Rich Base Schema(RiBS) :
A Unified Framework for OODB Schema Version

Abstract

In this paper, we propose a model of schema versions for object-oriented databases called *RiBS*. At the heart of this model is the concept of the Rich Base Schema (*RiBS*). In our model, each schema version is in the form of a class hierarchy view over one base schema, called *RiBS*, which accumulates all the necessary schema information ever defined in any one of the schema versions. Users, insulated from *RiBS* layer, access databases only through schema versions. Moreover, users can impose schema evolution directly on schema versions, and its effects are, if necessary, automatically propagated to *RiBS*. We first describe the structural part of the model and then introduce a set of invariants that should always be satisfied by structural parts. As the third element of our model, we give a set of schema update operations, the semantics of which are defined so as to preserve all the invariants.

1 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 et al., 1987; Penney, & Stein, 1987; Zicari, & Ferrandina, 1997). This is mainly because target applications of OODBMSs, such as CAD/CAM, CASE, and multi-media frequently 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.,) support various schema update primitives and provide on-line schema evolution mechanisms. In addition, some products, such as O2, Objectivity and ObjectStore, support user-defined functions for schema updates.

Under these systems, however, only a single schema can exist at any point in time; if a schema evolution operation completes, the previous schema state is no longer maintained. This single 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. Schema version mechanisms were introduced to overcome these problems, and many researchers have stressed

their importance since a characteristic of design applications is being able to cope with frequently changing meta-data (Kim, & Chou, 1988; Lautemann, 1996).

Recently, the necessity for schema versions has been rejuvenated in several new OODB application areas including Repositories (Bernstein, 1997; Bernstein et al., 1997; Silberschartz et al., 1995), Portable Common Tool Environment (PCTE) (Charoy, 1994; Loomis, 1992; Wakeman, & Jowett, 1993), and the Word 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 (Silberschartz et al., 1995). Among many requirements for repository management systems, the ability to change the structure of information and its meta-data without breaking existing applications (that is, the functionality of schema versions) is mandatory (Bernstein et al., 1997; Silberschartz et al., 1995). Another strong requirement for the functionality of schema version comes from PCTE, where, as Loomis says (Loomis, 1992), an important role of the OODBMS is to manage PCTE schema, support their evolution over time, and support the resulting schema versions. In fact, schema evolution and schema version management become a more serious problem in PCTE (Charoy, 1994). Finally, WWW applications, needless to say the most promising areas for OODBMS (Atwood, 1996; Bapat et al., 1996; Yang, & Kaiser, 1996), also require schema versions due to their dynamic nature. Atwood points out in (Atwood, 1996) that *"web sites must be able to publish new versions of their applications that use new versions of the database schema $\cdots$. The new versions of the application see the new schema; the existing versions of the applications see the old version of the schema $\cdots$"*.

1.1 Contributions

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 yet. Traditional schema version approaches has three outstanding problems: (1) storage overhead for redundant objects (Kim, & Chou, 1988; Ra, & Rundensteiner, 1995), (2) limited schema update capabilility (Bertino, 1992), and (3) complexity for managing consistent schema versions (Monk, & Sommerville, 1993). Refer to section 8 for the details. The *RiBS* model overcomes all the above problems, thus facilitating efficient and flexible schema management. The main contributions of the *RiBS* model can be summarized as follows.

1. The *RiBS* model, the first view-based schema version apporoach, proposes a framework for mapping each schema evolution over schema version 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 their semantics.

2. As the direct consequnce of above contribution, the *RiBS* model solves the limted schema update capability of traditional view-based schema evolution approach (Bertino, 1992), and, in contrast to class versioning apporach (Monk, & Sommerville, 1993), allows to manage schema version more easily. That is, with the *RiBS* model, users are allowed to impose all the well-known schema evolution operations (Banerjee et al., 1987) over schema verson without any restriction like in (Bertino, 1992), and the effects of these schema changes are reflected into database without any user intervention as in (Monk, & Sommerville, 1993).

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

1.2 Our Perspectives and Paper Organization

A database schema is a representation of entities and their semantics in the real world, which the database is intended to model. In our view, a schema version in an OODB is another schema, the purpose of which is either to represent a semantically significant snapshot of a schema at a point of time under the ever-evolving real world, or to customize different but simultaneous user perspectives.

In this paper, we propose a simple, yet powerful model of schema versions for OODBs, based on the concept of the Rich Base Schema (*RiBS*). The remainder of this paper is organized as follows. Section 2 gives a brief overview of the *RiBS* model using an illustrative example. Section 3 describes the object model assumed in this paper. In section 4, 5, and 6, we give a detailed description of each component of the *RiBS* model, that is, 1) structural part, 2) a set of invariants, and 3) schema evolution operations and their semantics, respecitevely. Section 7 touches upon several issues about the implementations of the *RiBS* model. Our work is compared to related work in Section 8. Section 9 concludes the paper with a summary and an outline of future work.

