
SANG-WON LEE, Seoul National University
HYOUNG-JOO KIM, Seoul National University

Introduction

The functionality of schema evolution is one of the important differences between object-oriented database management systems (OODBMS) and relational database management systems (RDBMS). Object-oriented data models emerged in the mid 1980s and since then many approaches to schema evolution have been proposed (Banerjee, Kim, Kim, & Korth, 1987; Penney & Stein, 1987; Zicari & Ferrandina, 1997). This is because the applications of OODBMSs, such as CAD/CAM, CASE, and multimedia, require dynamic schema changes and flexible schema management. Currently, several commercial OODBMSs, such as GemStone (Penney & Stein, 1987), O2 (Zicari & Ferrandina, 1997), ObjectStore (Object Design, Inc., 1994), and Objectivity (Objectivity, Inc., 1998) support various schema update primitives and provide online schema evolution mechanisms.

Under these systems, however, only a single schema can exist at any time; if a schema evolution operation completes, the old schema is no longer maintained. This schema modification mechanism has several drawbacks (Kim, 1991). First, schema updates may invalidate programs written against old schema. Second, because all the users share a single schema, schema updates by one user may change the views of all the other users. In summary, the current schema evolution approach raises the problem of lack of logical data independence in OODBMSs. One of the most required characteristics of a good data model is logical data independence; the ability to modify the database schema without causing application programs to be rewritten. In a practical aspect, the property of logical data independence relieves the database users including the database administrator of much of the database application maintenance nightmare. In fact, much of the success of relational data model is attributable to its support for logical data independence via the concept of view. Considering the requirements for dynamic schema changes of OODBMS applications, we think OODBMSs also should be empowered with logical data independence at a level comparable to that of relational data model.

Schema version mechanisms were introduced to enhance the logical data independence of OODBMSs, and many researchers have stressed their importance since a characteristic of design applications should cope with frequent schema changes (Kim & Chou, 1988; Lautemann, 1996). Recently, the necessity for schema versions has been
rejuvenated in several new OODB applications including Repositories (Bernstein, 1998; Silberschatz, Stonebraker, & Ullman, 1995), Portable Common Tool Environment (PCTE) (Loomis, 1992), and the World Wide Web (WWW) (Atwood, 1996; Yang & Kaiser, 1996), all of which may use an OODBMS as an integrator of data. Data repositories are expected to be one of the important new uses of DBMS technology (Silberschatz et al., 1995). Though much work has been done to provide schema version mechanisms for object-oriented databases (OODB) (Bertino, 1992; Kim & Chou, 1988; Monk & Sommerville, 1993; Ra & Rundensteiner, 1995), this field has not reached a satisfactory status. Traditional schema version approaches have three outstanding problems: (1) storage overhead for redundant objects (Kim & Chou, 1988; Ra & Rundensteiner, 1995), (2) limited schema update capability (Bertino, 1992), and (3) complexity for managing consistent schema versions (Monk & Sommerville, 1993). The RiBS model overcomes all these problems, enabling efficient and flexible schema version management. Our main contribution can be summarized as follows.

1. The RiBS model, the first view-based schema version approach, proposes a framework for mapping each schema evolution over schema version(s) to schema evolution(s) over base schema. This framework includes: (1) structural representation of schema versions and base schema, (2) a set of invariants, and (3) a set of schema evolution operations and its semantics.

2. The RiBS model solves the limited schema update capability of a traditional view-based schema evolutions and, unlike class versioning approach, allows for managing schema version more easily. That is, with the RiBS model, a user can impose the well-known schema evolution operations (Banerjee et al., 1987) over schema versions without any restrictions as in Bertino (1992), and their effects are reflected in database without user interventions as in Monk & Sommerville (1993).

3. Finally, the RiBS model does not incur any storage overhead; every physical object resides only in RiBS, and every object accessed from schema versions are view objects of its corresponding physical object.

