International Journal on Emerging Technologies 1(1): 75-79(2010) ISSN : 0975-8364

The dynamic roles of object processing query in real-time object oriented distributed database architecture

Sajidullah S. Khan and M.S. Ali***

**Prof. Ram Meghe Institute of Technology & Research, Badnera Amravati, (MS) INDIA*
***Prof. Ram Meghe College of Engineering & Management Badnera Amravati, (MS) INDIA*

(Received 5 Dec., 2009, Accepted 15 Jan., 2010)

ABSTRACT : The paper presents the general framework of a distributed object query language for a database based on an object-oriented with dynamic object roles. A number of questions both for the language and its database (the so-called object-based approach) are covered, among others, managing a database stack, binding names, and casting. The paper also considers the issue of query optimization for the language. The discussion is illustrated with examples and figures.

Keywords : object-orientation, dynamic object roles, distributed databases, object query processing

I. INTRODUCTION

The object-oriented database is a generalization of the relational one and is believed to eliminate many of its flaws through incorporating modern concepts. One of them is dynamic object roles with the following characteristics [1]: a role has its own properties and behavior; an object can acquire and abandon roles dynamically without changing its identity; an object can play different roles simultaneously; an object can play the same role several times.

The current object-oriented database can express static properties, *e.g.*, the fact that an employee "is a" person. However, it is more precise to say that a person "becomes" an employee for a period of time and later he/she terminates the employee role. Moreover, a person can be an employee two or more times.

In general, the concept of dynamic object roles assumes that an object (a so-called "owner-object" or "owner") can be associated with other objects (so-called "role-objects" or "roles"). Roles are treated as objects with some additional special features such as: a role cannot exist without its owner; deleting an owner implies deleting all of its roles; roles can exist simultaneously and independently. As an object, a role can have its own additional attributes, behavior, etc. Moreover, roles can be further specialized as subroles, sub-subroles, etc, *e.g.*, specialization of an *Employee* role can be a *Superior* role.

Dynamic roles can significantly support conceptual model and, in comparison to the classical object models, do not lead to the anomalies and limitations of multiple, repeating, and multi-aspect inheritance. For instance, two roles (of the same object) can contain attributes and methods with the same names without implying any conflict. This is a fundamental difference in comparison to the concept of multiple inheritances.

Associations between objects can connect not only owners with owners, but also owners with roles and roles with roles. For example, a *works-in* association can connect an *Employee* role with a *Company* owner-object.

A database with dynamic object roles involves two kinds of inheritance: static and dynamic. Static inheritance is the inheritance between classes in the traditional sense, where the properties of a class are imported by its subclasses at compile time. The mechanism of dynamic inheritance is similar with the following difference: it concerns objects whose values are imported by their roles at run time.

To a big degree dynamic roles can be an important paradigm for object-oriented databases and their query languages constructed e.g. in the spirit of the ODMG standard [2]. The concept is already present (under another name and with specific semantics) in the standard SQL3 (abandoned) and its successor SQL1999 [3].

In spite of many proposals, the problem of object-oriented query languages is still considered open. The ODMG standard with its OQL (*Object Query Language*) is criticized by some specialists due to its inconsistencies, lack of precise specification, etc. [4]. Therefore in our research on dynamic object roles we are proposing – the *Object-Oriented Database* (OOD) along with its Query Language (OODQL). In our opinion it does not have the disadvantages of its competitors. Therefore in this paper we assume at least a general familiarity with these concepts; the reader is referred to [5, 6].

In this paper, we have using dynamic object roles into OOD and also discuss how this concept can be incorporated into OODQL.

This concept of the object roles has been implemented in our prototype architecture of the object oriented database. Currently we are working on a prototype

architecture of object oriented distributed database where we intend to implement the ideas presented here.

