Capacity-Augmenting Schema Changes on Object-Oriented Databases: Towards Increased Interoperability*

Elke A. Rundensteiner
Department of Computer Science, Worcester Polytechnic Institute
Worcester, MA 01609-2280, USA
rundenst@cs.wpi.edu

Amy J. Lee, and Young-Gook Ra
Department of EECS, University of Michigan
Ann Arbor, MI 48109-2122, USA
amylee@eecs.umich.edu

Abstract

The realization of capacity-augmenting schema changes on a shared database while providing continued interoperability to active applications has been recognized as a hard open problem. A novel three-pronged process, called transparent object schema evolution (TOSE), is presented that successfully addresses this problem. TOSE uses the combination of views and versioning to simulate schema changes requested by one application without affecting other applications interoperating on a shared OODB. The approach is of high practical relevance as it builds upon schema evolution support offered by commercial OODBMSs.

Keywords: Transparent schema evolution, object-oriented views, object-oriented databases, application migration.

1 Introduction

Current schema evolution technology suffers from the problem that schema updates on a database shared by interoperating applications often have catastrophic consequences [BKKK87, KC88, MS93, PS87, TS93, Zic91]. In such a multi-user environment, a schema change demanded by one application could have an undesired impact on other applications, possibly rendering them inoperational. In current OODB systems, schema update capabilities are hence

*This work was supported in part by the NSF RIA grant #IRI-9309076, NSF NYI grant #IRI 94-57609, and the University of Michigan Faculty Award Program, IBM Partnership Program, Intel, and AT&T.

more limited by their impact on existing programs rather than by the power of the supported schema change language.

In the context of a 5-year NSF-sponsored effort[1], we are exploring solutions to overcome this schema change problem critical to many industries. Towards this goal, we have proposed a new concept - which we call transparent schema evolution (TSE) – as a promising mechanism for addressing this important problem [RR95]. TSE refers to the notion that a schema change initiated by one user is transparent to (not seen by) other users even though they share a common portion of the schema and the associated data. Our transparency approach is to simulate schema changes as views instead of executing in-place changes on a shared schema, thus shielding other users from these changes [RR97].

However, conventional view mechanisms, being derived functions on the stored database, cannot simulate *capacity-augmenting* [2] schema changes due to their inherent limitation of not augmenting the underlying schema. It has been assumed that one either has to limit oneself to non-capacity-augmenting schema changes or utilize an advanced (capacity-augmenting) view mechanism. In an earlier effort, we [RR95, RR97] as well as Bertino [Ber92] chose the latter approach by extending traditional object-oriented view mechanisms to also support *capacity-augmenting views*. This means that the view mechanism must be capable of the storage of base data directly with view classes. While we have indeed built such an advanced view mechanism, our experiments have shown that this is hard to achieve in an efficient manner [KRR95]. In addition, it imposes restrictions on view classes to be physically materialized so to store these additional attributes with them [KRR95]. More importantly, commercial OODB systems do not support such capacity-augmenting views [3]. For this reason, we now explore an alternative solution to this problem that does *not rely on such powerful (and not available) view support.*

In this paper, we propose a more practical solution to this problem. In contrast to prior work [RR95, RR97], our new approach, which we call TOSE (Transparent Object Schema Evolution), is now neither confined to capacity-reducing or capacity-equivalent schema changes nor requires an advanced view mechanism that supports capacity-augmenting virtual classes. The solution we put forth requires only traditional schema evolution support offered by most commercial OODB systems [BKKK87, PS87] and conventional (derivation-based) view mechanisms [AB91, Run92]. This increases the practical viability of this proposed technology.

The fundamental idea underlying TOSE is a new three-pronged process for

[1] We want to express our thanks to the NSF Expert Systems and Database Program, NSF NYI #IRI-94-57609.

[2] The capacity of a schema is defined as the set of possible states of the database under the schema.

[3] As far as we know, the only exception of a fully implemented capacity-augmenting OODB tool is the MultiView research prototype which we have been developing in our on-going NSF-funded research project.

capacity-augmenting transparent evolution. The three steps are: (1) translate the change request specified on the view to the base schema in order to physically restructure the underlying base schema so to hold sufficient capacity as required by the capacity-augmenting schema change operation; (2) derive the desired target view schema from the augmented base schema; and (3) reconstruct the original base schema from the augmented base schema as a view schema in order to preserve all existing input schemas in the system.

Besides the development of this new solution approach for achieving evolution transparency, another contribution of this paper is to provide the first formal treatment of the *transparency* notion for schema evolution. In particular, we establish *sufficiency conditions* under which a schema change operation can be guaranteed to be *transparently achievable.* This represents a characterization of the expressiveness as well as limitations of the TOSE approach. The sufficiency criteria identifies the notion of *update semantic preserving* (usp) as a key requirement of the view derivation underlying TOSE. To ground these formal requirements, we conducted a case study using MultiView [KR96a] that confirms that object-oriented view systems can satisfy the *usp* requirements (needed for TOSE).

Using this formal foundation, we show that the schema change operations achievable in TOSE are comprehensive since they include the standard set of operations commonly supported by OODB systems, i.e., those proposed in [BKKK87] and adopted in most other schema evolution research [Kim88, MS93, SZ86]. The new TOSE framework allows us to achieve transparency not only for capacity-equivalent and -reducing, but also for capacity-augmenting operations. To validate its feasibility, we have implemented a working TOSE prototype [4] using the Gemstone OODB [5].

The remainder of this paper is structured as follows: Section 2 reviews related research, while Section 3 introduces the basic object and view model. In Section 4, we give an overview of the new TOSE approach. Section 5 identifies the *update-semantic-preserving* property as a key requirement of views for TOSE, while Section 6 presents the theoretical background for TOSE. Section 7 explains the transparent schema change process of our TOSE framework using examples, and Section 8 concludes this paper.