The remainder of this paper is organized as follows. Next, a brief overview of the RiBS model using an illustrative example is given. Then, the object model assumed in this paper is described. This is followed by a detailed description of each component of the RiBS model, which includes the 1) structural part, 2) a set of invariants, and 3) schema evolution operations and their semantics, respectively. A few issues about the implementation of the RiBS mode are given and the work is compared to related work. Finally, the paper concludes with a summary and an outline of future work.

Preliminary

In this section, we illustrate some basic ideas of the RiBS model with an example. For brevity, we assume the following informal description of a schema in an OODB: A schema in a database consists of classes and attributes. A class in the class hierarchy through “is-a” (ISA) relationships between them. Each class, in turn, consists of properties including both attributes and methods. To every class is attached a collection of objects, extent. Each instance object belongs to the extent of a single class, and is referred to as a direct instance of the class.

Rich Base Schema and Schema Versions

Before proceeding with the example, we introduce the concept of “Rich Base Schema,” and discuss how it can be exploited in supporting schema versions. We say that a schema S1, is richer in schema information than the other schema S2 if all the following conditions hold.

1. Every class in S2 has a corresponding class in S1.
2. Every property in S2 has a corresponding property in S1.
3. Every direct ISA relationship in S2 has a corresponding (direct or indirect) ISA relationship in S1.

If S1 is richer than S2, it means intuitively that S1 has more schema information than S2. This, in turn, means that S2 can be specified as a view of S1. This concept of rich schema can be re-stated in terms of relative information capacity (Hull, 1984); S1 dominates (or subsumes) S2. Our model is based on this concept of rich schema. A physical base schema, RiBS (Rich Base Schema), which is richer in schema information than any schema version, is maintained, and each schema version is represented as a view over RiBS. In addition, when a schema update is imposed on a schema version, RiBS is, if necessary, automatically augmented so as to be richer than the modified schema version in addition to all other ones. In summary, a schema version is an updatable class hierarchival view over RiBS, in the sense that schema evolution operations can be directly imposed on the view.

In our model, schema versions are strictly separated from RiBS. This separation prevents several problems arising when the schema information of schema versions is mingled with that of RiBS. Some previous works on views in OODB (Abiteboul & Bonner, 1991; Bertino, 1992) put normal classes and derived views together in a class hierarchy. However, this approach has several disadvantages (Kim, 1995). First, it is difficult to understand the complicated class hierarchy. Next, the extents of classes may overlap. Finally, it is difficult and, in certain cases impossible, to decide where to locate the view class in the class hierarchy.

An Intuitive Example

Now let us consider the example in Figure 1, where two
schema versions SV1 and SV2 are represented as views over RiBS. Schema information in RiBS is rich enough to contain all the classes and properties for either SV1 or SV2. The base class corresponding to a class version in a schema version is called the direct base class of the class version. For example, Person in RiBS is the direct base class of class version Person in SV1. Every instance object in the direct base class becomes an element of the logical extent of the class version. In Figure 1, both SV1 and SV2 share instance objects, Ulman and Korth, in RiBS. Under a specific schema version, an object of a class version is derived from an object of the corresponding base class.

In Figure 1, we assume that after SV1 is created (we call it the root schema version) and SV2 is derived from it, SV2 undergoes the following three schema updates: (1) rename class version Person as Member, (2) drop class version Professor, and (3) add attribute version phone# to class version Member. We now describe how each schema update affects SV2 and/or RiBS. The first operation renames the class version Person as Member only within SV2, without affecting RiBS or SV1, since each schema version maintains names for its own class versions and their property versions, independently of base classes and properties in RiBS. The second operation just drops the schema information of class version Professor from SV2, without affecting RiBS or SV1. However, this operation raises a subtle semantic issue: that is, the effect of dropping class version Professor on its logical extent. For this, the RiBS model chooses the semantics to migrate all the (logical) direct instances of Professor to the extent of a superclass. Thus, class version Member in schema version SV2 has base classes Person and Professor in RiBS as its extent base classes. The last operation, adding attribute phone# to class version Member, is different from the above two operations as it requires changes in RiBS as well as in SV2: a corresponding attribute should be added to the direct base class of class version Member, that is, base class Person in RiBS.