2 Basic Idea

In this section, we illustrate some basic ideas of the *RiBS* model with an example. A detailed description of each component of the model will be given in the following sections.

For brevity, in this section, we assume the following informal description of a schema in an OODB: A schema in a database consists of classes that are organized into a 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.

2.1 Rich Base Schema and Schema Versions

Before proceeding with the example, we motivate the concept of "Rich Base Schema", and discuss how it can be exploited in supporting schema versions. We say that schema S1 is richer in schema information than schema S2 if all the conditions set out below hold.

1. *For each class in schema S2, there is a corresponding class in S1.*
2. *For each property of a class in schema S2, there exists a corresponding property in the corresponding class of S1.*
3. *For every direct ISA relationship in schema S2, there is also a corresponding ISA relationship (direct or indirect) in schema S1.*

If a schema S1 is richer than another schema S2, it means intuitively that S1 has more schematic information than S2. This, in turn, means that S2 can be specified as a subset view of S1. This concept of rich schema can be re-stated in terms of *relative information capacitey* (Hull, 1984; Miller et al., 1995); *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 existing schema version, is maintained, and every 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 schema versions. In summary, a schema version is an updatable class hierarchical view [1] over *RiBS*.

In our model, schema versions are strictly separated from *RiBS*. This separation enables the prevention of problems from occurring 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 one class hierarchy. However, this approach has several disadvantages (Kim, 1995). First, it is difficult for users to understand the complicated class hierarchy. Next, the extents of classes in the hierarchy may overlap. Finally, it is difficult and, in certain cases impossible, to decide where to locate the view class in the class hierarchy.

2.2 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*. As illustrated in Figure 1, 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,

[1] "updatable" means that the schema evolution operation can be directly imposed on the view.

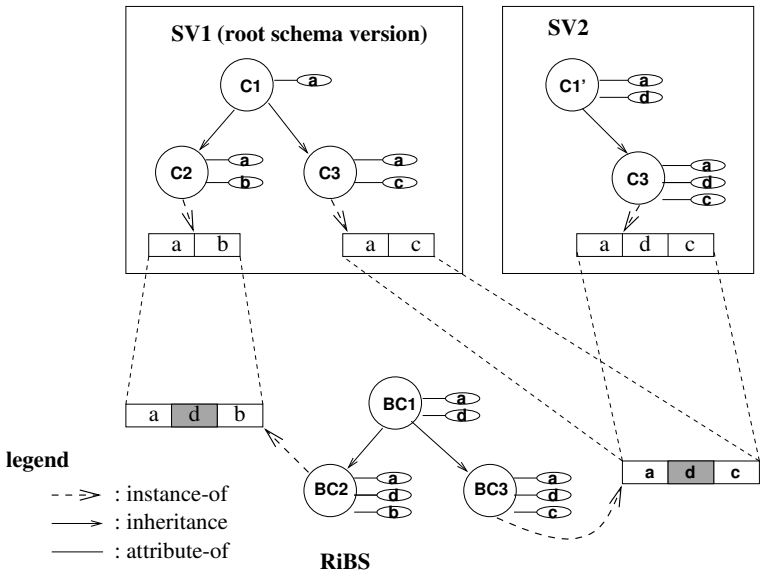

Figure 1: *RiBS* and Schema Versions: An Example

BC1 in *RiBS* is the direct base class of class version C1 in SV1. [2] 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 in *RiBS*. Under a specific schema version, an instance object of a class version is derived from an instance object of the corresponding base class through projection. ((Monk, & Sommerville, 1993)), where an object shared by several class versions has different types under the different class versions.

In Figure 1, we assume that after SV1 is created initially (we call it the *root schema version*) and then SV2 is derived from it, SV2 undergoes three schema updates as follows: (1) rename class version C1 as C1', (2) drop class version C2, and (3) add attribute version d to C1'. We now describe how each schema update affects SV2 and/or *RiBS*.

In our model, each schema version maintains names for its own class versions and their property versions, independently of base classes and properties in *RiBS*. Thus, the first operation renames the class version C1 as C1' only within SV2, without affecting *RiBS* or SV1. The second operation just drops the schema information of class version C2 from SV2, without effecting *RiBS* or SV1. However, this operation, in contrast to the previous operation, raises a subtle semantic issue: that is, the effect of dropping c2 on c2's logical extent. For this, the *RiBS* model chooses the semantics to migrates all the (logical) direct instances of C2 to the extent of a superclass. Thus, class version C1' has base classes BC1 and BC2 as its extental base classes. Details will be discussed in Section 6. The last operation, adding attribute d to class version C1', is different from the above two operations as it requires a change in *RiBS* as well as changes in SV2: a corresponding attribute

[2] In this paper we use the naming convention such as BC1 and BC2 for classes in *RiBS*. This is only for illustrative purpose. Mechanisms such as object identity (OID), which can uniquely identify class and properties within a system, will be sufficient.

should be added to the direct base class of C1' in *RiBS*.