II. OODQL WITH DYNAMIC OBJECT ROLES

In the discussion we assume that Person objects can possess Employee and Student roles. Figure 1 presents an example object store built in accordance with our database with dynamic roles :

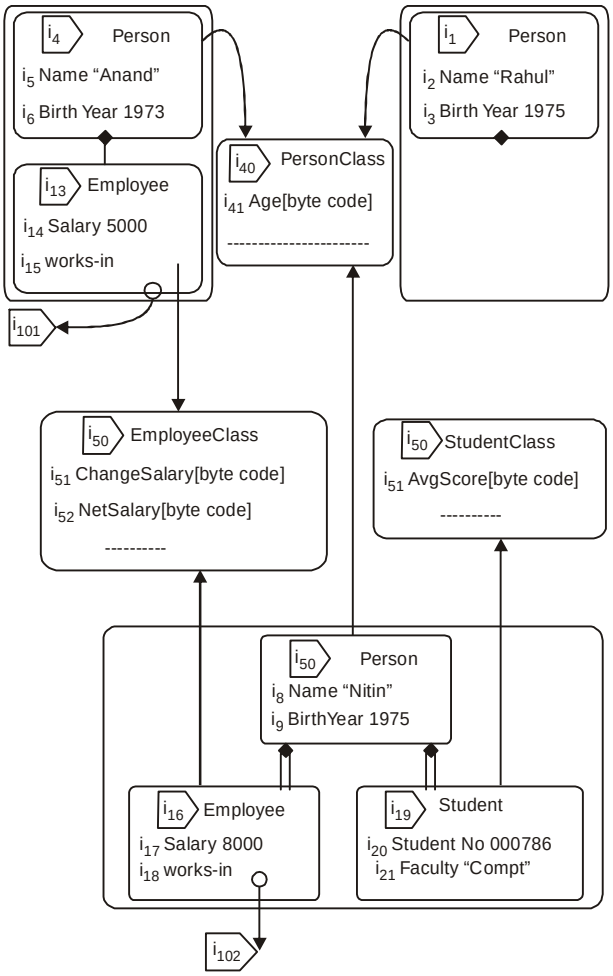

Fig.1. Example object database storage.

(*i*) Each store object is constructed of the following elements: an identifier, a name, and a value (which can a literal, a link, or a set of objects); *e.g.*, one of the objects is named Person, its identifier is i_1, and its value is two objects with identifiers i_2 and i_3.

(*ii*) There are three objects storing the invariant properties of Person, Employee, and Student objects; they are PersonClass, EmployeeClass, and StudentClass, respectively.

(*iii*) Each object has access to the invariant properties of its class. This is denoted as a thick black arrow; *e.g.*, the Student role is connected to the StudentClass object.

(*iv*) Roles dynamically inherit the properties of their owner-objects. This is denoted as a double-line with a diamond end; *e.g.*, Employee roles dynamically inherit from their Person owner.

A. Database stack

In programming languages a special data structure called a *Database Stack* is responsible for scope control and binding names. A new location of volatile objects (called *active record*) is pushed onto the database stack when a procedure is started, and the location is popped when the procedure is terminated. An active record for a procedure invocation contains volatile variables (objects) declared within this procedure, its actual parameters, its return address, and other data. Binding follows the "search from the top" rule. The last added location is the first one visited during the binding, and objects from some locations remain invisible for the binding (called *static scoping*).

OOD involves Database Stack – the general idea of the database stack semantics for object query languages is that some query operators (non-algebraic) act on Database Stack in a similar way as invocations of program procedure. For instance, in the query

Employee **where** Salary < 5000 **and** Age > 40 [-->] the part Salary < 5000 **and** Age > 40 is a query evaluated in a new query fired, which is determined by the currently processed Employee object. Thus, for the evaluation of this subquery, Database Stack is augmented with a new location containing information about the internal properties of the object. After the evaluation this location is popped.

Database Stack consists of locations, which are sets of binders. A binder is a pair (n, x), where n is an external name, and x is some value, in particular, a reference to an object. Such a pair is written as $n(x)$.

In general, binders serve name binding occurring in queries. For instance, if binder $n(x)$ is on Database Stack and we want to bind the name n, then the result of the binding is x. The "search from the top" rule means that when n is being bound, we are looking for the binder $n(x)$ that is closest to the stack's top. To cover bulk data structures of the store data, we assume that binding is multi-valued: if the relevant location contains more binders whose names are $n : n(x_1), n(x_2), n(x_3), ...$, then all of them form the result of the binding. In such a case binding n returns the collection $\{x_1, x_2, x_3, ...\}$.

B. Opening new locations on database stack

Consider a query including some name n :

Employee **where** ... n ... [==>]

In the classical OOD (*i.e.*, without dynamic object roles), while binding n, Database Stack has the following locations

going from the top : the internal query fired of the currently processed object; the environment of its class (binders to EmployeeClass's properties); the environment of the superclass (binders to PersonClass's properties).