**Object Model**

The object model assumed in this paper is defined, which is common to RiBS and schema versions. A class $C$ in a database defines the properties of objects. Each class maintains its extent. A property represents either an attribute or a method. Each class may have more than one superclass. The set of direct superclasses of a class $C$ is denoted as $P(C)$. All the properties of the superclass(es) are inherited into the subclass. The newly defined local properties of a class $C$, denoted as $LP(C)$, together with the inherited ones, denoted as $IP(C)$, constitute the interface of the class, $I(C)$. For an inherited property $p$ of a class, there exists an origin property, denoted as $Org(p)$, from which $p$ is inherited. The notations for the object model are summarized in Table 1. The transitive closure of $P(C)$, namely the set of all the direct or indirect superclasses of class $C$, is denoted as $P^*(C)$.

Table 2 summarizes the inheritance semantics of the object model. As Taivalsarri (1996) points out (p. 432), although much research has been focussed on inheritance, researchers rarely agree on its meaning and usage. Even in ODMG-93 (Cattell, 1997), de facto object database standard from the Object Database Management Group (ODMG), no clear semantics are given for inheritance, in particular multiple inheritance. Thus, we need to develop these axioms to clarify the inheritance semantics in RiBS model.

Through axioms 1 to 3, we force a schema to be Direct Acyclic Graph (DAG). Axiom 4 means that the interface of a class consists of inherited properties and locally defined properties. Axiom 5 states that the inherited properties of a class $C$ are the unions of the interfaces of all the superclasses.

### Table 1: Notations for object model

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>Class</td>
</tr>
<tr>
<td>$S$</td>
<td>RiBS or schema version</td>
</tr>
<tr>
<td>$\tau(S)$</td>
<td>Set of all classes of schema $S$</td>
</tr>
<tr>
<td>$P$</td>
<td>Property</td>
</tr>
<tr>
<td>$P(C)$</td>
<td>Set of direct parents of $C$</td>
</tr>
<tr>
<td>$P^*(C)$</td>
<td>Set of all parents of $C$</td>
</tr>
<tr>
<td>$ISA(C_i,C_j)$</td>
<td>$C_j$ is a direct subclass of $C_i$</td>
</tr>
<tr>
<td>$ISA(S)$</td>
<td>All direct or indirect ISA relationships within schema $S$</td>
</tr>
<tr>
<td>$I(C)$</td>
<td>Interface of class $C$ (that is, the set of properties)</td>
</tr>
<tr>
<td>$IP(C)$</td>
<td>Inherited properties of $C$</td>
</tr>
<tr>
<td>$LP(C)$</td>
<td>Locally defined properties of $C$</td>
</tr>
<tr>
<td>$E(C)$</td>
<td>Extent of class $C$ (set of direct instances)</td>
</tr>
<tr>
<td>$Org(p)$</td>
<td>Original property of $p$</td>
</tr>
</tbody>
</table>
Table 2: Axioms for Inheritance

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Axiom of Closure</td>
<td>( \forall C \tau, P(C) \varphi t )</td>
</tr>
<tr>
<td>(2) Axiom of Acyclicity</td>
<td>( \forall C \tau, C \neq P(C) \neq t )</td>
</tr>
<tr>
<td>(3) Axiom of Rootedness</td>
<td>( P(Object) = { } )</td>
</tr>
<tr>
<td>(4) Axiom of Interface</td>
<td>( I(C) = IP(C) = LP(C) )</td>
</tr>
<tr>
<td>(5) Axiom of Property Inheritance</td>
<td>( IP(C) = \pi_C P(C) )</td>
</tr>
<tr>
<td>(6) Axiom of Superclasses</td>
<td>( P'(C) = \pi_C P'(C') = P(C) )</td>
</tr>
</tbody>
</table>

of \( C \). Axiom 6 means that \( P'(C) \) is the transitive closure of \( P \) relationships of class \( C \). According to these axioms, a class, of which two or more superclasses share a common superclass, inherits properties from the common superclass only once (like virtual inheritance in C++). Name conflicts between two or more properties of different superclasses, or between inherited and locally defined properties are allowed and users are responsible for designating a specific property (also similar to C++).