In this section, we gave an overview of the *RiBS* model. The *RiBS* model has three components: (1) the structural part, which consists of schema versions and a *RiBS*, (2) a set of invariants to preserve semantic consistency within the structural part, and (3) a set of schema manipulation operations, which are applicable to the schema versions. In Sections 4, 5, and 6, we will elaborate on these components, respectively.

3 Object Model

This section defines the object model assumed in this paper, 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. [3] Each class may have more than one superclass, that is, multiple inheritance is supported. 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 $P^*(C)$.

Term	Description
C	Class
S	*RiBS* or schema version
$\mathcal{T}(S)$	Set of all classes of schema S
$C.name$	Class name
p	Property
$P(C)$	Set of direct parents of C
$P^*(C)$	Set of all parents of C
$ISA(C_1, C_2)$	C_1 is a direct subclass of C_2
$ISA^*(S)$	All direct or indirect ISA relationships within schema S
$I(C)$	Interface of class C (that is, the set of properties)
$IP(C)$	Inherited properties of C
$LP(C)$	Locally defined properties of C
$E(C)$	Extent of class C (set of direct instances)
$Org(p)$	Original property of p

Table 1: Notations for object model

Table 2 summarizes the inheritance semantics of the object model. As pointed out in (Taivalsaari, 1996), although much research has been focussed on inheritance, researchers rarely agree on its meaning and usage. Even in ODMG-93[4] (Cattell, 1996), no clear semantics are given for

[3] In this paper, we use the term "property" to represent both attributes and methods. When we need to distinguish them, we will use the specific terminology.

[4] an object database standard from ODMG(Object Database Management Group)

inheritance, in particular multiple inheritance. Thus, we need to develop these axioms to clarify the inheritance semantics in *RiBS* model.

(1) Axiom of Closure	$\forall C \in \mathcal{T},\ P(C) \subset \mathcal{T}$
(2) Axiom of Acyclicitey	$\forall C \in \mathcal{T},\ C \notin P^*(C)$
(3) Axiom of Rootedness	$P(Object) = \{\} \wedge \forall C \in \mathcal{T} - \{Object\},\ Object \in P^*(C)$
(4) Axiom of Interface	$I(C) = IP(C) \cup LP(C)$
(5) Axiom of Property Inheritance	$IP(C) = \cup_{C' \in P(C)} I(C')$
(6) Axiom of Superclasses	$P^*(C) = \cup_{C' \in P(C)} P^*(C') \cup P(C)$

Table 2: Axioms for Inheritance

Through axioms 1 to 3, we force a schema to be a Direct Acyclic Graph (DAG). Axiom 4 means that, as mentioned above, the interface of a class consists of inherited properties and locally defined properties. The inherited properties of a class C, as stated in axiom 5, are the unions of the interfaces of all the superclasses 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++*).

4 Structures

The structural component of the *RiBS* model has a three-level architecture: (1) the (extensional) 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 one of the schema versions. Each schema version is in the form of a class hierarchy view over this *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*. In this section, 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 the schema version itself is also a class hierarchy. It is also a logical view in the sense that all the objects visible through a schema version are derived from the objects stored in the extensional object base.

Definition 3 (Current Schema Version, CSV) We call a specific schema version, under which application programs/users access and manipulate the database at a certain point of time, *current schema version (CSV)*.

With the *RiBS* model, a user should designate a schema version as CSV before (s)he accesses the database. A user can do all the normal database operations against the CSV. Moreover, users can change the schema structure of CSV; that is, the schema can evolve. We will give a detailed description of this mechanism in a later section and describe the semantics of schema changes on schema versions.

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 explicitely derive a new schema version from existing ones: the former is called the *"child schema version"* and the latter *"parent schema version(s)"*. *Derived-from* relationships between schema versions constitute the Schema Version Derivation Graph, defined as follows:

Definition 4 (Schema Version Derivation Graph, $SVDG$) *A Schema Version Derivation Graph (SVDG)* 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 need to be explicitely modeled in a schema version *SV*, so there might not be a corresponding class version in *SV*. With respect to schema information capacitey, *B(CV)* is a superset of CV; that is, $B(CV)$ has all the schema information necessary for CV. The purpose of maintaining the direct base class is as follows:

when a specific schema update is imposed against CV and its effects need to propagate to *RiBS*, the schema change in *RiBS* starts from $B(CV)$.

As mentioned earlier, users are concerned only with schema versions. Hence, each class version in a schema version, like normal classes, is expected to have its own class extent. For this, we maintain extental base classes for each class version.

