Extending a View Mechanism to Support Schema Evolution in Federated Database Systems

Zohra BELLAHSENE
LIRMM UMR 9928 CNRS - Montpellier II
161 rue ADA 34392 Montpellier Cedex 5, France
e-mail : bella@lirmm.fr

Abstract. This paper discusses the impact of autonomy requirement on schema design in tightly coupled federated database systems. In a federated database system, an important feature of a schema evolution strategy should be how the conceptual autonomy can be preserved. The conceptual autonomy requirement states that evolution of a local schema should not affect the remote schemas. In this paper we propose a view mechanism enhanced with import/export facilities to support schema evolution enforcing the autonomy requirement. More precisely, some schema evolution operations having ability to entail incompatibility of existing programs with regard to the new schema will not actually performed but simulated by creating specific views. Our approach is concerned with preserving both remote schema from schema changes arising on a local schema and maintaining compatibility for local and remote existing programs.

Keywords: Federated database systems, conceptual autonomy, view mechanism, schema evolution, virtual class.

1 Introduction

A federated database system (FDBS) is a collection of cooperating but autonomous component database system [19]. In federated database systems, distribution is due to the existence of multiple database systems (DBSs) before the federation is built. The three-level schema architecture describing the architecture of a centralized DBS has been extended to support the three dimensions of federated database system-distribution, heterogeneity, and autonomy [10]. The main schemas appearing in this architecture are : the local schema as the conceptual schema of local database, the component schema as the translation of the local schema into a common model, the federated schema as the integration of multiple exported schemas, forming the global conceptual schema and lastly the external schema as a special view of the global schema, customized for a group of federated users. For simplicity, we assume single global (i.e. federated) schema for the entire tightly coupled FDBS.

The schema changes arise for many reasons: changed real world domain, reusability and extendibility of classes or cooperation with other systems, etc. [22]. Two schema evolution approaches strategies have been proposed. The first strategy called *conversion or instances adaptation* restructures the affected instances to conform to the structure of their modified classes [2] and [9]. In an object-oriented database, the effects

of the schema changes can be particularly disruptive to the other users, because changes to the properties of a class must be propagated to the all subclasses of the class. The primary disadvantage of *conversion* approach is its lack of support for program compatibility since the former schema is discarded. The second approach, called *emulation,* is based on a class versioning rather than modifying the schema and converting the instances [11], [8]. Each class version evolution entails a new version of the class. This approach is more appropriate for evolution management notably it allows the continued support of old programs.

Recently some work has been done to show how schema evolution can be captured through views in object database systems [6], [21], [17] [5]. Schema evolution is supported by simulating each change operation by a specific view. In this paper, we discuss the extension of a view mechanism enhanced with import/export facilities to support schema evolution in distributed databases. The main concern of our approach is enforcing the conceptual autonomy requirement and maintaining compatibility for existing local and global application programs. When a schema change (entailing a risk of incompatibility) arises, a new virtual schema reflecting its changes, is assigned to the user, while the old one is maintained by the system, as long as other application programs continue to operate on it.

This paper is organized as follows. In section 2, we present the specific terms that we use in our proposition. Section 3 describes the functionalities provided by a view mechanism in federated database systems. A general update strategy is presented in section 4. How local schema can be updated is discussed in section 5. Related work is presented in section 6. Finally, section 7 contains concluding remarks.

2 Preliminaries

We assume that data model concepts such as database state, database schema, class, inheritance are familiar to the reader [12], etc. Our approach is independent from the Database System (DBS). However, the environment of our study is the O2 DBS. Thus, the examples described in this paper are based on the O2 data model [13].

2.1 Naming

The O2 data model does not provide class extension at the logical level. However, the class extension can be simulated by creating a named collection of objects which is a root of persistency. Names are entry points in the database. From these entry points, other objects can be reached through navigation.

2.2 The Sample Schema

Figure 1 describes an example of two object-oriented database schemata which will be used in this paper for defining and updating local schemas. This schema is inspired

from the one presented in [22]. The first schema named SalesDB stores information about articles and customers. While the second schema, named ProdDB, stores information about materials and their suppliers. The schema description with the object data description language (ODDL) [16] is as follows:

```
Create schema SalesDB;
Class Article
type tuple (anum: integer, price: float,
            bought_by: list   (Customer))
end;

Class Customers
type tuple (name: string,
            addr: string,
            bought: set (Article))
end;

Create schema ProdDB;
Class Material
type tuple (matnum: integer, weight: float,
            type: string, suppl_by: Supplier)
end;

Class Supplier
type tuple(name: string,
           addr: string,
           supplies: set (Material))
end;

name The_Articles: set(Articles); /* this is a root of persistency and a database
                                     entry point */
name The_Materials: set(Material); /* this is another root of persistency */
```

Fig.1. Example of local schemata.