Structure

The structural component of the RiBS model has a three-level architecture: (1) the object base, (2) the rich base schema (RiBS), and (3) the schema versions. Every object physically resides in the extensional object base. RiBS accumulates all the necessary schema information ever defined in any schema version. Each schema version is in the form of a class hierarchy view over RiBS. Users are concerned only with the schema versions in the uppermost layer. Direct schema updates on schema versions are allowed, and their effects are, if necessary, automatically propagated down to RiBS. We give descriptions of all the structural components of the RiBS model.

Definition 1 (Base Schema, RiBS). In a database, there exists a single (Rich) Base Schema, called RiBS, which describes the structures of objects physically stored in the database. RiBS includes a set of base classes, and inheritance relationships between them constitute the class hierarchy of RiBS. The structure of each object stored in an object base conforms to the definition of the base class in RiBS to which the object belongs. To each base class of RiBS an extent is attached, which is the set of all the direct instance objects of the class.

Definition 2 (Schema Version, SV). A schema version \( SV \) is a logical class hierarchy view over RiBS, which represents either a snapshot of the ever-evolving database schema at a certain point of time, or a customized view for a particular user. A schema version \( SV \) is a class hierarchy view because it also has its class hierarchy. It is also a logical view in that all the objects visible from a schema version are derived from the objects stored in RiBS.

Definition 3 (Current Schema Version, CSV). We call a specific schema version, under which an application program/user accesses the database at a certain point of time, the current schema version (CSV). With the RiBS model, a user should designate CSV before (s)he accesses the database. A user can do all the normal database operations against the CSV. Moreover, a user can change the schema structure of CSV; that is, the schema can evolve.

In the RiBS model, the execution of a schema evolution operation, however, does not imply derivation of a new schema version. Instead, we provide an operation for users to explicitly derive a new schema version from existing one(s): the former is called the "child schema version" and the latter "parent schema version(s)". Derived-from relationships between schema versions constitute Schema Version Derivation Graph, defined as follows:

Definition 4 (Schema Version Derivation Graph, SVDG). A Schema Version Derivation Graph is a Directed Acyclic Graph, where each node represents a schema version and each directed edge between nodes represents a "derived-from" relationship. When a database is initialized at its creation time, the "root schema version" is created, in addition to the initial RiBS. The root schema version is the root of SVDG.

Definition 5 (Class Version, \( CV \), and Direct Base Class, \( B(CV) \)). A class version \( CV \) of a particular schema version represents a facet of a base class in RiBS, which needs to be modeled within the schema version. We call the base class the direct base class of \( CV \), and formally denote it as \( B(CV) \).

For each class version \( CV \), there is one and only one direct base class \( B(CV) \) in RiBS. However, the converse is not true; that is, a base class in RiBS may not be explicitly modeled in a schema version \( SV \), so it might have no corresponding class version in \( SV \). With respect to schema information capacity, \( B(CV) \) is a superset of \( CV \); \( B(CV) \) has all the schema information required in \( CV \). When the effects of a schema update against \( CV \) need to propagate to RiBS, the schema change in RiBS starts from \( B(CV) \).

In the RiBS model, users are concerned only with schema versions. Hence, each class version, like normal classes, is expected to have its own extent. For this, we maintain extent base classes for each class version.

Definition 6 (Extent Base Classes, \( B'(CV) \)). The Extent base classes of a class version \( CV \), \( B'(CV) \), are a set of base classes in RiBS. The union of the extents of these base classes comprises the logical extent of \( CV \). From these extent base classes, the logical extent of a class version \( CV \) is derived as follows: \( E(CV) = \pi_C B'(CV) E(C') \)

Definition 7 (Property Version, \( PV \), and Direct Base Property, \( B(PV) \)). A property version \( PV \) of a class version...
CV represents either an attribute or a method. The direct base property of a PV in a class version CV, denoted as B(PV), is a corresponding property in B(CV). Every PV has its direct base property.