Definition 6 (Extental Base Classes, $B^+(CV)$) The *Extental 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 extental base classes, the logical extent of a class version CV is derived as follows.

$$E(CV) = \cup_{C' \in B^+(CV)} E(C')$$

As will be discussed later, the set of base classes in $B^+(CV)$ is a connected subgraph of *RiBS*, rooted at $B(CV)$ (thus, we employ the notation B^+). Therefore, all the objects in $B^+(CV)$ carry values for all the properties necessary in CV.

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 of the CV. The *Direct base property* of a PV in a class version CV, denoted as $B(PV)$, is a corresponding property of $B(CV)$. Every PV in a schema version has its direct base property.

The purpose of maintaining a direct base property for each PV is as follows. When a logical object is retrieved through CSV, its values are 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 extental base classes, however, complicates this process in that, if the corresponding physical object of an object being accessed under CSV is an instance of a base class which 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? As stated in the previous section, we assume that in *RiBS*, a subclass inherits all properties from its superclass(es) and keeps the property information locally, as in Orion (Banerjee et al., 1987). Thus, when deriving the value of each PV, the value of the property which has $B(PV)$ as its origin property is used.

5 Invariants

In this section, as a second component of the *RiBS* model, we introduce a set of invariants which should always be satisfied by the structural part. Moreover, this set of invariants plays a critical role in defining the semantics of schema evolution operations for schema versions, as discussed in the next section.

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 paper, we describe the last two categories. With respect to *RiBS*, we assume the well known invariants for schema evolutions, such as DAG invariance, name invariance, origin invariance, full inheritance invariance, and no redundant ISA relationships from (Banerjee et al., 1987; Penney, & Stein, 1987; Rundensteiner, 1992; Zicari, & Ferrandina, 1997).

5.1 Invariants on Schema Version

For schema versions, in addition to the traditional invariants for schema evolutions, 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. In the next section, we will explain how these two invariants guide the semantics of **class drop**.

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 SV*

- *for each object O referenced by another object, there should exist a class version CV, where* $O \in Ext(CV)$.

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_i *and* CV_j *in a schema version SV, where* $i \neq j$, $Ext(CV_i) \cap Ext(CV_j) = \phi$

5.2 Invariants between *RiBS* & Schema Version

As mentioned above, each schema version is a logical view over *RiBS*. The following 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 $C \in B^+(CV)$.*

The above invariant means that the union of the extental base classes of all class versions in a schema version should be equal to the set of base classes in $RiBS$, as formalized in the following.

$$\bigcup_{CV_k \in \mathcal{T}(SV)} B^+(CV_k) = \mathcal{T}(RiBS)$$

6 Operations

In this section, we give a set of operations for schema version management, which is the last component of the *RiBS* model. These operations are classified into two groups: one group is concerned with $SVDG$ manipulation, while the other includes schema evolution operations against schema versions. Schema evolution primitives available in *RiBS* model are similar to those from (Banerjee et al., 1987; Penney, & Stein, 1987). In this paper, we make a new taxonomy for the eight fundamental schema change operations of Orion, depending on their impacts on *RiBS* and other schema versions. These eight fundamental operations are complete in the sense that all the schema changes can be achieved by combining these operations (Kim, 1988).

- 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

6.1 Operations for $SVDG$ manipulations

Derive *sv-name* **from** *parent-list* As mentioned before, this operation is used to derive a new schema version *sv-name* from existing ones in *parent-list*. When a schema version is derived from

a single parent, this operation can easily be implemented; that is, the schema information of the parent is simply copied into the child. At this point, the schema information of both parent and child, including class versions and their property versions, is exactly the same. In the case of multiple parents, however, the parent schema versions with different structural information should be merged into a new consistent one. We call this process *"schema-version-merging"*. We identified several conflicts in schema-version-merging, and then devised a semi-automatic schema-version-merging algorithm to resolve these conflicts. For a complete and detailed description, refer to (Lee, & Kim, 1997).

Delete *sv-name* When a schema version is no longer needed, this operation is used to remove it from *SVDG* and delete its schema information from the database; that is, all the class versions and their property versions are deleted.

Set *current schema version* **to** *sv-name* As mentioned above, every program or query in the *RiBS* model should be written against a specific schema version, called the *current schema version (CSV)*. This operation is used to designate CSV before applications or query accesses to the database.

6.2 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 paper, we will explain three operations, each of which is from each group and raises subtleties in defining their semantics. For complete descriptions of all the operations and their formal semantics, refer to (Lee, & Kim, 1997).

6.2.1 Drop an existing class version C

This operation drops a class version C from CSV. C is dropped out from the subclass list of each class version in $P(C)$ and from $P(C_{sub})$ of each subclass version C_{sub} of C, if any. If C is the only superclass of any subclass C_{sub}, class versions in $P(C)$ become new superclasses of C_{sub} (Banerjee et al., 1987). All the properties that are locally defined within C are also dropped from all its subclasses.

