

SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework *

Kajal T. Claypool, Jing Jin, and Elke A. Rundensteiner
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280
{kajal|jing|rundenst}@cs.wpi.edu

Abstract

With current database technology trends, there is an increasing need to specify and handle complex schema changes. The existing support for schema evolution in current OODB systems is limited to a pre-defined taxonomy of schema evolution operations with fixed semantics. Given the variety of types, complexity, and semantics of transformations, it is sheer impossible to a-priori provide a complete set of all complex changes that are going to meet all user's needs. This paper is the first effort to successfully address this open problem by providing an extensible framework for schema transformations. Our proposed SERF framework succeeds in giving the user the *flexibility* to define the semantics of their choice, the *extensibility* of defining new complex transformations, and the power of *re-using* these transformations through the notion of templates. To verify the feasibility of our SERF approach we have implemented one realization of our concepts in a working prototype system, called OQL-SERF, based on OQL, ODMG MetaData and Java's binding of ODL. We have also conducted a thorough case study demonstrating that our SERF approach can handle all the schema evolution operations we identified from the literature to validate the completeness of our approach.

Keywords: Schema Evolution, Transformation Templates, Object-Oriented Databases, Modeling Database Dynamics, OQL, ODMG, Schema Consistency.

1 Introduction

With current database technology, object-oriented database systems (OODBs) can support very complex object models like the ODMG object model [Cea97]. These complex

object models have paved the road for modelling complex and dynamic applications which by their very nature have frequent schematic changes and upgrades.

The existing support for schema evolution provided by current OODBs [BK87, Tec94, BMO⁺89, Inc93, Obj93] is limited to a *pre-defined* taxonomy of *simple fixed-semantic* schema evolution operations. However, such simple changes, typically to individual types only, are not sufficient for many advanced applications [Bré96]. More radical changes of the schema, such as combining two types or redefining the relationship between two types, are either very difficult or even impossible to achieve with the current commercial database technology [KGBW90, Tec94, BMO⁺89, Inc93, Obj93]. In fact, most OODBs would typically require the user to write ad-hoc programs to accomplish such transformations. In the last two years, research has begun to look into this issue of complex changes [Bré96, Ler96]. However, this new work is again limited by providing a *fixed* set of some selected (even if now more complex) operations.

The provision of any fixed set, may it be simple or complex, is not satisfactory, as it would be very difficult for any one user or system to pre-define all possible semantics and all possible transformations that could ever exist. In fact, it would be sheer impossible to predict all possible transformations a user may desire. This problem exists for simple transformations but becomes even more pronounced for complex transformations. For example, consider the merging of two source classes into one single merge class. Different semantics like union, intersection and difference, can be possible for deciding what properties should be assigned to the merge class. Once, through some mechanism, we have collected this appropriate list of properties, we can then use schema evolution primitives to create the class and add properties to it. There can now be many different semantics for populating the merge class, like a value-based join on some pair of properties, an oid-based join, etc. Similarly, choices exist for the fate of the source classes and the placement of the merge class in the class hierarchy. It can be seen that the more complex the transformation, the more difficult it becomes to finitely predict all possible semantics.

To address this limitation of *current* schema evolution technology, we propose the SERF framework which allows users to perform a wide range of complex user-defined schema transformations *flexibly, easily* and *correctly*. To the best of our knowledge, our work is the very first to address this problem by proposing an extensible schema evolution framework. Our approach is based on the hypothesis that complex schema evolution transformations can be broken down into a sequence of basic evolution primitives, where each basic

*This work was supported in part by the NSF NYI grant #IRI 94-57609. We would also like to thank our industrial sponsors, in particular, IBM for the IBM partnership award and Informix for software contribution. Special thanks also goes to the PSE Team specifically, Gordon Landis, Sam Haradvala, Pat O'Brien and Breman Thuraising at Object Design Inc. for not only software contributions but also for providing us with a customized patch of the PSE Pro2.0 system that exposed schema-related APIs needed to develop our tool.

primitive is an invariant-preserving atomic operation with fixed semantics. In order to effectively combine these primitives and to be able to perform arbitrary transformations on objects within a complex schema operation, we recognize the need of a language that allows such expressibility. Unlike previous research which resorted to the use of a programming language [Tec94, KGBW90] for this purpose, we now propose the use of a standard query language like OQL [Cea97]. OQL has been designed as a declarative language which is powerful yet simple to use and understand. It is a superset of the standard SQL and as such is powerful enough to specify queries that move objects from any one or more types to any different type. And, although OQL has no separate data manipulation support, it is able to invoke operations native to object types such as the schema evolution primitives for schema-type modifications and the standard `get()` and `set()` methods for object manipulations.