The usage of direct base property is as follows. When a logical object is accessed in CSV, its value is derived from the corresponding physical object. In this process, the value of each PV is derived from that of the B(PV) of the physical object. The concept of extent base classes, however, complicates this process. If the corresponding physical object of an object being accessed under CSV is an instance of a base class that is not a direct base class of any class version of CSV, how do we derive the value of each PV from the physical object? In the RiBS model, like Orion (Banerjee et al., 1987), a subclass inherits all properties from its superclass(es). Thus, when deriving the value of each PV, the value of the property that has B(PV) as its origin property is used.

Invariants

As a second component of the RiBS model, we introduce a set of invariants that should be satisfied by the structure. This set of invariants plays a critical role in defining the semantics of schema evolution operations for schema versions. Invariants in the RiBS model can be classified generally into three categories: RiBS invariants, schema version invariants, and invariants between RiBS and schema version. In this section, we describe the last two categories. For the first category, we assume the well-known invariants for schema evolutions from Banerjee et al. (1987), Penney & Stein (1988), and Zicari & Ferrandina (1997).

For schema versions, we identify two new invariants, “no phantom reference” and “no multiple classification”, both of which are related closely to the schema evolution operation class drop. Incorrectly defined semantics for this operation might result in some anomalies.

Invariant 1 (No Phantom Reference). The value of an attribute of an object may be a reference to a phantom object, which is not a direct instance of any class in the schema version. We refer to this kind of reference as a phantom reference. This is in contrast to a dangling reference, which is a reference to non-existing object. There should be no phantom references within a schema version SV; that is, within a schema version SV, for each object O referenced by another object, there should exist a class version CV, where O Ext(CV).

Invariant 2 (No Multiple Classification). This invariant restricts each logical instance object in a schema version to be a direct instance of only one class version. In other words, logical extents of each class version in a schema version should be disjoint to each other, which can be formalized as follows: for every class version CV and CV' in a schema version SV, where i \ j, Ext(CV) \ Ext(CV') = \emptyset

The following two invariants should hold between each schema version SV and RiBS.

Invariant 3. For each class version CV (and property version PV) in a schema version, there should exist a corresponding B(CV) (and B(PV)) in RiBS.

Invariant 4. Within a schema version, for each base class C in RiBS, there should exist a class version CV, such that C \ B+(CV). This invariant means that the union of the extent base classes of all class versions in a schema version should be equal to the set of base classes in RiBS, that is, \CV C \ Ext(CV) \ τ \Ext(CV) = \τ \C

Operations

We give a set of operations for managing schema versions, which is the last component of the RiBS model. These operations are classified into two groups: one group is concerned with SVDG manipulation, and the other includes schema evolution operations against schema versions. Schema evolution primitives available in RiBS model are from Orion data model (Banerjee et al., 1987). We make a new taxonomy for the eight fundamental schema change operations in Orion, depending on their impacts on RiBS and other schema versions. These eight fundamental operations are complete in that all the schema changes can be achieved by combining these operations (Kim, 1991).

• Operations for SVDG manipulations
  1. Derive sv-name from parent-list
  2. Delete sv-name
  3. Set current schema version to sv-name

• Operations for schema evolution
  1. Operations which have no impact on RiBS
     (a) Change the name of a class version C
     (b) Drop an existing class version C
     (c) Drop an existing property version v from a class version C
     (d) Drop an edge to remove a class version S as a superclass of another class version C
     (e) Change the ordering of superclasses of a class version C
  2. Operations which have impacts on RiBS
     (a) Add an edge to make a class version S a superclass of class version C
     (b) Add a new property version v to a class version C
  3. Operations which have impacts both on RiBS and on other schema versions
     (a) Create a new class version C

Operations for SVDG manipulations

Derive sv-name from parent-list. This operation derives a new schema version sv-name from existing one(s) in parent-list. When a schema version is derived from a single parent, this operation can be easily implemented; that is, the
schema information of the parent is simply copied into the child. In the case of multiple parents, however, the parents with different schema should be merged into a new consistent one. We call this process “schema-version-merging.” We identified several conflicts in schema-version-merging, and devised a semi-automatic algorithm resolving them (Lee & Kim, 1997).