As mentioned before, 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. In relation to this issue, there have been at least two reasonable approaches for **class drop** (Banerjee

et al., 1987; Object Design, Inc., 1994; Zicari, & Ferrandina, 1997) 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; Object Design, Inc., 1994). However, this semantics, as pointed out in (Banerjee et al., 1987), may introduce the dangling reference problem. A commercial OODBMS, ObjectStore (Object Design, Inc., 1994), overcomes this problem by nullifying all the references to the deleted instance objects. However, this, in turn, makes the operation potentially very time consuming (Object Design, Inc., 1994). 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, there could be another possible approach to class drop, where all objects in $Ext(C)$ are filtered out from CSV. According to this approach, objects in $Ext(C)$ cannot be accessed through the extent of any class within CSV when the class drop operation is completed. However, it should be noted that all the physical objects still exist in *RiBS*. This approach seems to be similar to the view mechanism in relational databases, which provides the functionality of content-based authorization (Kim, & Seo, 1991), in that it hides some objects from the view of the user. Many other researchers have anticipated that some kind of view mechanism for OODB will also provide the same functionality of content-based authorization (Bertino, 1992; Kim, 1995).

However, navigational object access through object identity (OID) in the object-oriented data model is drastically different from the relational data access paradigm where the only unit of access is either table or view. 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 value for an attribute. This characteristic of object traversal through OID introduces the *"phantom reference"* problem under our previous semantics of **class drop**. As shown in Figure 2, even after a class version CV 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. It should be noted that the phantom reference problem is not confined to the *RiBS* model. Any view mechanism in OODB should consider and solve this phantom reference problem in order to provide for the functionality of content-based authorization. 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 semantics 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 explicitely designate a target superclass in the *RiBS* model

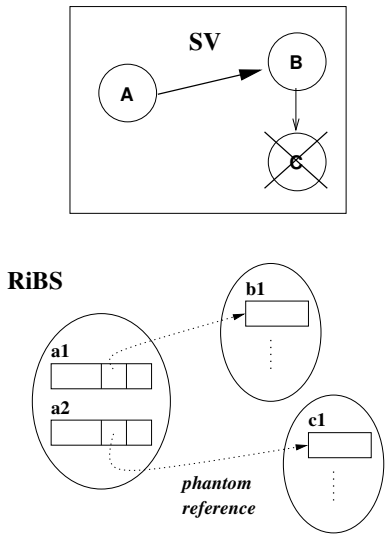

Figure 2: Phantom Reference

6.2.2 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 th 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 any direct or indirect superclass 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 schema version SV_j was derived from SV_i and class version A was made a new superclass of class version C in SV_i. Then, when a schema update which adds class version B to *P(C)* is imposed on SV_j, 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_i, the corresponding $(B(A), B(C))$ exists in $ISA^*(RiBS)$.

6.2.3 Create a new class version C as a subclasss of S

This operation creates a new class version in CSV. If any class version with the same name already exists in CSV, the operation is rejected. To satisfy the **invariant 3**, a new base class $B(C)$ needs to be created in *RiBS*. Direct base classes of each superclass of C become the superclasses of $B(C)$. In addition, the direct base class of each domain of an attribute defined in C becomes the domain

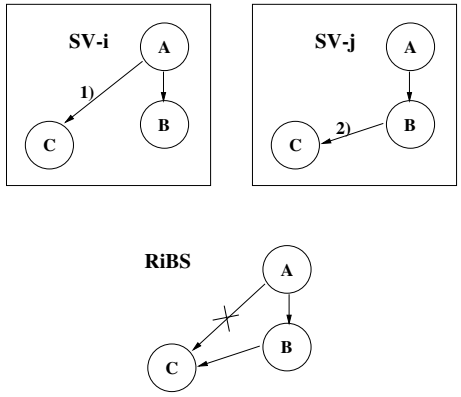

Figure 3: Addition of a Superclass to a Schema Version

of the attribute in $B(C)$. The base class is created with the superclass list and property list, and then the new base class is set to *B(C)*.

This operation affects other schema versions, in addition to $RiBS$. According to **invariant 4**, *B(C)* needs to be included into extensional base classes of an appropriate class version in schema versions other than CSV. In order to do this, we choose the following solution: "in schema versions other than CSV, add $B(C)$ to the extental base classes of a class version CV which has $B(S)$ as its extental base class" (refer to formal semantics in the next section).

7 Implementation Considerations

In this section, we discuss several issues that should be considered when implementing the *RiBS* model, including data structures, preprocessing, object adaptation, object identity, and space optimization. In addition, we show that the *RiBS* model could be supported by current OODBMSs with some extensions, and argue that the performance overhead to support the *RiBS* model is small.