2.3 Virtual Class

A virtual class VC_i is defined by a couple (P_v, Q_v)
where Q_v is a query allowing to compute the extent of VC_i and P_v, is a set of specific properties (non-derived attributes and methods).
This definition shows that a virtual class may own both properties derived from the root class (trough the query) and specific properties. This feature allows to simulate schema extension and schema modification operations.

Example 2.3. We imagine that the database administrator (DBA) wants to provide users with a view, providing a special treatment for computer materials. The query defining such a view is expressed as follows:

```
Virtual class Computers
query :  Select x
         From x in The_Materials
         where x->type = "computer";
end;
```

3 Views in Federated Databases

A view mechanism can be used at several levels for information design in a federated database. First, views can be defined to support customization and access control on local databases by specifying the exported part of data. That is only visible properties are included in a virtual class which will be exported.

Example 3. Suppose that the local Database Administrator (i.e. DBA) of SalesDB schema, wants to share only a part of its schema, for instance, some properties (anum, price) of the class Article. While the ProDB local administrator wants to export only a part of the class Material (type, weight, supplier's name). He (she) can do so by creating virtual classes including only those properties allowed to be exported. First, he (she) has to specify virtual classes and then exports it with our view definition language [5] as follows:

```
Create virtual class Public_Articles
query :  Select x
         From x in The_Articles; /* persistent set of the class Article instances */
         Hide bought_by;
end;

Create virtual class Public_Materials
query :  Select x
         From x in The_Materials;
         Hide matnum;
end;

Export virtual class Public_Articles;
Export virtual class Public_Materials;
```

Within the federated level, in the context of a loosely coupled FDBS, views can be used to build a uniform, virtual interface over multiple databases [15], [7]. Finally, a view mechanism can be used to specify a subset of information in the federation that is relevant to the users of external schemas. Furthermore, views may provide access control to the data managed by the federation.

4 A Schema Update Strategy for a Federated Database

4.1 The General Strategy

We classified the set of schema modifications in two categories according their impact on the existing application programs. The former concerns the extending operations which augment the schema content. The second category includes the update operations corresponding to the delete and modify operations. The main difference between the two categories lies in the fact that the extending operations do not entail a

risk of incompatibility of existing application programs with regard to the new schema. The operations belonging to the second category will not actually performed but simulated by creating specific views. This kind of operations can entail structural or semantic conflicts with regard to the global schema. The problem of structural and semantic conflicts are out the scope of this paper, see related work in [3][20][18].

4.2 Schema Consistency

Any schema change must ensure the schema consistency. In the context of an object database system, structural consistency is provided by using a set of invariants that define the consistency requirements of the class hierarchy. In our approach, two kinds of invariants have to be taken into account. The first kind of invariants concerning the class hierarchy is inspired from ORION ODBS [2]. We propose a new invariant enforcing the conceptual autonomy property. These invariants providing the basis of the semantics specification of schema updates propagation, have to ensure both the preservation of existing application programs and the schema consistency.

Conceptual Autonomy Principle
A federated database system provides *conceptual autonomy* if the specification and the semantics of the remote local schemas are not affected by any local schema change.

5 Local schema evolution

The schema evolution strategy we propose is independent from the data model. However, in this paper, we will illustrate it through an object-oriented schema evolution. Our virtual schema update language includes the basic schema changes [23] and more sophisticated modifications such as nest and unnest operations, affecting the aggregation links [5]. However, due to space limitations these schema changes will not be described here.

5.1 Add an Attribute to a Class

The semantics of the add attribute operation is to augment the type of a class. A relevant issue concerns the visibility of the added property for the remote databases and the existing derived classes if the specified class is a root derivation class The visibility of the added attribute should be specified in the Add command. The syntactic form of the add attribute command is :
Add Attribute <attribute specification><visibility> to <class name>;

The value of the visibility should be *private* or *public*. The default value is private.
Here is an algorithm for add an attribute. Let C_{exp} be a set of exported classes.
Add_Attribute(A: T, visibility, C) /* add an attribute A with a type T to C*/
If A exists then reject the operation
Else Add Attribute A: T to class C;

```
If (C ε C_exp) Then
        If (visibility = public) /* A will be added to the global schema */
                Add Attribute A: T to the global schema;
If (V = (derivation root class)) and (visibility = private) then
For i=1, k          /* k is the number of derived virtual classes from V */
        Hide Attribute A in view V_i
end Else;
End Add_attribute;
```