**Delete sv-name.** When a schema version is no longer needed, this operation removes it from $SVDG$ and deletes its schema information from the database; including all the class versions and their property versions.

Set current schema version to sv-name. Every program or query should be written against CSV. This operation designates CSV before applications or query accesses to the database.

**Schema Evolution Operations**

The schema evolution operations of the first group require changes only in the schema information of CSV. In this respect, they are related to earlier works on simulating schema updates using the OODB view (Bertino, 1992; Kim, 1995). However, these approaches have a serious drawback; that is, they do not support operations from our last two groups. In this section, we explain two operations, which raise subtleties in defining their semantics. For complete descriptions of all the operations, refer to Lee & Kim (1997).

**Drop an existing class version C**

This operation drops a class version $C$ from CSV. $C$ is removed from the subclass list of each class version in $P(C)$ and from $P(C_{\text{tr}})$ of each subclass version $C_{\text{tr}}$ of $C$, if any. If $C$ is the only superclass of any subclass $C_{\text{tr}}$, class versions in $P(C)$ become new superclasses of $C_{\text{tr}}$. All the properties that are locally defined in $C$ are also removed from all its subclasses.

A (logical) extent in the RiBS model is attached to each class version. Thus, when deleting a class version, we should consider the issue of how to deal with its extent. There exist two reasonable approaches for class drop in the area of schema evolution. In the first approach, all the instance objects of a class are deleted from the database (Banerjee et al., 1987). However, this semantic may introduce the dangling reference problem. A commercial OODBMS ObjectStore overcomes this problem by nullifying all the references to the deleted object (Object Design, Inc., 1994). However, this makes the operation potentially very time consuming. In the second approach, which is exemplified by the $O2$ system (Zicari & Ferrandina, 1997), the class drop operation is allowed only if the extent of the class is empty.

In the RiBS model, another approach is possible, where all objects in $Ext(C)$ are filtered out from CSV. Objects in $Ext(C)$ cannot be accessed through the extent of any class within CSV after a class version $C$ is dropped. However, it should be noted that all the physical objects still exist in RiBS.

In the object-oriented data model, a class may be used as domains of attributes of other classes. Hence, an object may have the OID of another object as its attribute value. This characteristic of object traversal through OID may introduce the “phantom reference” problem under our previous semantics of class drop. As shown in Figure 2, even after a class version $C$ is dropped, the object $c1$ is still accessible through object $a2$. Under the last semantics, however, object $c1$ cannot be accessed through the extent of any class version in CSV; that is, $c1$ is a phantom object. This phantom reference problem leads us to choose a compromised semantics for class drop. Within CSV, all objects in $Ext(C)$ are migrated (logically) to the extent of a superclass of $C$. For example, in Figure 2, all objects in $Ext(C)$ are migrated to $Ext(B)$ after the class version $C$ is dropped. Under this semantic for class drop, multiple inheritance complicates the situation: to which superclass should the logical extent of the class being dropped migrate? In order to guarantee invariant 2, we require users to explicitly designate a target superclass in the RiBS model.

**Add an edge to make class version S a superclass of class version C**

This operation adds a class version $S$ to $P(C)$. This operation is rejected if it introduces a cycle or a redundant ISA within CSV. $C$ inherits all the properties of $S$. This operation
also affects RiBS, except in the following two cases. The first case is where \( S \) is deleted from \( P(C) \) within \( CSV \) before this operation occurs. The second is when another schema update in another schema version has already had the required effect on RiBS. These two cases can be inferred by checking whether \( (B(S), B(C)) \) is in \( ISA'(RiBS) \). In the case where \( (B(S), B(C)) \) is not in \( ISA'(RiBS) \), \( B(S) \) is added into \( P(B(C)) \).