Figure 4 shows a generic data structure for the implementation of the RiBS model, using the OMT (Object Modeling Technique) notation (Rumbaugh, 1995). 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", [5] are used to describe the data structures concisely and precisely. In current OODBMSs, a module called SM (*scheme manager*), maintains the schema information corresponding to system classes Class and Property (Zicari, & 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,

[5] We assume that readers are familiar with OMT notation. Refer to (Rumbaugh, 1995) for more detailed descriptions regarding the OMT.

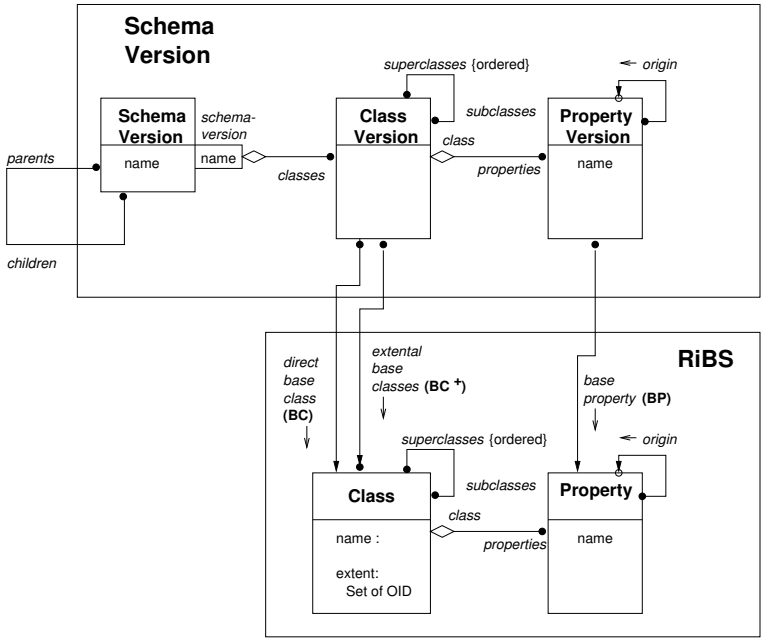

Figure 4: Systems Classes for the *RiBS* Model

SchemaVersion, ClassVersion, and PropertyVersion.

In the *RiBS* model, a program or query is written against a schema version, and translated so as to run against *RiBS*. This translation can be handled by an ODL/OML (Object Definition Language/Object Manipulation Language) preprocessor (Cattell, 1996), as suggested by ODMG. During the translation, the preprocessor might need to interact with the SM module to get information about the schema mapping between *RiBS* and *current schema version*. The final program or query against *RiBS* can be executed without extra run-time overhead.

In general, there have been two approaches to the adaptation of objects (Banerjee et al., 1987; Penney, & Stein, 1987; Zicari, & Ferrandina, 1997), changing the representation of affected objects to a state consistent with the new schema. The first approach is *deferred update*, where the format of each object is changed only when it is accessed after schema updates. The second approach is *immediate update*, in which affected objects are updated instantly upon schema updates. This paper is mainly concerned with the semantics of schema version evolutions for both schema versions and *RiBS*; thus, object adaptation is not within the scope of this paper. However, either approach can be applied to bring physical objects residing in an extensional base up to a consistent state in the *RiBS* model.

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. As mentioned above, 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 the OID scheme used.

With the *RiBS* model, there might be opportunities for space optimization. For example, consider a base property for which no corresponding property version exists in any schema version. Physical objects in the extensional base reserve space for the obsolete base property, but the space is no longer necessary because the information kept there is not accessible through any schema version. This fact can be exploited by the database administrator by dropping unnecessary base properties from *RiBS* periodically.

SOP (*S*NU *O*ODBMS *P*latform) is an ODMG-compliant OODBMS developed from scratch at Seoul National University (Ahn et al., 1997). SOP consists of several modules, including an object storage system (*Soprano*) (Ahn, & Kim, 1997), a 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. We plan to implement the *RiBS* model on SOP by extending the SM module, the preprocessor, and the query processor to understand the schema version layer.

8 Related Work

In the field of OODBs, there have been several research activities closely related to the *RiBS* model, including papers on views, schema versions/evolutions, and dynamic objects. In this section, we summarize these articles and outline their differences from the *RiBS* model.

8.1 Views and the *RiBS* model

There have been several attempts to support views in OODB (Abiteboul, & Bonner, 1991; Kim, 1995; Rundensteiner, 1992). In (Abiteboul, & Bonner, 1991), in the context of the O_2 data model, a view mechanism which allows the restructuring of the class hierarchy and supports virtual classes is described with a number of examples. In (Rundensteiner, 1992), the authors proposed a MultiView methodology, where a view schema from a global schema can be defined according to need. (Kim, 1995) presents a view semantic within an Object/Relational DBMS, UniSQL, by augmenting semantics of relational views with object-oriented concepts such as inheritance,

method, and OID. In addition, they extend the use of views to dynamic windows for schema, with which schema evolution in OODB can be simulated without affecting the database. This is along the same line as the approach in (Bertino, 1992) simulating schema evolution using views.