5.2 Delete a Property

Before propagating a delete operation, we need to know if the related property is currently used by application programs in the local database or in the remote databases. We define a function named "fused(p)" which returns a Boolean (true if p is currently used and false otherwise). This kind of function is also provided by programming language, for instance Smalltalk. Figure 2 summarizes the different cases arising when a property is deleted in a local schema.

position of p	p is used	Action to perform
exported	globally used	The deletion will not actually performed but simulated since p is used in remote programs
exported	locally used	The deletion will not actually performed but simulated since p is used in local programs.
exported	not used anywhere	Delete p in the local and the global schemata.
imported	locally not used	Delete p in the local schema only.
imported	globally not used	The deletion will not actually performed but simulated since p is imported.
imported	locally used	The deletion will not actually performed but simulated since p is used in local programs.
imported	globally used	The deletion will not actually performed but simulated since p is used in remote programs.

Fig. 2. Deletion strategy in a federated database.

The general syntactic form of a delete operation is :
Delete Attribute <property name> in < class name>;
The semantics of the delete operation is to remove the specified attribute from the type of the class C in the current local schema and affects the global schema only if the related attribute is not currently referenced by an application program in the remote databases. Let C_{exp} be a set of exported classes and let C_{imp} be a set of imported

classes. The algorithm dealing with the delete operation is described by the following procedure:

Delete_property (p, C) /* delete a property p in a class C */
If not(fused(p)) and C<>derivation root
  Then /* p is not used in existing programs and C is not a derivation root */
    If ($C \varepsilon C_{imp}$) Delete p in class C; /* p is deleted in the local schema */
    If ($C \varepsilon C_{exp}$) Then delete p in class C in the global schema
  end then
Else /* create a new version of C in which p will be hidden */
  create virtual class C' from C
  query : select x from x in C_extent;
  Hide p ;
  end;
  Rename C' as C;
End Delete_property;

Example 5.2. We imagine that the local DBA wants to delete the attribute weight in the class Material. The deletion operation is expressed as follows:
Delete Attribute weight in class Material;

Now suppose this attribute is referenced by an existing program then the deletion of this attribute will be not actually performed but simulated by creating a virtual class as follows:
create virtual class Material_V1 from Material
    query : select x from x in The_Materials;
    Hide weight;
    end;
Rename Material_V1 as Material;

5.3 Modify a Property

The semantics of this operation consists in modifying the type of an attribute or the code of a method. Adopted strategy is very closed to the strategy of the delete operation. Here is an algorithm to modify an attribute.

Modify_Attribute (A: T, C) /* Modify A in class C with a new type T */
If not(fused(p)) and C <> derivation root
    Then Modify A in class C in local schema;
      If ($C \varepsilon C_{exp}$) Then Modify A in class C in the global schema;
      end then
  Else /* the modification is not actually performed but simulated */
    create virtual class C' from C
    query : select x

```
        from x in C_extent;
        Hide A;
        Add Attribute A : T /* A has the new type T */
        end;
        Rename C' as C;
End Modify_Attribute;
```

6 Related Work

The original idea of using the view mechanism to support schema evolution has been published in [6]. Schema evolution is supported by using a view definition language allowing capacity-augmenting views. However no solution was provided to managing the specific attributes introduced by capacity-augmenting views.
In [21], the simulation process is enhanced and included in an external schema definition. An external schema is defined as a subset of base classes and a set of views corresponding to the schema modifications.
More recently, the Multi-view system is enhanced with schema evolution support [17]. However, the proposed view definition language does not allow the virtual classes derived from several base classes. On contrary, our method still works for composite virtual classes.
Another work [22] attempts to define a variable architecture of a federated database based on three schema transformation processors : (i) schema extension, (ii) schema filtering and (iii) schema composition.
An approach based on schema versioning and instance multifaceting to support program compatibility in distributed object database systems has been proposed in [8]. The originality of this work lies in pointing out similarities between schema evolution and heterogeneous schema integration. However, the taxonomy of proposed evolution operations do not include operations including more than one class.