In addition, to ensure no redundant ISA invariant in RiBS, the existence of superclasses of \( B(S) \) in \( P(B(C)) \) in RiBS should be checked. If one exists, the inheritance relationship is removed from RiBS. This situation is exemplified in Figure 3, where we assume that \( SV' \) was derived from \( SV \), and class version \( A \) was made a new superclass of class version \( C \) in \( SV' \). Then, when class version \( B \) is added to \( P(C) \) in \( SV' \), a new edge from \( B(B) \) to \( B(C) \) is added and the edge from \( B(A) \) to \( B(C) \) is deleted. This is required to avoid redundant ISA relations in RiBS. Note that, for \( ISA(A,C) \) in \( SV' \), the corresponding \( (B(A), B(C)) \) exists in \( ISA'(RiBS) \).

Implementation Issues

We discuss several issues arising when implementing the RiBS model. In addition, we show that the RiBS model could be supported by current OODBMSs with some extensions and argue that the performance overhead for supporting the RiBS model could be very small.

Data Structures

Figure 4 shows a generic data structure for the implementation of the RiBS model, using the OMT (Object Modeling Technique) notation. The data structures consist of five system classes and their relationships to each other. These classes and their relationships implement the structural components of the RiBS model. The various modeling constructs of the OMT object model, such as “qualified association,” “aggregation,” and “ordering,” are used to describe the data structures concisely and precisely. We assume that readers are familiar with OMT notation. Refer to (Rumbaugh, 1995) for more detailed descriptions regarding the OMT. In current OODBMSs, a module called SM(Schema Manager) maintains the schema information corresponding to system classes Class and Property (Zicare & Ferrandina, 1997). For the implementation of the RiBS model, this SM module needs some extensions to incorporate the system classes for the schema version layer, SchemaVersion, ClassVersion, and PropertyVersion.

Preprocessing

In the RiBS model, a program or query is written against a schema version, and translated so as to run against RiBS for its execution. This translation can be handled by an ODL/OML (Object Definition Language/Object Manipulation Language) preprocessor, as suggested by ODMG (Cattell, 1997). During the translation, the preprocessor might need to interact with the SM module to get information about the schema mapping between RiBS and the current schema version. The final program or query against RiBS can execute without extra run-time overhead.

OID

Two OID schemes, physical OID and logical OID, have been commonly adopted by OODBMSs. A physical OID encodes the permanent address of the object referred to by itself. This approach provides efficient access to disk-resident objects, but lacks location independence. In contrast, a logical OID is generated by the object storage system independently of the physical address of an object. Thus, this representation allows flexible object movement and replication, but with some performance degradation due to the mapping overhead between logical OIDs and their physical addresses. Because a program or query in the RiBS model runs on the RiBS layer after translation, the RiBS model can be supported by any OODBMS, regardless of its OID scheme.

Implementations using SOP ODMG-compliant OODBMS

SOP (SNU OODBMS Platform) is an ODMG-compliant OODBMS developed from scratch at Seoul National University (Ahn, Lee, Song, & Kim, 1997). SOP consists of several modules, including an object storage system (Soprano), an SM (schema manager) module, an ODMG ODL/OML C++ preprocessor, and a cost-based query processor. Soprano supports a physical OID scheme. The SM module maintains the class and property information and supports basic schema evolution primitives from Orion. The current ODMG ODL/OML C++ preprocessor was developed to provide a seamless integration of C++ programming with SOP by enabling the persistence to be orthogonal to the type. The ODMG standard is going to include specification on
After equipping with the SOP system with these schema evolution operations, we would like to implement the RiBS model on SOP by extending its current schema manager module and make its query processor to understand the schema version concept.

**Related Work**

In the field of OODBs, there have been several researches related to the RiBS model regarding views, schema versions, and schema evolutions. There have been several attempts to support views in OODB (Abiteboul & Bonner, 1991; Kim, 1995; Rundensteiner, 1992). Abiteboul and Bonner (1991) described a view mechanism, which allows class hierarchy restructuring and supports virtual classes in the context of the O2 data model. Rundensteiner (1992) proposed a MultiView methodology, where a view schema from a global schema can be defined according to need. Kim (1995) presented a view semantic for UniSQL Object/Relational DBMS, by augmenting semantics of relational views with object-oriented concepts such as inheritance and OID. In addition, they extend the use of views to dynamic windows for schema, with which schema evolution can be simulated without affecting the database. This is along the same line as the approach in Bertino (1992). Our RiBS approach is similar to these articles in that each schema version is defined over one base schema RiBS. However, there is a big difference between the RiBS model and the work on views in OODB. While direct schema updates against a schema version are allowed in the RiBS model, earlier works (Bertino, 1992; Kim, 1995; Rundensteiner, 1992) a view schema can be changed only by redefining a new view from scratch after deleting the old one.