The rules for opening new locations on Database Stack by a non-algebraic operator for the object oriented database with roles are a natural modification of the rules for the object oriented database without roles. The most important differences for the database with roles are the following : first, there are new locations for the properties of the roles (and possibly their owners); second, the database locations contain binders to root roles.

In the discussion below we consider a query q_1 θ q_2, where q is a non-algebraic operator, q_1 and q_2 are sub-queries.

(*a*) *The database without roles*

Let q_1 return the identifier of some object O. Let O be a member of $C1O$ class, which inherits statically from $C2O$, which in turn inherits from $C3O$, etc. Let O, $C1O$, $C2O$, $C3O$, ..., have identifiers i_O, i_{C1O}, i_{C2O}, i_{C3O}, ..., respectively (in OOD, classes, methods, etc, are objects too). Then q pushes onto the top of Database Stack the corresponding locations in the order shown in Fig.2 (nested is a function returning binders to the internal properties of the object, whose identifier is the argument of the function).

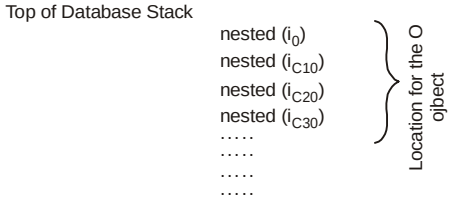

Fig.2. Locations pushed onto Database Stack by a non-algebraic operator in OOD without roles.

(*b*) *The database with roles*

Let q_1 return the identifier of an $R1$ role. Let $R1$ inherit dynamically from $R2$, which in turn inherits dynamically from $R3$, etc. Let Ri (i = 1, 2, ...) be a member of $C1Ri$ class, which inherits statically from $C2Ri$, which in turn inherits from $C3Ri$, etc. The corresponding identifiers are: i_{R1}, i_{R2}, i_{R3}, ..., i_{C1R1}, i_{C2R1}, i_{C3R1}, ..., i_{C1R2}, etc. In such a case θ pushes onto the top of Database Stack the corresponding locations in the order shown in Fig.3.

All the opened locations are removed after processing the $R1$ role. Other rules concerning opening and closing Database Stack locations by a non-algebraic operator remain unchanged.

For simplicity, Fig.2 and Fig.3 neglect encapsulation that subdivides properties into public and private. However, the rule for the object database with roles remains the same as for the classical model, that is, private properties of an object of a given class are available only to the methods that are stored within this class.

Fig.3. Locations pushed onto database stack by a non-algebraic operator in OOD with roles.

Fig.4 presents an example state of Database Stack while processing *Employee* roles for the store from Fig.1 in accordance with the Database Stack manipulation idea shown in Fig.3.

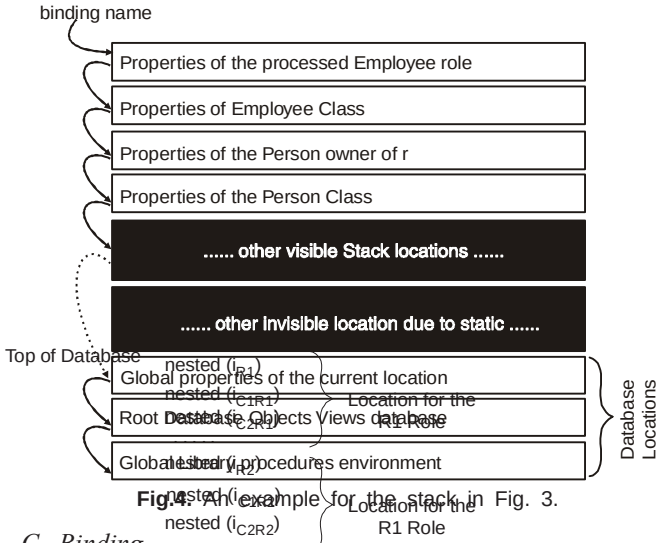

Fig.4. An example for the stack in Fig. 3.

C. Binding

Binding rules for the database with roles are exactly the same as for the classical model, see Fig.4 for binding name *n* in query [==>]. All roles' names can be bound in a database location of Database Stack. If the database with roles is used for programming of applications, then roles' names can also be bound in the location of the current user session and in locations containing local environments of procedures and methods. The rules for auxiliary naming (the **as** operator known from OQL) are the same as for the classical model.