2 Related Research

In our previous work, we have built a TSE system using capacity-augmenting views [RR95, RR97]. Our current results go beyond our previous work [RR95, RR97] in several dimensions. New contributions, besides the new TOSE methodology for achieving transparency without requiring capacity-augmenting views,

[4] An earlier version of the TOSE system has been formally demonstrated at SIGMOD'96 [RKR+96]

[5] GemStone is a registered trademark of GemStone Inc.

include a formal definition of the transparency notion, analysis of update semantics preservation of views required by our approach, theorems characterizing the properties of the TOSE framework, and an implementation of the TOSE system for a comprehensive set of change operations.

Tresch and Scholl [TS93] also advocate views as a suitable mechanism for simulating schema evolution, but they fail to provide solutions for *capacity-augmenting* schema changes. However, *capacity-augmenting* schema evolution tends to be important for extending an existing database under changing database requirements - this is the problem that we address in this paper.

Bertino [Ber92] presents a view mechanism and indicates that it can be utilized to simulate schema evolution. Similar to our original approach of TSE [RR95], the proposed view mechanism is extended to be *capacity-augmenting* in that new stored attributes are added to views. However, as indicated earlier, dependence on such an advanced view mechanism may restrict the practical usage of the approach. Therefore, we investigate in this paper how such transparent schema evolution can be achieved *without* relying on capacity-augmenting views.

To our best knowledge, there is no working system that performs *transparent* schema evolution with the transparency notion as defined in this paper. Some schema version systems (Encore [SZ86], Orion [KC88], Goose [MNK91], CLOSQL [MS93], and COAST [Lau97]) are however conceptually related to our notion of transparency. A major distinction of these versioning approaches in contrast to TOSE is that (1) duplicates of object instances are typically created, which causes serious limitations; (2) updates through newer schema versions are often not propagated to older (now out-of-date) versions; and (3) data inconsistencies between different versions of an object may arise over time. For these reasons, true interoperability between old and new applications is not achieved. A more detailed discussion of versioning systems can be found in [RR95, RR97].

3 Background Material

Object-Oriented Data Model. Below we describe the fairly standard object model we assume, which is compatible with the ODMG model [ea97]. Let $\mathbf{O}$ be an infinite set of object instances. Each object $obj \in \mathbf{O}$ consists of state (the attributes), behavior (the methods to which the object can respond), and a unique object identifier (OID). A property refers to either an attribute or a method. Objects that have common properties are grouped into *classes*. We use the term *type* to indicate the set of properties shared by all objects of the class, and the term *extent* for the set of objects that are direct instances of the class.

Let $\mathbf{C}$ be the set of all classes in a database. For two classes C_i, $C_j \in \mathbf{C}$,

C_i is a *subtype* of C_j, if and only if (iff) $type(C_i) \supseteq type(C_j)$. All properties defined for a supertype are inherited by its subtypes. C_i is a *subset* of C_j, iff ($\forall$ obj $\in$ **O**) $((obj \in C_i) \Rightarrow (obj \in C_j))$. For two classes C_i, $C_j \in \mathbf{C}$, C_i is a *subclass* of C_j, denoted by C_i is-a C_j, iff C_i is a subtype as well as a subset of C_j. C_i is a *direct subclass* of C_j, if (C_i is-a C_j) and there are no other classes $C_{k_m} \in \mathbf{C}$ with m = 1, ..., n such that the following is-a relationships hold: (C_i is-a C_{k_1}) and (C_{k_1} is-a C_{k_2}) and $\cdots$ and (C_{k_n} is-a C_j). C_i is called an *indirect subclass* of C_j, denoted as C_i is-a* C_j, if there are one or more classes $C_{k_m} \in \mathbf{C}$ for which the above is-a relationships hold. A set membership *Extent* is defined for a class C_j as $Extent(C_j) = \bigcup$ extent(C_i), $\forall$ C_i is-a* C_j. A class C_i has a unique name, a type, and an Extent.

An *object schema* is a rooted directed acyclic graph S = (**V**,**E**), where **V**, the finite set of vertices, correspond to classes $C_i \in \mathbf{C}$ and **E**, the finite set of directed edges, correspond to a binary relation on $\mathbf{V} \times \mathbf{V}$ that represents all direct *is-a* relationships. Each directed edge $e \in \mathbf{E}$ from C_i to C_j represents the relationship C_i is-a C_j. We allow multiple inheritance, e.g., C_i is-a C_m and C_i is-a C_n, where n <> m. The *Root* class has an Extent equal to all database objects and an empty type description.

View Model. We distinguish between base and virtual classes. Base classes are usually defined by an initial schema definition. A virtual class is defined by the application of a query operator to one or more source classes, which could be base or virtual classes, that restructures the source's type and/or extent membership. Virtual classes can be added dynamically to the schema throughout the lifetime of the database. The virtual class forming operators include: *hide*, *refine*, *select*, *union*, *intersect*, *join*, *difference*, and *ident*, which defines the type and extent of the virtual class [Run92]. These object operators determine the properties and extents of the virtual classes. The is-a relationship between virtual classes is similarly defined as that between base classes, i.e., a virtual class VC_i *is-a* another virtual class VC_j, iff Extent(VC_i) $\subseteq$ Extent(VC_j) and type(VC_i) $\supseteq$ type(VC_j). We provide three generic update operations, *create*, *delete*, and *set* for performing updates [SLT91]. We call a database schema that contains all base classes a *base schema* (BS), and one that contains virtual classes a *view schema* (VS). When it is not important to distinguish among these three schemas we use the term *schema* (S).

Schema Capacity. Like in [MIR93], the *capacity* of a schema S = (**V**, **E**) is defined to be the set of all valid database states of the schema S. We use *states*(S) to denote the set of all possible database states of a schema S (at any time) and use DB_t(S) to denote the data in the database at time t. The following relationship holds: *states*(S) $\supseteq \cup_{\forall t} DB_t(S)$. Intuitively, a schema S_j has a larger or equal capacity than a schema S_i if every database state of S_i can be mapped to a database state of S_j without loss of information.