Our *RiBS* approach is similar to these articles in the sense that each schema version is defined over one global 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, in earlier work a view schema can be changed only by redefining a new view from scratch after deleting the old one.

8.2 Schema Versions/Evolutions and the *RiBS* model

The work in (Kim, & Chou, 1988) is the first substantial research on schema versions in OODB, based on the object version model of ORION (Banerjee et al., 1987). In this work, the schema version model is expressed 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 schema version derivation hierarchy results in a tree.

Another approach to schema versions is found in (Ra, & Rundensteiner, 1995). This work is 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 in schema merging, including homonyms/synonyms and extental migration conflicts. In addition, their automatic classification algorithm introduces a new class in the global schema for every capacitey-augmenting schema update, which makes the global schema complicated.

As an alternative to schema versions, there has been the class versioning approach (Kim, 1988; Monk, & Sommerville, 1993), where the units of versioning are individual classes, instead of the entire class hierarchy. We will return to this approach in the next subsection.

During the past decade there has been much research on the subject of 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 was touched upon in section 7. A basic solution to the first problem 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 another extension of this framework toward support of schema version functionality, with substantial add-ons. First, we identify several new invariants for schema versions and *RiBS*, in addition to traditional invariants for schema evolution. Second,

we extend the semantics of primitive schema change operations to guarantee all these invariants.

8.3 Dynamic Objects and *RiBS* model

As noted in (Monk, & Sommerville, 1993; Papazoglou, & Krämer, 1997), many complex application domains require *dynamic object* mechanism where an individual object can change its class(and type) as time goes by, or an object has different states and behaviors under different contexts. However, traditional class-instance relationships of object oriented data model fail to support the concept of dynamic object because it strictly requires an object to be a direct instance of only one class(and thus only one type) at a certain time point. The works in (Monk, & Sommerville, 1993; Papazoglou, & Krämer, 1997) try to overcome this drawback of object oriented data model.

(Monk, & Sommerville, 1993) proposes a class versioning system CLOSQL, based on *dynamic instance conversion* which enables an instance object to be seen from the outside by a number of class version interfaces, and determines the type of an instance object by the context of concern(that is, *dynamic instance objects*). In this respect, we can argue that, in *RiBS* model, a physical object residing in extensional bases is also dynamic object since it changes its type dynamically depending on *CSV(current schema version)* accessing the object.

ORM(Object Role Model) (Papazoglou, & Krämer, 1997) is a database model for dynamic objects, with the central concept of *role*. A role refers to the ability to change the classification of an object dynamically, so that the same object can simultaneously be an instance of different classes. Based on role concept, several special operators for creating role classes and restructuring class hierarchy are introduced. With this ORM, an individual object can be an instance of different role classes while retaining its object identity. In contrast, *RiBS* model requires a physical object to remain its base class through its life time. In addition, though an object in *RiBS* model can be simultaneously classified into different classes under different schema versions, the capability of dynamic classification in *RiBS* model lacks in power and flexibility compared to ORM.

9 Conclusion

We strongly believe that the functionality of the schema version will be a prerequisite for OODBMSs to be widely accepted by newly emerging database applications, including repositories and the WWW. In this paper, we proposed a schema version model for OODBs based on the concept of *RiBS*. Each schema version is in the form of a class hierarchy view over one global schema, *RiBS*. Users are supposed to be concerned only with schema versions. Direct schema updates on schema versions are allowed, which are, if necessary, automatically propagated to *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. Finally, we touched upon several implementation issues for the *RiBS* model.

We plan two future projects. With the current *RiBS* model, customization of the class hierarchy is somewhat restricted. Hence, we intend to incorporate more operations into our schema update taxonomy, such as class partitioning, class merging, and dynamic class (Abiteboul, & Bonner, 1991; Kim, 1991; Papazoglou, & Krämer, 1997; Wieringa et al., 1995), for increased flexibility. We expect that this will substantially enhance the modeling capability of the *RiBS* model. Next, we plan to extend all three elements of our model to support the reorganization of nested complex objects. This extension makes it possible for an OODBMS to support customizable WWW views naturally (Yang, & Kaiser, 1996). Role-class defining operations based on the inter-object relationships of ORM (Papazoglou, & Krämer, 1997) shed light on the road for the future.