In Fig.5 we present example steps of Database Stack during evaluation of query [-->] for the object store in Fig.1. Locations inessential for this example are not shown. The first Database Stack steps contains only the database location containing binders for owners and their roles, where the name *Employee* is bound (returning $\{i_{13}, i_{16}\}$). The second step presents the situation when **where** is processing the object whose identifier is i_{16}. The name *Salary* is bound at the top (returning i_{17}) and the name *Age* is bound in the *PersonClass* location, the fourth from the top (returning i_{41}). During execution of the method, the location of the

Employee role and the location of *EmployeeClass* are invisible due to static scoping. The final step, after executing the query, is the same as the beginning step.

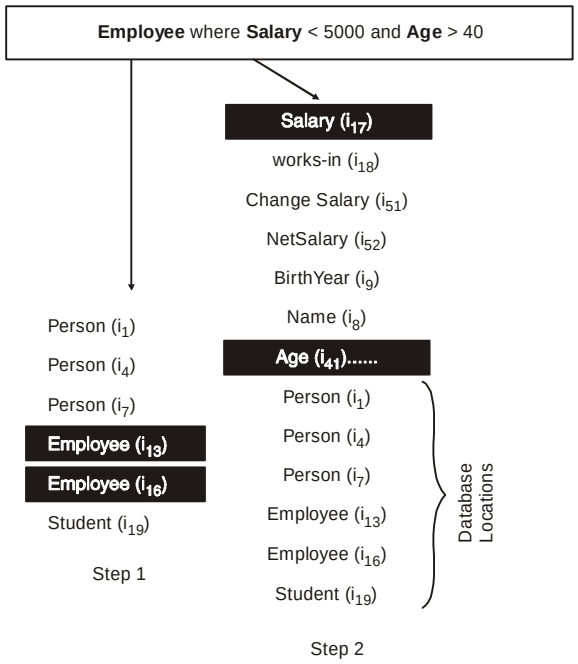

Fig.5. Steps of database stack during processing queries.

D. Polymorphism and overriding

The discussed above stack-based semantics supports polymorphism due to the fact that each role and its class are encapsulated. Thus the designer can use the same name for different methods stored within different classes.

Overriding is naturally supported by the scoping rules as well. In particular, it is possible to override a method defined for a role by a method defined for its sub-role. The overriding mechanism is extended : it is possible to override an attribute defined for a role by a method defined for its sub-role (and vice versa). Such a feature can be useful e.g. when in a specialized role one wants to replace an attribute with a virtual attribute.

E. Operators casting

The well-known technique of casting can be applied in OOD with roles where it enables one to make an explicit conversion between :

(*i*) a role and its owner-object,

(*ii*) different roles of an owner-object,

(*iii*) an owner-object and one of its roles.

Such a feature is necessary e.g. for the query "get all employees who are students". In contrast to the typical cast operators, our cast operators not only convert types, but they are also regular operators mapping a collection of identifiers into another collection of identifiers.

Syntactically, the operator is written as :

(name) query

Where name is the name of an owner-object or a role, and query returns identifiers of owner-objects or roles. If query returns owner-objects' identifiers, then the operator returns the identifiers of name roles within those objects. If query returns roles' identifiers, then the operator returns the identifiers of their owner-objects (for name objects) or the identifiers of other roles within the same objects (for name roles). If an object has no role named name, then the result is empty (null).

Through the operator we can express, for instance, the queries below.

Example 1 : "Get employees who are students."

(Employee) Student

Evaluation of the query *Student* returns a collection of the identifiers of *Student* roles. Then the (*Employee*) cast operator converts each of them into the identifier of an *Employee* role or into null, if a given object has a *Student* role, but has no *Employee* role. The result is a collection of the identifiers of *Employee* roles; nulls are ignored.

Example 2 : "Get persons named Sameer who are students."

(Person) (Student **where** Name = "Sameer")

Evaluation of the query Student **where** Name = "Sameer" returns a collection of the identifiers of Student roles in those objects for which the value of the Name attribute is "Sameer" (note that this value can be accessed in Student roles through dynamic inheritance from their Person owner-objects). Then the (Person) operator converts each of them into the identifier of the corresponding Person owner-object. The result is a collection of those identifiers.

Example 3 : "Suppose that students have a Scholarship attribute. For each person, get his/her name and income: the salary for employees, the scholarship for students and the sum of the salary and scholarship for working students. For a person, who is neither an employee nor a student, the income is 0."