Schema Transformations. A mapping *f* between two sets A and B, f: A $\rightarrow$ B, is *total* if *f* is defined on every element of A, *functional* if for any a $\in$ A there exists at most one b $\in$ B, *injective* if its inverse f^{-1} is functional, and *onto* if its inverse f^{-1} is total. A functional, injective, total, and onto mapping is called a *bijection*.

Capacity-Reducing (CR)	Capacity-Equivalent (CE)	Capacity-Augmenting (CA)
delete-attribute	change-attribute-name	add-attribute
delete-class	change-class-name	add-class
delete-edge		add-edge
change-attribute-domain (new domain: subclass)		change-attribute-domain (new domain: superclass)

Figure 1: Schema Change Operations Supported by TOSE.

A *schema transformation* $\mathcal{T}$ is a total mapping, $\mathcal{T} : \mathbf{S} \rightarrow \mathbf{S}$ with $\mathbf{S}$ a set of schemas. $\mathcal{T}$ is *capacity-augmenting* if there exists a total and injective mapping f: states(S) $\rightarrow$ states($\mathcal{T}$(S)), $\mathcal{T}$ is *capacity-equivalent* if f is a bijection, $\mathcal{T}$ is *capacity-reducing* if the mapping of the inverse transformation $\mathcal{T}^{-1}$ is total and injective, and $\mathcal{T}$ is *capacity-incompatible* otherwise [MIR93]. In our work, $\mathcal{T}$ corresponds to the standard set of schema change operations proposed in the literature [BKKK87, Zic91] (Figure 1) [6].

4 Transparent Object Schema Evolution (TOSE)

The goal of our proposed TOSE framework is to provide continued support to existing applications cooperating on a shared database, even under a schema change. TOSE achieves this goal by simulating schema changes using views, which shield users/applications from schema changes requested by other users (Figure 2). Users work on their view schemas instead of directly working on the shared schema, while TOSE maintains the shared schema as foundation for all existing view schemas. A user constructs her initial view schema VS by selecting desired base classes from the shared schema and importing them as *ident* virtual classes into VS. Then, the user tailors the view schema according to her special requirements by issuing a sequence of schema change operations upon the view schema (see the user space in Figure 2). Instead of in-place schema changes, TOSE then simulates the schema change requests by a process further explained below [7].

As subschemas (instead of the complete base schema [BKKK87, PS87])

[6] Note that we only allow an attribute domain change to take place if the attribute is locally defined. The *change-attribute-domain* operator can be either capacity-augmenting or capacity-reducing depending on whether the new domain is a supertype or a subtype of the old domain.

[7] That is, a user does not create virtual classes directly, instead virtual classes are created by TOSE.

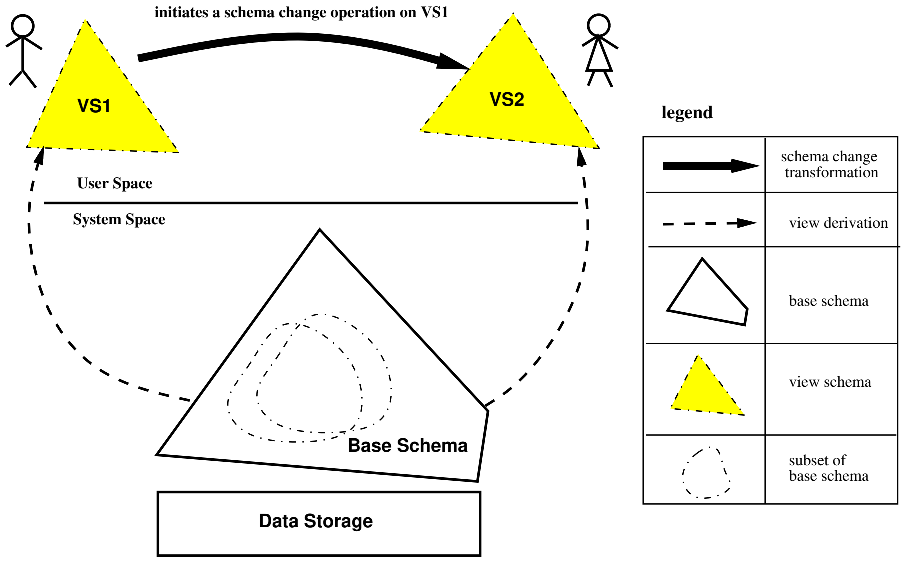
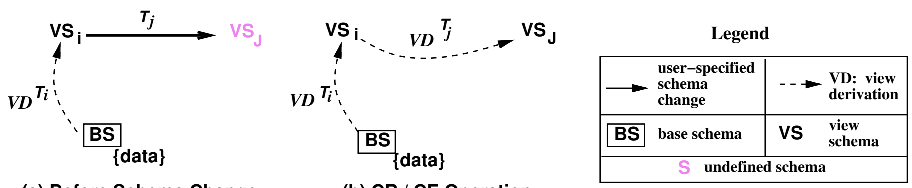


Figure 2: TOSE Simulates Schema Change Using Views.

can be a unit of schema change in TOSE, application developer requesting the change are more likely to be familiar with the to-be-modified schema - making the task easier and less error-prone for the users. Unlike previous research [RR95, RR97, Ber92], TOSE does not rely on the availability of an advanced (i.e., capacity-augmenting) view mechanism, rather a conventional view mechanism for OODBs suffices.