In our SERF framework, we go one step further by introducing the new concept of a *template* which makes our transformations generic and thus applicable to any schema, re-usable for building new transformations, and collectable in a template library as resource.

In summary, our proposed framework gives the user:

- The *flexibility* to define the transformation semantics of their choice.
- The *extensibility* of defining new complex transformations meeting specific requirements.
- The *generalization* of these transformations through the notion of templates.
- The *re-useability* of a template from within another template.
- The *ease* of template specification by programmers and non-programmers alike.
- The *soundness* of the transformations in terms of assuring schema consistency.
- The *portability* of these transformations across OODBs as libraries.

We have developed a prototype to test the viability and feasibility of our approach. For the implementation of our prototype, referred to as OQL-SERF, we have chosen to use the ODMG standard as our foundation to assure the wide applicability of our results [CJR98a]. Thus our object model is the ODMG object model and OQL is our transformation language. We use ODI's persistent storage engine (PSE¹) as our persistent system for our OQL-SERF system. We have extended the core functionality provided by PSE by building several tools on top of PSE as needed by SERF, such as an ODMG-compliant Schema Repository, a dynamic Schema Evolution Manager and an OQL Engine [Cea97].

The rest of the paper is organized as follows. Section 2 talks about related research. Section 3 details our framework and Section 4 shows how the consistency of a schema is preserved by a template. In Section 5 we present an implementation of our framework. We conclude in Section 6.

¹PSE is registered trademark of Object Design Inc.

2 Related Work

Schema evolution is a problem that is faced by long-lived data. The goal of schema evolution research is to allow schema evolution mechanisms to change not only the schema but also the underlying objects to have them conform to the modified schema. One key issue in schema evolution is understanding the different ways of changing a schema. The first taxonomy of primitive schema evolution operations was defined by Banerjee et al. [BKKK87]. They defined consistency and correctness of these primitives in the context of the Orion system. Until now, current commercial OODBs such as Itasca [Inc93], GemStone [BMO⁺89], ObjectStore [Obj93], and O₂ [Tec94] all essentially handle a set of evolution primitives similar to Orion's.

In recent years, the advent of more advanced applications has led to the need for support of complex schema evolution operations. [Bré96, Ler96, Cla92] have investigated the issue of more complex operations. [Ler96] has introduced compound type changes in a software environment, i.e., focusing on the type and not on the object instance changes. She provides compound type changes like *Inline*, *Encapsulate*, *Merge*, *Move*, *Duplicate*, *Reverse Link* and *Link Addition*.

[Bré96] proposed a similar list of complex evolution operations for O₂, i.e., now considering both schema as well as object changes. [Bré96] claims that these advanced primitives can be formulated by composing the basic primitives that are provided by the O₂ system. He shows the consistency of these advanced primitives. Like other previous work, the paper however still provides a fixed taxonomy of primitives to the users, instead of giving them the flexibility, extensibility and customization as offered by our approach. Also for object changes, the user is limited to using the object migration functions written in the programming language of O₂.

In summary, all previous research in this area tends to provide the users with a *fixed* set of schema evolution operations [FFM⁺95, BKKK87]. No provision is made for the situation where this does not meet the user's specific needs. How to add extensibility to schema evolution is now the focus of our effort.

Peters and Ozsu [PO95] have introduced a sound and complete axiomatic model that can be used to formalize and compare schema evolution modules of OODBs. This is the first effort in developing a formal basis for schema evolution research, and it may in the future be a foundation based on which to further formalize the concepts introduced in our current paper.

Further research has studied the issue of when and how to modify the database objects to address such concerns as efficiency, availability, and impact on existing code. Research on this issue has focused on providing mechanisms to make data and the system itself more available during the schema evolution process, in particular deferred and immediate propagation strategies [FMZ94b, FMZ94a]. In principle, either of these propagation strategies could be implemented for our framework.

Another important issue focuses on providing support for existing applications that depend on the old schema, when other applications change the shared schema according to their own requirements. Research to address this issue has followed along two possible directions, namely, views [RLR98, RR97, Ber92] and versions [SZ86, Lau97]. Some of this on-going research may need to be re-examined in order to handle the complex notion of transformations as introduced by our templates.