References

Abiteboul, S., & Bonner, A. (1991). "Objects and Views". *Proceedings of the ACM SIGMOD International Conference on Management of Data.*

Ahn, J.-H., & Kim, H.-J. (1997). "Seof: An Adaptable Object Prefetch Policy for Object-Oriented Database Systems". *Proceedings of International Conference on Data Engineering.*

Ahn, J.-H., Lee, K.-W., Song, H.-J., ,, & Kim, H.-J. (1997). "Soprano: Design and Implementation of an Object Storage System". *Journal of Korea Information Science Society(C).*

Atwood, T. (1996). "Object Databases Come of Age". *Object Magazine.*

Banerjee, J., Kim, W., Kim, H.-J.,, & Korth, H. (1987). "Semantics and Implementation of Schema Evolution in Object-Oriented Databases". *Proceedings of the ACM SIGMOD International Conference on Management of Data.*

Bapat, A., Waesch, J., Aberer, K.,, & Haake, J. M. (1996). "HyperStorM: An Extensible Object-Oriented Hypremedia Engine". *The Seventh ACM Conference on Hypertext.*

Bernstein, P. A. (1997). "Repositories and Object Oriented Databases". *Proceedings of BTW '97.*

Bernstein, P. A., Harry, B.,, & Sanders, P. (1997). "The Microsoft Repository". *Proceeding of International Conference on Very Large Data Bases.*

Bertino, E. (1992). "A View Mechanism for Object-Oriented Databases". *Extending Database Technology.*

Cattell, R., editor (1996). *"The Object Database Standard: ODMG-93".* Morgan Kaufmann.

Charoy, F. (1994). "An Object-Oriented Layer on PCTE". *Technical paper available from* `http://gille.loria.fr:7000/oopcte/oopcte.html`.

Hull, R. (1984). "Relative Information Capacity of Simple Relational Database Schemata". *Proceedings of ACM International Conference on PODS.*

Kim, H. J. (1988). "Issues in Object Oriented Database Schema". *PhD dissertation.*

Kim, W. (1991). *"Introduction to Object Oriented Databases".* MIT press.

Kim, W., editor (1995). *"Modern Database Systems: The Object Model, Interoperability, and Beyond".* ACM Press.

Kim, W., & Chou, H. (1988). "Versions of Schema for Object-Oriented Databases". *Proceeding of International Conference on Very Large Data Bases.*

Kim, W., & Seo, J. (1991). "Classifying Schematic and Data Heterogeneity in Multidatabase Systems". *IEEE Computer*, 24(12).

Lautemann, S.-E. (1996). "An Introduction to Schema Versioning in OODBMS". *Proceedings of Database and Expert Systems Applications.*

Lee, S.-W., & Kim, H.-J. (1997). "A Model of Schema Versions for Object-Oriented Databases, based on the concept of Rich Base Schema". *Technical Report, SNU OOPSLA Laboratory.*

Loomis, M. E. (1992). "Object Database - Integrator for PCTE". *Journal of Object-Oriented Programming.*

Miller, R. J., Ioannidis, Y. E.,, & Ramakrishnam, R. (1995). "The Use of Information Capacity in Schema Integration and Translation". *Proceeding of International Conference on Very Large Data Bases.*

Monk, S., & Sommerville, I. (1993). "Schema Evolution in OODB Using Class Versioning". *SIGMOD Record*, 22(3).

Object Design, Inc. (1994). "ObjectStore Technical Overview, Release 3.0".

Objectivity, Inc. "Schema Evolution in Objectivity/DB". *White paper available from* `http://www.objy.com/ObjectDatabase/WP/Schema/schema.html`.

Papazoglou, M. P., & Krämer, B. J. (1997). "A Database Model for Object Dynamics". *The VLDB Journal*, 6(2).

Penney, D. J., & Stein, J. (1987). "Class Modifications in the GemStone Object-Oriented DBMS". *Proceedings of International Conference in Object-Oriented Programming: Systems, Languages, and Applications.*

Ra, Y., & Rundensteiner, E. A. (1995). "A Transparent Object-Oriented Schema Change Approach Using View Evolution". *Proceedings of International Conference on Data Engineering.*

Rumbaugh, J. (1995). "OMT: The object model". *Journal of Object-Oriented Programming.*

Rundensteiner, E. A. (1992). "MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases". *Proceeding of International Conference on Very Large Data Bases.*

Silberschartz, A., Stonebraker, M.,, & Ullman, J. (1995). "Database Research: Achivements and Opportunities Into the 21st Century". *Report of an NSF Workshop on the Future of Database Systesms Research.*

Taivalsaari, A. (1996). "On the Notion of Inheritance". *ACM Computing Surveys*, 28(3).

Wakeman, L., & Jowett, J. (1993). *"PCTE: The Standard for Open Repositories"*. Prentice Hall.

Wieringa, R., de Jonge, W.,, & Spruit, P. (1995). "Using Dynamic Classes and Role Classes to Model Object Migration". *Theory and Practice of Object Systems*, 1(1).

Yang, J. J., & Kaiser, G. E. (1996). "An Architecture for Integrating OODBs with WWW". *Columbia University Tech-Report CUCS-004-96.*

Zicari, R., & Ferrandina, F. (1997). "Schema and Database Evolution in Object Database Systems". *In Part6, Advanced Database Systems.*