of expressiveness as well as generating the access plan for the efficient query optimization[15].

5. Conclusions

In this paper, we have proposed the algorithm which translates the object calculus into the object algebra expression. The method is an approach which generates the efficient access plan of queries on object-oriented query models. Also the calculus and the algebra are specified, which are based on the query model, for the formal query processing of object-oriented database. The object calculus has a property of the aggregation inheritance for the declarative query processing.

The translation algorithm is summarized as follows. First, in order to guarantee the closure property of query operations, the prenex normal form is analyzed initially. Second, the prenex normal form is translated into object algebra operations. Third, the object algebra expression and its operators are generated. Finally, an object algebra operator graph is generated after executing the last step.

As a result, it has not only proved the equivalency of expressiveness, but generated the access plan for the efficient optimization of queries.

The proposed translation algorithm can be applied to the conventional object-oriented database systems for a new object-oriented query processing system. The development of new object algebra operators and applications to the conventional systems will be studied for future work.

REFERENCES

[1] Kim, W. and Lochovsky, F.H., ed., *Object-Oriented Concepts, Databases, and Applications*, Addison Wesley, Reading, MA, 1989.

[2] Shaw, G. and Zdonik. S., "An Object-Oriented Query Algebra," in Proc. DBPL, at Salisham Lodge, Oregon, 1989.

[3] Deux, O., "The Story of O2," IEEE TKDE, Mar. 1990, pp.91-108.

[4] Abiteboul, S. and Hull, R., "IFO:A Formal Semantic Database Model," in Proc. ACM PODS, 1984, pp.119-132.

[5] Bancilhon, F. and Khoshafian, S., "A Calculus for Complex Objects," In Proc. PODS, 1986, pp. 53-59.

[6] Codd, E. F., "A Database Sublanguage Founded on the Relational Calculus," in Proc. of the ACM-SIGFIDET Workshop, Data Description, Access, and Control, at San Diego, Calif., Nov. 1971, pp.35-68.

[7] Roth, M. A., Korth, H. F., and Silberschatz, A., "Extended Algebra and Calculus for non-INF Relational databases," ACM TODS, Vol.13, No. 4, 1988, pp.389-417.

[8] Osborn, S. L., "The Role of Polymorphism in Schema Evolution in an Object-Oriented Database," IEEE TKDE, Sep. 1989, pp.310-317.

[9] Elore, P., Shaw, G.M., and Zdonic, S. B., "The ENCORE Object-Oriented Data Model," Technical Report, Brown Univ., Nov. 1989.

[10] Straube, D., "Queries and Query Processing in Object-Oriented Database Systems," Ph.D Thesis, the Univ. of Albert at Edmonton, 1991.

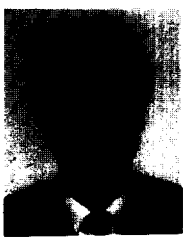
[11] Kim, W., "A Model of Queries for Object Oriented Database," in Proc. VLDB, 1989, pp. 423-432.

[12] Lee, H.R, "A Logical Optimization of Queries in Object Oriented Database System," Ph.D Thesis, Chonbuk National Univ., Aug. 1994.

[13] Cardelli, L. and Wegner, P., "On Understanding Types, Data Abstraction, and Polymorphism," ACM Computing Survey, Vol.17, No.4, Dec. 1985, pp.471-522.

[14] Ling, L., "A Recursive Object Algebra Based on Aggregation for Manipulating Complex Objects," North-Holland, Data & Knowledge Engineering, Vol.11, 1993, pp.21-60.

[15] Demuth, B., Geppert, A., and Gorchs, T., "Algebraic Query Optimization in the CoOMS Structurally Object-Oriented Database System," Query Processing for Advanced Database Systems, Ed. Freytag, J. C., et al., MORGAN KAUFMANN PUBLISHERS, 1994, pp.121-142.



이 홍 로

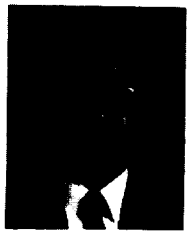
1984년 전북대학교 전기공학과 졸업(학사)

1986년 전북대학교 대학원 전자계산기 전공(공학석사)

1994년 전북대학교 대학원 전산응용공학 전공(공학박사)

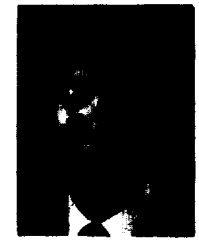
1994년~현재 충북대학교 컴퓨터과학연구소 연구원

관심분야: 객체지향 언어, 객체지향 데이타베이스, 질의처리



곽 훈 성

1971년 전북대학교 졸업(학사)
1973년 전북대학교 대학원 전자공학 전공(공학석사)
1978년 전북대학교 대학원 전자공학 전공(공학박사)
1981년~1982년 Univ. of Texas 객원교수
1992년~1995년 전북대학교 전산소장
1978년~현재 전북대학교 컴퓨터공학과 교수
관심분야: 영상처리, HDTV, 패턴인식, 인공지능, 멀티미디어 및 객체지향 시스템



류 근 호

1976년 숭실대학교 전산학과 졸업(학사)
1980년 연세대학교 산업대학원 전산전공(공학석사)
1988년 연세대학교 대학원 전산전공(공학박사)
1976년~1986년 육군군수 지원사 전산실(ROTC장교), 한국전자통신연구소(연구원), 한국방송통신대 전산학과(조교수) 근무
1989년~1991년 Univ. of Arizona 연구원
1986년~현재 충북대학교 컴퓨터과학과 교수겸 컴퓨터과학연구소장
관심분야: 시간지원 데이타베이스, 시공간 데이타베이스, DBMS 및 OS, 객체 및 지식베이스 시스템