4.1 Capacity-Reducing and Capacity-Equivalent Schema Change Operations

The underlying principle of this proposed mechanism is that capacity-reducing (CR) and capacity-equivalent (CE) schema changes are achieved by constructing a view that reflects the intention of the schema changes (see Figure 3 for an illustration of the TOSE methodology). For a CR or CE schema change request $\mathcal{T}_j$, TOSE is able to compute the desired schema change by creating a new view schema VS_j that reflects the intention of the schema change operation. We have designed a set of virtual class forming queries for all schema change operations supported in TOSE [RRL97] to simulate the schema change effects on views. The set of view forming queries that realizes the schema change request $\mathcal{T}_j$ is called a *view derivation* operation, denoted as $\mathrm{VS}_j = \mathrm{VD}^{\mathcal{T}_j}(\mathrm{VS}_i)$,. Each of the queries maps virtual classes in the input view VS_i to virtual classes in the output view VS_j [8].



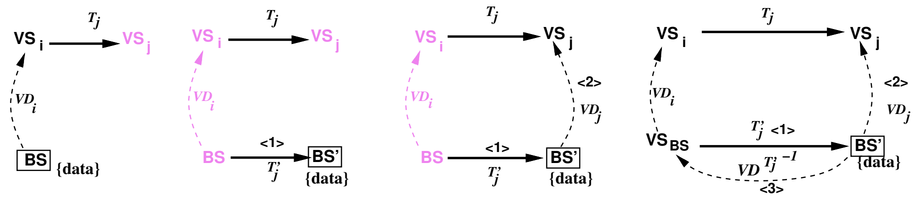
(a) Before Schema Change. (b) CR / CE Operation.

Figure 3: TOSE Methodology for Capacity-Reducing/Equivalent Schema Change Operations.

The old schema VS_i is kept intact for continued support of existing applications running against it, while the new schema VS_j provides a more appropriate interface for new application programs. Furthermore, VS_j can be viewed as if it were (directly) derived from the base BS, i.e., $\mathrm{VS}_j = \mathrm{VD}^{\mathcal{T}_j} \circ \mathrm{VD}^{\mathcal{T}_i}(\mathrm{BS})$ – is the sequence of view derivations that derives VS_j from BS. This assures that all interfaces VS_i and VS_j operate on the same base schema (and data).

4.2 Capacity-Augmenting Schema Change Operations

However, capacity-augmenting (CA) schema changes cannot be supported in this fashion due to the inherent limitation of view mechanisms not being able to augment the capacity of the database. For this, we introduce a new solution based on a three-pronged process that does not rely on the availability of advanced (capacity-augmenting) view mechanisms.



(a) Before Change. (b) Base Augmentation. (c) Target Generation. (d) Base Restoration.

Figure 4: TOSE Methodology for Capacity-Augmenting Schema Change Operations.

First, the base-schema-augmentation step translates the schema change request $\mathcal{T}_j$ specified on the view schema VS_i into a corresponding operation $\mathcal{T}'_j$

[8] When it is clear from the context, subscripts are omitted for succinctness' sake in remainder of this paper.

to be executed on the base schema BS in order to physically restructure BS, denoted by <1> in Figure 4. This is an essential step as schema evolution semantics have not been defined in the literature when applied to virtual classes but only when applied to base classes. After Step 1 is executed, the original base schema BS does no longer exist in its initial form, rather it now has been modified by either augmenting it with information possibly irrelevant to other applications or even by completely modifying the organization of the schema. For this reason, VS specified upon it may have become undefined (BS and VS_i in Figure 4). Furthermore, the data is associated with the newly modified BS' instead of with the old BS.

Second, the target-view-generation step, denoted by <2> in Figure 4, generates the target view schema VS_j as if the schema change operation $\mathcal{T}_j$ had been performed on the view schema VS_i directly (and VS_i were a base schema). Given that the capacity of BS has been appropriately augmented by step 1, we can assure that such a mapping always exists in our context.

Third, since the original BS does not exist any more, for continuing support of all existing view schemas (and thus existing applications) defined on BS, the base-schema-reconstruction step, denoted by <3> in Figure 4, restores the previous base schema BS as a view schema VS_{BS} from the augmented base schema BS'.

5 Update Semantics Preservation (USP) Property of View Schemas

Since the TOSE framework achieves transparency by requiring users to operate on their respective view schemas, we need to assure that the users can work on their view schema as if it were a base schema. For this, we need to make sure that (1) updates on views are unambiguously translated into updates on the base, and (2) the effects of back-propagation from the underlying base schema to the views has the identical effect as if the updates were directly performed on the views. We give the formal definitions of the update semantics of virtual classes, view schemas, and transformations, while proofs can be found in [RRL97, VW94].

Definition 1 *A virtual class VC is* usp, *iff the following generic updates are satisfied at any time t:*

- *create(VC) returns a new object* obj *such that* $Extent_{t+1}(VC) = Extent_t(VC) \cup \{\text{obj}\}$ [9].
- *delete(*obj*), where* obj $\in Extent_t(VC)$, *removes the object* obj *from* $Extent_t(VC)$, *i.e.,* $Extent_{t+1}(VC) = Extent_t(VC) - \{\text{obj}\}$.

[9] Note that $\text{Extent}_t(\text{C})$ is the class extent of the class C at time t.

- *set(*obj, attr, newValue*), where* obj $\in$ *Extent(VC), changes the attribute* attr *of the object* obj *to* newValue.

We have conducted a case study on the set of schema change operations supported in TOSE (Figure 1) [RRL97], and found that only the *hide*, *difference*, *ident*, *union*, and *intersection* object algebra operators are needed to simulate this set of schema changes. We then verified that a usp update propagation behavior could be found for all virtual classes generated by these algebra operators for the three generic update operations *create*, *delete*, and *set* (see our technical report [RRL97]). For brevity, the discussion is not repeated here.