(Person **as** p) . (p . Name, sum(0, ((Student) p) . Scholarship, ((Employee) p) . Salary))

In the above example, sum is an aggregate function similar to the corresponding SQL function.

Similarly we can introduce a Boolean operator testing presence of a given role within an object. Another operator can return the names of those roles that are currently present within an object. Such operators increase the generic programming ability.

III. OPTIMIZATION OF QUERY

Because the database with dynamic roles is based on the standard OOD, the general idea of static query optimization through static analysis, as presented *e.g.*, in [5, 7, 8], remains generally the same. To cover the concept

of roles, the database schema graph needs a new kind of nodes to store definitions of roles and a new kind of edges for the is_role_of relationship between roles.

Some modifications are needed for the query optimization techniques. The stack's sizes calculated in the method of independent subqueries [5].

In addition, optimization techniques, which have not been considered so far. For instance, the cast operators discussed in the previous section can lead to a situation when an auxiliary name cannot be eliminated but the query can still be rewritten to a more efficient form. The following illustrates the idea. Consider the query "get persons born after 1970 along with their companies" in OODQL with dynamic roles :

((Person **as** p) ⋈ ((Employee) p) . works-in . Company) **where** p.BirthYear > 1970

In this query the dependent join operator ⋈ joins each Person owner-object (named p in this query) with the Company object that the person works-in iff the person has an Employee role, which is determined by the cast operator (Employee). The resulting pairs <p(i_{Person}), $i_{Company}$> (where i_{Person} and $i_{Company}$ are references to Person and Company objects, correspondingly) are then filtered by **where**, which selects only those pairs, for which the person is born after 1970. According to the optimization rules presented in the papers referred to above, the selection predicate can be pushed before the ⋈ operator:

(((Person **as** p) **where** p.BirthYear > 1970) ⋈ ((Employee) p) . works_in . Company)

but the auxiliary name p cannot be removed, because it is used after the join in casting. Nevertheless, we can perform the selection before introducing p:

(((Person **where** BirthYear > 1970) **as** p) ⋈ ((Employee) p) . works_in . Company)

Such a case can be especially common when the optimization concerns queries involving views, that is, when such a selection is applied to a view invocation, which is then macro-substituted. The example shows an ability to apply query-rewriting techniques to queries addressing the object model with roles.

Although optimization techniques for the database need further development, we do not expect that this object oriented database implies totally new query optimization problems.

IV. SUMMARY

In the paper we have presented general assumptions concerning to the Object Oriented Database with Query Language, which incorporates the concept of dynamic object roles. The concept is a powerful object oriented database that makes it possible to express that e.g. an object during its lifetime can acquire and lose roles without changing its identity.

Our further research will continue to develop Architecture for Real Time Object Oriented Distributed Database

REFERENCES

[1] F. Steimann, "On the Representation of Roles in Object-Oriented and Conceptual Modeling", *Data & Knowledge Engineering*, Elsevier Science, **35**(1)**:** 83-106(2000).

[2] R.G.G. Cattel, D.K. Barry (Eds.), *Object Data Management Group: The Object Database Standard ODMG, Release 3.0*, Morgan Kaufmann Publishers, (2000).

[3] American National Standards Institute (ANSI) Database Committee (X3H2), Database Language SQL Part 2: Foundation (SQL/Foundation), *J. Melton (Ed.)*, Working Draft, (1999).

[4] S. Alagic, "The ODMG Object Model: Does it Make Sense?" *Proc. of OOPSLA*, SIGPLAN Notices, Atlanta, USA, **32**(10)**:** 253-270(1997).

[5] J. P³odzieñ, A. Kraken, "Object Query Optimization through Detecting Independent Subqueries", *Information Systems*, Elsevier Science, **25**(8)**:** 467-490(2000).

[6] K. Subieta, Y. Kambayashi, J. Leszczy³owski, "Procedures in Object-Oriented Query Languages", *Proc. of VLDB*, Zurich, Switzerland, 182-193(1995).

[7] J. P³odzieñ, K. Subieta, "Applying Low-Level Query Optimization Techniques by Rewriting", *Proc. of DEXA*, Springer LNCS 2113, Munich, Germany, 867-876(2001).

[8] J. P³odzieñ, K. Subieta, "Static Analysis of Queries as a Tool for Static Optimization", *Proc. of IDEAS*, *IEEE Computer Society*, Grenoble, France, 117-122(2001).