Kim and Chou (1988) is the first substantial research on schema versions in OODB, based on the object version model of ORION (Kim, 1991). They expressed the schema version model as several rules about schema version management and access scope. According to the access scope rules, each schema version has a different set of objects visible to it, that is, the access scope of the version. An instance object may thus not be shared among schema versions. In contrast to the RiBS model, a new schema version can be derived from only one parent schema version and thus the SVDH results in a tree. Another approach to schema versions is found in (Ra & Rundensteiner, 1995). This work is the most similar to ours in that it also supports schema evolution through views, sharing of instance objects among all the schema versions, and schema merging. However, the authors do not consider such issues as phantom references and conflicts arising from schema merging. In addition, their automatic classification algorithm introduces a new class in the global schema for every capacity-augmenting schema update, which makes the global schema complicated.

During the past decade there has been much research on schema evolutions in OODB (Banerjee et al., 1987; Penney & Stein, 1987; Zicari & Ferrandina, 1997). These articles consider two important issues in schema evolution: semantics of schema change operations and adaptation of objects. The second issue is beyond the scope of this paper, and for details refer to Banerjee et al. (1987). A basic solution to the first issue is to define a set of invariants that should be satisfied by the schema and then to define rules and/or procedures for each schema change operation to guarantee the invariants. In this respect, the RiBS model can be taken as an extension of this framework toward support of schema versions, with substantial add-ons. First, we identify several new invariants for schema versions and RiBS. Second, we extend the semantics of primitive schema change operations to guarantee all these invariants.

**Conclusion**

We strongly believe that the functionality of schema versions will be a prerequisite for OODBMSs to be widely accepted by new database applications. We proposed a schema version model for OODBs based on the concept of RiBS. To avoid anomalies such as phantom reference and multiple classification, we introduced several invariants. In addition, we gave the taxonomy of schema update operations over schema versions and defined their semantics. In a practical aspect, the RiBS model provides OODBMSs with the logical data independence at a level comparable to that of RBDMSs.

For future work, we intend to enhance the modeling power of the RiBS model. With the current RiBS model, customization of the class hierarchy is somewhat restricted. Hence, we will incorporate operations such as class partition-
ing, class merging, and dynamic class (Abiteboul & Bonner, 1991; Kim, 1991; Papazoglou & Kramer, 1997) into the RiBS model. Also, we would like to extend all three elements of the RiBS model to support the reorganization of complex objects, This extension makes it possible for OODBMSs to support customized WWW views naturally (Yang & Kaiser, 1996). Role-class defining operations based on inter-object relationships of ORM (Papazoglou & Kramer, 1997) shed light on the road.

Endnote
1 This research was partially supported by Research Institute of Engineering Science, Seoul National University, under project “A Development of Object-Oriented Component Technology for a Web Transaction Server.”

Reference

Sang-Won Lee received his B.S., M.S. and Ph.D. degrees in computer engineering from Seoul National University, Seoul, Korea, in 1991, 1994 and 1999, respectively. He is now working for Oracle Korea. His research interests include schema management for object-oriented databases, data warehouse, ERP and EC.

Hyung-Joo Kim received his B.S. degree in computer engineering from Seoul National University, Seoul, Korea, in 1982 and his MS and PhD in computer engineering from University of Texas at Austin in 1985 and 1988, respectively. He was an assistant professor of Georgia Institute of Technology, and is currently an associate professor in the Department of Computer Engineering at Seoul National University. His research interests include object-oriented databases, multimedia databases, HCI, and computer-aided software engineering.