Definition 2 *Let VC_i and VC_j be virtual classes. (VC_i, VC_j) is a usp pair iff we have:*

- *If VC_i is a subclass of VC_j, any* create *operation applied to VC_i at time t adds an object to both $Extent_{t+1}(VC_i)$ and $Extent_{t+1}(VC_j)$, and any* create *operation applied to VC_j at time t does not affect $Extent_{t+1}(VC_i)$. If VC_i is not a subclass of VC_j,* create *applied to VC_i does not affect $Extent_{t+1}(VC_j)$.*
- Delete *applied to object* obj *of VC_i at time t removes the object* obj *from $Extent_{t+1}(VC_i)$, i.e., $Extent_{t+1}(VC_i)$ = $Extent_t(VC_i)$ - {obj}. If* obj $\in$ *$Extent_t(VC_j)$, the object* obj *is also removed from $Extent_{t+1}(VC_j)$; otherwise $Extent_{t+1}(VC_j) = Extent_t(VC_j)$.*
- Set *applied to VC_i at time t changes the attribute* attr *of an object* obj *in both $Extent_{t+1}(VC_i)$ and $Extent_{t+1}(VC_j)$, if* attr *is also defined in VC_j and* obj $\in$ *$Extent_t(VC_j)$; otherwise $Extent_{t+1}(VC_j) = Extent_t(VC_j)$.*

Note that an arbitrary combination of two usp virtual classes is not necessarily a usp pair. This is best explained by the example below.

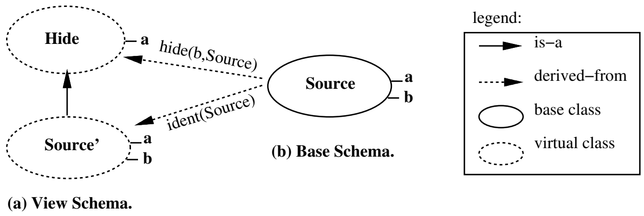


Figure 5: Two Individual Usp Virtual Classes Is not Necessary a Usp Pair.

Example 1 *Let's create two virtual classes, Hide and Source', by applying the* hide *and* ident *object algebra operators to the class Source. The Source' virtual class is an identical copy of its source class, Source, and the Hide virtual class hides the property b from its interface. Therefore, the Hide virtual class is a superclass of the Source' virtual class [Run92]. As mentioned earlier (and shown in [RRL97]) individually* hide *and* ident *virtual classes are usp, i.e., the Hide and Source' classes are usp. However, Hide and Source' together in one schema do not preserve the update semantics, because an object created for the Hide class (a superclass) is not only visible to the Hide class, but also visible to the Source' class (a subclass)* [10]*. This violates the first condition of Definition 2. Therefore, we have shown that (Hide, Source') is not usp, although both of the virtual classes are usp, individually.*

We call this behavior *downward visibility anomaly*, which is not base-like. This unexpected behavior is caused by the view derivation relating two classes, i.e., Hide derived-from Source and Source' derived-from Source, is not hidden from the user. Theorems for usp pair commutativity, associativity, and so on can be found in [RRL97].

Definition 3 *Let a view schema VS consist of* usp *classes* $\{VC_1, VC_2, \ldots, VC_n\}$*. Then VS is called* usp *if every pair of the classes in VS is a usp pair.*

Intuitively, a usp view schema has the following characteristics: let VS be a *usp* view schema and $\mathrm{DB}_t(\mathrm{VS})$ be the state of the stored database under VS at time *t*. Then any update on $\mathrm{DB}_t(\mathrm{VS})$ results in the next snapshot $\mathrm{DB}_{t+1}(\mathrm{VS})$ at time *t+1* that would have resulted if VS were a base schema. In other words, the derivations defining classes in VS don't cause any unexpected side-effects. A usp view derivation is defined next.

Definition 4 *A view derivation* $VD(S_1) = S_2$*, which maps the schema* S_1 *to the schema* S_2*, is called usp if* S_2 *is guaranteed to be a usp view schema whenever* S_1 *is usp.*

Theorem 1 *Let* VD^1 *and* VD^2 *be two usp view derivations. Then, the composition* $VD^2 \circ VD^1(S) = VD^2(VD^1(S))$*, with S any schema and '$\circ$' defined as sequential application, is also usp.*

Again, the proof of the above theorem is omitted due to lack of space [RRL97].

[10] The create operation issued on the Hide class is first translated to the create operation on Hide's base class, which is Source. Then the effect is propagated to *all* the virtual classes derived from it, i.e., Hide and Source'.

6 Formal Foundation of Transparency Property of TOSE

All schema transformations supported in TOSE (i.e., they can be specified and correctly executed over view schemas) have the same semantics as the corresponding conventional schema transformations defined over base schemas. Since TOSE aspires to provide users with an environment of a view schema that 'behaves' exactly like a base schema, we now formalize the *transparency* notion of our TOSE framework. We then show that TOSE achieves schema changes *transparently* for capacity-reducing, -equivalent, and -augmenting transformations.

Definition 5 *Let $\mathcal{T}$ be a schema transformation as defined in Section 3. The inverse transformation of $\mathcal{T}$, denoted as $\mathcal{T}^{-1}$, is defined as $\mathcal{T}^{-1} \circ \mathcal{T}(S) = S$, where S is a schema.*

That is to say applying a transformation and then its inverse (or vice versa) to a schema S results in S.

Definition 6 *Two schemas S_i and S_j are said to be "identical" from time $t = t_k$ onward, iff the same sequence of updates $(\mu_t, \mu_{t+1}, \ldots)$ is applied to both schemas, and we have $\mu_t(DB_t(S_i)) = DB_{t+1}(S_i)$, $\mu_t(DB_t(S_j)) = DB_{t+1}(S_j)$, and $DB_t(S_i) = DB_t(S_j)$ for $t \geq t_k$.*

Definition 7 *Let **VS** be a set of view schemas. Let the input view schema VS_i be usp* [11]*, and let $\mathcal{T}$ be a schema transformation $\mathcal{T}(VS_i) = VS_j$, where VS_i and $VS_j \in$ **VS**. $\mathcal{T}$ is said to be* transparent *iff there exists a* usp *view derivation $VD^{\mathcal{T}}$ such that either $VD^{\mathcal{T}}(VS_i)$ is identical to VS_j (for capacity-reducing/equivalent transformations), or $VD^{\mathcal{T}^{-1}}(VS_j)$ is identical to VS_i (for capacity-augmenting transformations)* [12].