3 The SERF Framework

In this section we present the fundamental principles of our proposed transformation framework. In particular, we will demonstrate how our proposed framework succeeds in giving the user the *flexibility* to define the semantics of their choice, the *extensibility* of defining new complex transformations, and the *re-usability* of these transformations through the notion of templates.

3.1 Features of the Framework

Our proposed framework aims at addressing the limitations of current OODB technology that restrict schema evolution to a *predefined* set of operations with *fixed* semantics. In particular, our goal is to instead support *arbitrary user-customized* and possibly *very complex* schema evolution operations. Similar to [Br 96], our first step in this direction is to allow users to build other schema transformations with different semantics using the set of schema evolution primitives provided by the underlying OODB system.

However, a pure composition of schema evolution primitives is not powerful enough to express a sufficiently large class of transformations. Hence we need to employ some language as “glue logic” and to achieve the value-based transformations of objects across types. In current OODB systems, users have to resort to writing ad-hoc code using a programming language to achieve such transformations. This has the drawback of being both programming language and system dependent and also of not having any guarantee of schema consistency.

In our framework, we therefore advocate the use of a declarative query language like OQL [Cea97] as ideally suited to meet the needs of our transformation language. The query language must have an interface for invoking the schema evolution primitives, as those provide the basic mechanism to syntactically change the type structure. The query language must also have the expressive power for realizing any arbitrary object manipulations to transform objects from one object type to any other object type. In [CJR98b], we have shown that this approach also guarantees consistency for all schema transformations.

In our framework, these arbitrarily complex transformations can be encapsulated and generalized by assigning a name and a set of parameters to them. From here on these are called *transformation templates* or *templates* for short. By parameterizing the variables involved in a transformation such as the input and the output classes and their properties, a transformation becomes a *generalized reusable* module applicable to any application schema. By assigning a name to such a template, it can also now be *re-used* from within other transformations. This leads us to the idea of collecting these templates in a *template library* and thus guaranteeing the availability of these templates to any user at any time, just as the fixed set of schema evolution operations are available to the users in any regular schema evolution system.

In summary, a schema transformation in our framework lets the user combine an object transformation language with a standard set of schema evolution primitives to produce arbitrarily complex transformations. Moreover, these transformations are generalized and stored in a standard library for later re-use. Transformations in this general form are called *templates* in the framework and the library, the *template library*.

3.2 System Architecture

Figure 1 gives the general architecture of our proposed framework. As we will explain further, the components listed on the top half of the figure make up the framework and thus are to be provided by any implementation realizing our framework. The components listed below the line represent system components that we expect any underlying OODB system to provide.

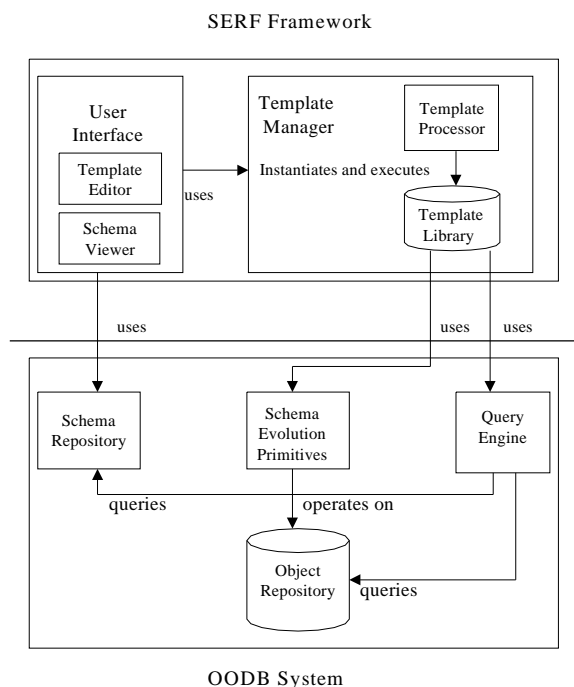


Figure 1: Architecture of the SERF Framework

Figure 1 also shows the flow of the composition of a template. In general, a template² uses a query language to query over the schema repository, i.e., the metadata³, and the application objects. The template also uses the query language to invoke the schema evolution primitives for modifying the schema types, and system-defined functions for updating the object instances. Although any query language could be used for this, from here on we will talk about OQL as our query language, since it is part of the ODMG standard and is perfectly suited for our needs as we will show below.