7 Conclusion

This paper describes a view mechanism enhanced with import/export facilities to support schema evolution enforcing the conceptual autonomy requirement in tightly coupled federated database systems. More precisely, our approach consists in maintaining virtual versions of local schema so that local and global applications and remote schemas are unaffected by local schema changes. However, a version of the original schema is created only if the schema evolution operation entails a risk of incoherence of existing views and remote schemas or application programs. Thus, this strategy has as primarily advantage to avoiding the proliferation of view versions occurring in the other view based approaches [6], [17].
The feasibility of our approach is confirmed by an experimentation on a centralized database. A small prototype named SETV has been developed on top of the O2 DBS [14], [5].

References

1. ABITEBOUL S., BONNER A. ,"Objects and Views", in Proc. ACM SIGMOD, Conference on Management of Data, pp 238-247, Denver, Colorado, May, 1991.

2. BANERJEE J., KIM W., KIM K.J., KORTH H., "Semantics and Implementation of Schema Evolution in Object-oriented databases", in Proc. ACM SIGMOD Conference on Management of Data, San Francisco, 1987.

3. BATINI C., LENZIRIN M. et NAVATHE S., "A Comparative Analysis of Methodologies for Database Schema Integration", ACM Computer Surveys 18(4), pp. 323-364, December, 1986.

4. BELLAHSENE Z., PONCELET P.,TEISSEIRE M., " Views for Information Design without Reorganization", in Proc. of CAiSE'96, International Conference on Advanced Information System Engineering, Crete 20-24 May, Lecture Notes in Computer Science, Springer-Verlag, May, 1996.

5. BELLAHSENE Z., "Views for Schema Evolution in Object-oriented Database Systems", British National Conference on Database BNCOD-14", Lectures Notes in Computer Science, Springer Verlag, Edinburgh, July, 1996.

6. BERTINO E., "A View Mechanism for Object-oriented Database", 3rd International Conference on Extending Database Technology, March 23-24, Vienna (Austria),1992.

7. BERTINO E, "Integration of Heterogeneous data repositories by using object-oriented views", in Proc. of the first International Workshop on Interoperability in Multidatabase Systems, April, 1991.

8. CLAMEN S.M., "Schema Evolution and Integration", Journal of Distributed and parallel Databases 2 (1994), pp. 101-126.

9. FERRANDINA F., MEYER T., ZICARI R., G. FERRAN G., MADEC J., "Schema and Database Evolution in the O2 Object Database System", in Proc. of VLDB Conference, Zurich, Switzerland, september, 1995.

10. HEIMBIGNER D., D. McLEOD, "A Federated Architecture for Information Management", ACM Transactions on Office Information Systems, 3(3), pp. 253-278, July, 1985.

11. KIM W., CHOU H.T., "Versions of Schema for Object-Oriented Databases", in Proc. of the 14th VLDB Conf., L.A., California, pp 148-159, 1988.

12. KIM W., "Introduction to Object-Oriented Databases", MIT Press, Cambridge, Massachusetts, London, England, 1990.

13. LECLUSE C., RICHARD P., VELEZ F., "O_2, An Object Oriented Data Model", in Proc of the ACM SIGMOD Conference on Management of Data, Chicago, June, 1988.

14. LUCATO G., "Evolution de schema au travers des vues", Rapport de DEA, University of Montpellier II, June 1995.

15. MOTRO A., "Superviews: Virtual Integration of Multiple Databases", IEEE Trans. on Software Engineering, Vol. SE-13, N°7, July 1987, PP. 785-798.

16. The O2 User Manual, O2Technology, 1994.

17. RA Y.G., E. A., "A transparent object-oriented Schema Changes Approach Using View Evolution", IEEE Int. Conf. on Data Engineering, Taipei, Taiwan, 1995.

18. SIEGEL M., MADNICK S., "A Metadata Approach to Resolving Semantic Conflicts. In Proceedings of the 17th VLDB Conference, Barcelona, September, 1991.

19. SHETH A.P, LARSON J.A., "Federated Databases Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computer Surveys, 22(3):183-236, September, 1990.

20. SPACCAPIETRA S., PARENT C., DUPONT Y., " View integration: A Step Forward in Solving Structural Conflicts", IEEE Transaction on Knowledge and Data Engineering, October, 1992.

21. TRESCH M., SCHOLL M.H., "Schema Transformation without Database Reorganization", in SIGMOD Record, 22(1), March 1993.

22. TRESCH M., SCHOLL M. H., " Schema Transformation Processors for Federated Object Bases", in Proc. 3rd International Symposium on Database Systems for Advanced Applications (DASFAA), Daejon, Korea, April 1993.

23. ZICARI R., "A framework for O2 Schema updates", in Proc. of 7th IEEE International Conference on Data Engineering, PP. 146-182, April 1991.