The existence of $VD^{\mathcal{T}}$ or $\mathrm{VD}^{\mathcal{T}^{-1}}$ in the above definition such that $\mathcal{T}$ can be simulated transparently is the *sufficiency condition* for a $\mathcal{T}$ to be considered to be *transparent* within our framework. We have shown [RRL97] that for every capacity-reducing and -equivalent schema change operation supported in TOSE (See Figure 1), denoted by $\mathcal{T}_{re}$ with $\mathcal{T}_{re}(\mathrm{VS}_i) = \mathrm{VS}_j$, there exists a usp view derivation operation $VD^{\mathcal{T}_{re}}$ such that $VD^{\mathcal{T}_{re}}(\mathrm{VS}_i) = \mathrm{VS}_j$. VS_j can be shown to be a usp view schema by Theorem 1, since VS_i and $VD^{\mathcal{T}_{re}}$ are both usp. Hence, TOSE is transparent for $\mathcal{T}_{re}$ by Definition 7.

[11] Assume an initial view schema is constructed by selecting desired base classes from the shared schema – which contains base classes only hence it is usp by definition, then applies one or many usp view derivations to it. The output view schema is guaranteed to be usp by Theorem 1.

[12] $\mathrm{VD}^{\mathcal{T}^{-1}}$ is a usp view derivation corresponding to the inverse transformation $\mathcal{T}^{-1}$.

Next, we focus on showing that TOSE is transparent also for capacity-augmenting transformations $\mathcal{T}_a$. Four of the transformations we are considering (see Figure 1) are capacity-augmenting, namely add-attribute, change-attribute-domain (when the new attribute domain is a superclass of the original domain), add-class, and add-edge. There are two possible scenarios to consider:

Case 1: The capacity that the transformation asks to be added is already in the base schema though it is not necessarily visible through the current user view.

Case 2: The capacity that the user wishes to add is not available in the base schema.

Next, we argue why TOSE is transparent under both scenarios.

Case 1: Case 1 is relevant for the *add-edge* transformations only, since we assume *add-attribute* and *add-class* transformations add attributes and classes that do not exist in the database before. When the requested information is already in the base schema, TOSE is able to derive the target view schema with a usp view derivation. We have developed a general algorithm to simulate the required schema change by determining a usp view derivation that when applied to the base schema will generate the desired (view) schema interface (see [RRL97] for details). The output view schema is usp by Definition 4. This satisfies the transparency requirement of TOSE. ■

Case 2: When the information that the user requests to be added into VS_i does not exist in the base schema BS, the change request $\mathcal{T}_a$ specified on VS_i is translated into an in-place change $\mathcal{T}_a'$ on BS such that $\mathcal{T}_a'$ augments the capacity of BS to become sufficiently large to include the desired information (Figure 4). For each of the capacity-augmenting schema changes listed in Figure 1, we have proposed an algorithm for translating the schema change into a change on the *primary base* class(es) in the base schema [13].

For each of these CA transformations, the application of a usp view derivation VD_j on the modified base schema BS' generates the target (output) schema VS_j (*target view derivation step*). The target view schema VS_j is usp, since it is specified on an (augmented) base schema and the view derivation is capacity-reducing or -equivalent (shown to be usp for our framework). Again, we have given algorithms for how to compute the view derivation VD_j from BS' to VS_j based on the initial view derivation VD_i that was used to derive the input schema VS_i from the original BS for each of the schema changes [RRL97].

Finally the original base BS is reconstructed from the augmented base BS' (*the original base schema restoration step*) by applying a *view derivation operator* $VD^{\mathcal{T}_a'^{-1}}$ that is the inverse of $\mathcal{T}_a'$ to the new base BS'. It is proven [MIR93]

[13] Basically, a primary class is computed by tracing back the derivation history of a virtual class until a base class in found in the base schema. We have given algorithms for how to search the primary base class for each type of the virtual class used in TOSE in our technical report [RRL97].

that for every capacity-augmenting transformation $\mathcal{T}$ there always exists a usp view derivation $VD^{\mathcal{T}^{-1}}$ that can be used to get back the original input schema. We thus have:

$$VS_i \quad = \quad VD_i \circ VD^{\mathcal{T}_a^{'-1}}(BS') \tag{1}$$

With the theorem from [MIR93], Equation (1) naturally captures the third step of our TOSE framework. Because $VD^{\mathcal{T}_a^{'-1}}$ is a usp view derivation [MIR93] and VD$_i$ by assumption of our framework is usp, TOSE is transparent under this second scenario by Theorem 1. ■

7 Walk-through Examples of TOSE

7.1 A Capacity-Reducing Schema Change Operation: Deleting an Attribute

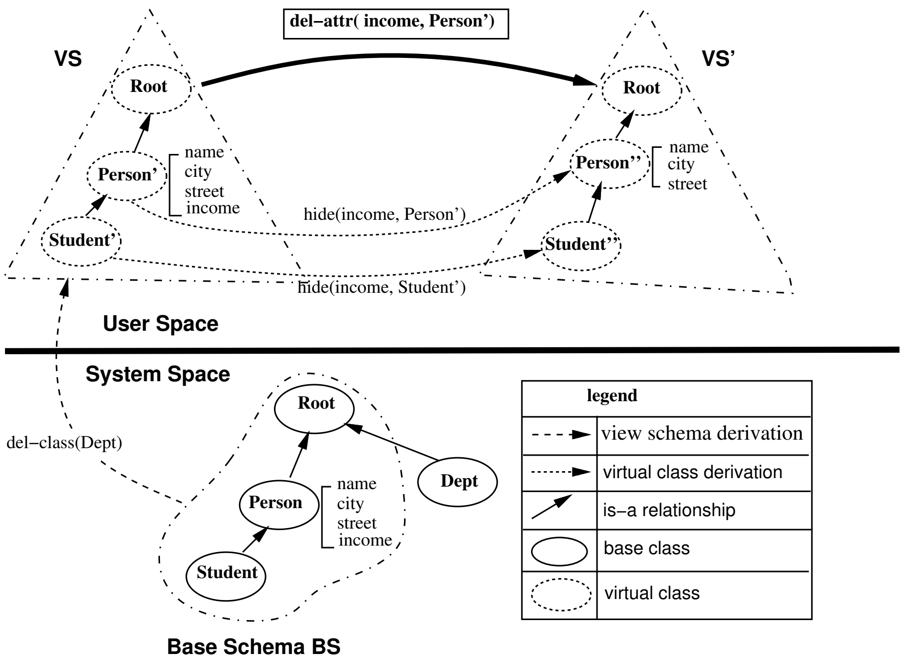


Figure 6: Example: TOSE Simulates a Capacity-Reducing Schema Change Using Views.

A database administer (DBA) may want to remove a sensitive attribute, e.g., *income*, from the database given in the upper left of Figure 6 before allowing access by less privileged users. To do this, the DBA issues the del-attr(*income*,*Person'*) schema evolution request on VS. Because deleting an attribute from a view schema is capacity-reducing, TOSE simulates this simply by a view derivation. Namely, it creates the new view VS' by hiding the attribute *income* from the *Person'* and *Student'* classes, respectively. The newly derived virtual classes are grouped into the new view schema VS' (upper right of Figure 6). Privileged users can still access the attribute *income* through the original view VS while new users now operate through the modified schema VS'.

7.2 A Capacity-Augmenting Schema Change Operation: Adding an Attribute

Now assume a new application needs to know a person's birthday. To satisfy this newly recognized requirement, the DBA issues the *add-attr(birthday,Person')* transformation on the current view VS (user space in Figure 7) [14]. This is a capacity-augmenting change operation.

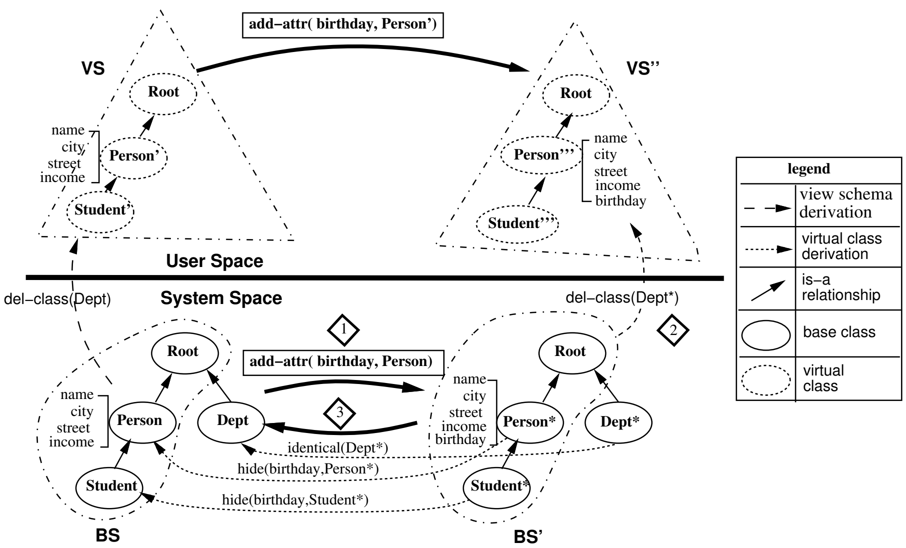


Figure 7: Example: TOSE Realizes a Capacity-Augmenting Schema Change in a Three-Step Process.

First, the *base-schema-augmentation* step translates the *add-attr(birthday,Person')*

[14] Add-attr(a,C) means adding a newly defined attribute a to the class C. Note that this attribute a has to be added to all the subclasses of C, since a virtual class is treated as a first-class citizen in our model [Run92].

originally specified on the view VS into the operation *add-attr(birthday, Person)* on the base schema, where Person is the corresponding primary base class of Person'. TOSE uses a conventional in-place schema change tool to add the attribute *birthday* to Person in BS. The modified class is re-named as Person* for the purpose of easily distinguishing it in our discussion. The *birthday* attribute is inherited into the base class Student as well. After step <1>, the base schema, refined to BS' in Figure 7, has been extended to now hold more capacity. All objects in the database are now associated with the new base BS', instead of the original BS.

Second, the target-view-generation step generates the target view VS'' by applying the same view derivation *del-class*, which was used to derived VS from BS, on the new base schema BS'. Thereafter, we can observe that the resulting target view schema VS'' is identical to the one we expected if we had directly applied the transformation on the input view schema VS.

Third, since the original base schema and the view schema (and thus applications built on it) become undefined, the *base-schema-reconstruction* step restores the previous base schema BS as a view. The inverse function of *add-attr* is the *del-attr* transformation, which is realized in TOSE by applying the *hide* algebra operators to the base classes Person* and Student*, respectively. Note that every physical object instance is associated with one and the same base schema, even though it appears differently to different users depending on the schema through which it is accessed. In the above example, the unique schema under which all instances are maintained used to be *BS* but after the application of the *add-attr* operation (in Figure 7) it has become BS'.

8 Conclusions

In this paper, we have presented a novel solution to the problem of schema evolution affecting existing programs based on a three-step process called TOSE. This TOSE methodology adds great value to many modern applications, as it addresses the pressing demand for application interoperation in a practical manner by utilizing available technology, i.e., a conventional in-place schema change tool and derivation-based view mechanisms, instead of depending on non-established technology. As proof of concept, a TOSE prototype has been built on top of the usp MultiView subsystem [KR96a, KRR95] and the Gemstone OODB [15]. The TOSE system supports all operations listed in Section 7, including capacity-reducing, -equivalent, and -augmenting transformations.

In the future, we plan to study issues related to the optimization of the TOSE implementation, such as integrating deferred evolution of base classes and incremental materialization of virtual classes used by TOSE.

[15] GemStone is a registered trademark of GemStone Inc.

References

[AB91] S. Abiteboul and A. Bonner. Objects and Views. *SIGMOD*, pages 238–247, 1991.

[Ber92] E. Bertino. A View Mechanism for Object-Oriented Databases. In *3rd Int. Conference on Extending Database Technology*, pages 136–151, March 1992.

[BKKK87] J. Banerjee, W. Kim, H. J. Kim, and H. F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD*, pages 311–322, 1987.

[ea97] R. G. G. Cattell et al. In *The Object Database Standard: ODMG 2.0*, chapter 1. Morgan Kaufmann Pub., 1997.

[KC88] W. Kim and H. Chou. Versions of Schema For OODBs. In *Proc. 14th VLDB*, pages 148–159, 1988.

[Kim88] H. J. Kim. *Issues in Object-Oriented Database Systems*. PhD thesis, University of Texas at Austin, May 1988.

[KR95] H. A. Kuno and E. A. Rundensteiner. Materialized object-oriented views in *MultiView*. In *ACM Research Issues in Data Engineering Workshop*, pages 78–85, March 1995.

[KR96a] H. A. Kuno and E. A. Rundensteiner. The *MultiView* OODB View System: Design and Implementation. In Harold Ossher and William Harrison, editors, *Theory and Practice of Object Systems (TAPOS), Special Issue on Subjectivity in Object-Oriented Systems.* John Wiley New York, 1996.

[KR96b] H. A. Kuno and E. A. Rundensteiner. Using Object-Oriented Principles to Optimize Update Propagation to Materialized Views. In *IEEE Int. Conf. on Data Engineering*, pages 310–317, 1996.

[KR98] H. A. Kuno and E. A. Rundensteiner. Incremental Maintenance of Materialized Views in OODBs. *IEEE Transaction on Data and Knowledge Engineering,*, 1998.

[KRR95] H. A. Kuno, Y. G. Ra, and E. A. Rundensteiner. The Object-Slicing Technique: A Flexible Object Representation and Its Evaluation. Technical Report CSE-TR-241-95, University of Michigan, 1995.

[Kun96] H. A. Kuno. *View Materialization Issues in Object-Oriented Databases.* PhD thesis, University of Michigan, Ann Arbor, June 1996.

[Lau97] S.-E. Lautemann. A Propagation Mechanism for Populated Schema Versions. In *IEEE Int. Conf. on Data Engineering*, pages 67–78, 1997.

[MIR93] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *Int. Conference on Very Large Data Bases*, pages 120–133, 1993.

[MNK91] M. A. Morsi, S. B. Navathe, and H. J. Kim. A Schema Management and Prototyping Interface for an Object-Oriented Database Environment. In F. Van Assche, B. Moulin, and C. Rolland, editors, *Object-Oriented Approach in Information Systems*, pages 157–180. Elsevier Science Publishers B. V. (North Holland), 1991.

[MS93] S. Monk and I. Sommerville. Schema Evolution in OODBs Using Class Versioning. In *SIGMOD RECORD, VOL. 22, NO.3*, September 1993.

[PS87] D. J. Penney and J. Stein. Class modification in the GemStone object-oriented DBMS. In *OOPSLA*, pages 111–117, 1987.

[RKR+96] E. A. Rundensteiner, H. A. Kuno, Y.-G. Ra, V. Crestana-Taube, M. C. Jones, and P. J. Marron. Demonstration paper: The MultiView Project: Object-Oriented View Technology and Applications. *SIGMOD*, page 555, 1996.

[RR95] Y. G. Ra and E. A. Rundensteiner. A Transparent OO Schema Change Approach Using View Schema Evolution. In *IEEE Int. Conf. on Data Engineering*, pages 165–172, March 1995.

[RR97] Y. G. Ra and E. A. Rundensteiner. A Transparent Schema Evolution System Based on Object-Oriented View Technology. *IEEE Transactions on Knowledge and Data Engineering*, pages 600–624, September 1997.

[RRL97] Y.G. Ra, E. A. Rundensteiner, and A.J. Lee. A Practical Approach to Transparent Schema Evolution. Technical Report WPI-CS-TR-97-3, Worcester Polytechnic Inst., Dept. of Comp. Scie., 1997.

[Run92] E. A. Rundensteiner. *MultiView*: Methodology for Supporting Multiple Views in Object-Oriented Databases". In *18th VLDB Conference*, pages 187–198, 1992.

[SLT91] M. H. Scholl, C. Laasch, and M. Tresch. Updatable Views in Object-oriented Databases. In *Proceedings of the Second DOOD Conference*, pages 189–207, December 1991.

[SZ86] A. H. Skarra and S. B. Zdonik. The Management of Changing Types in an Object-Oriented Databases. In *Proc. 1st OOPSLA*, pages 483–494, 1986.

[TS93] M. Tresch and M. H. Scholl. Schema Transformation without Database Reorganization. In *SIGMOD RECORD*, pages 21–27, 1993.

[VW94] V.M.P. Vidal and M. Winslett. Perserving Update Semantics In Schema Integration. In *Int. Conference on Information and Knowledge Management*, pages 263–271, 1994.

[Zic91] R. Zicari. A Framework for O_2 Schema Updates. In *7th IEEE Int. Conf. on Data Engineering*, pages 146–182, April 1991.