3.3 System Requirements

For our framework, we assume that the underlying OODB system provides us the following components:

- **Schema repository.** For a transformation (as shown in Figure 3) to be generalizable to a template (as shown in Figure 4) we need to be able to query and access the metadata in some manner. For example, to merge two source classes into a single merge class by doing a union of the properties of the source classes, we need to get the properties of the source classes and add them

²Although we distinguish between a transformation and a template, unless explicitly stated we use the term template to refer to both.

³More details on the schema repository are presented in Section 3.3.

to the merge class. In general to be able to get this information in a template, our framework requires access to the metadata. Most OODB systems indeed do allow access to the meta data, i.e., the data dictionary, and in most cases this is via some high-level declarative interface like a query language instead of just a procedural low-level API.

- **Taxonomy of schema evolution primitives.** The OODB needs to provide a set of schema evolution primitives that is complete⁴ and consistent [BKkk87]. Most OODB systems provide a taxonomy of schema evolution primitives and have invariants defined for preserving the consistency of the schema graph. The set of schema evolution primitives used is dependent on the underlying object model [BKkk87, Tec94].
- **Query language.** A transformation is a sequence of statements that gather metadata information, execute schema evolution primitives and invoke system-defined methods for object manipulations. There is therefore a need for a uniform language to access, query and modify both the metadata information and the application objects. For the SERF framework, we propose the use of a *declarative language*, namely the query language of the OODB system itself, to accomplish this task, thus requiring the query language to have the expressibility power to do all of the above. In particular, for the SERF Framework we require the query language to provide:
 - simple-to-use access to the OODB system,
 - constructs to invoke the schema evolution primitives,
 - support for creating, deleting and modifying objects either through some built-in features as in SQL or through the invocation of system-defined update methods, and
 - support for universal and existential quantification.

3.4 Schema Transformations

A schema transformation combines a query language with a standard set of schema evolution primitives to produce other more complex transformations. It can be used to express different semantics for primitives as well as to create new schema evolution operations.

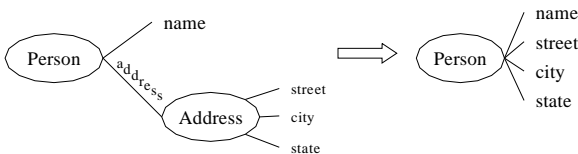


Figure 2: Example of the Inline Transformation.

For example, Figure 3 illustrates the transformation for *inlining*. *Inlining* is defined as the replacement of a referenced type with its type definition [Ler96]. For example in Figure 2 the **Address** type is inlined into the **Person** class, where all the attributes defined for the **Address** type (the

⁴By *complete* we mean a complete set of schema evolution primitives as defined by Banerjee et al. [BKkk87] in terms of achieving all possible basic types of schema graph manipulations.

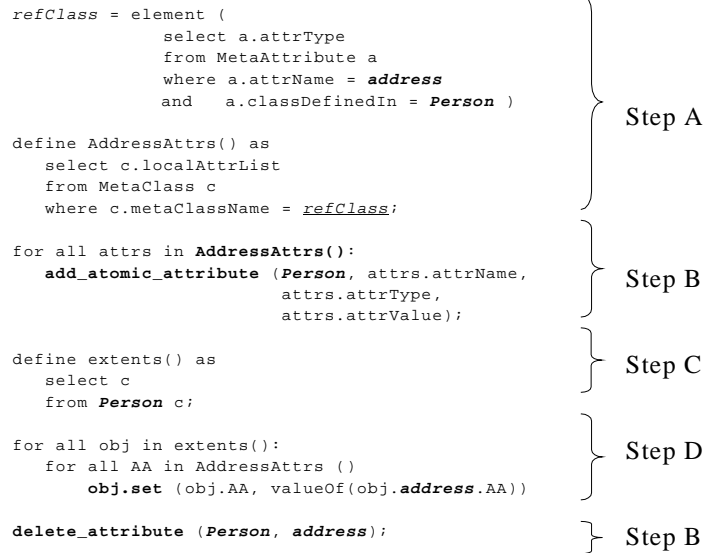


Figure 3: SERF Representation of the Inline Transformation using OQL

referenced type) are now added to the **Person** type resulting in a more complex **Person** class.

We claim that in general a transformation accomplishes this using the following four key steps⁵:

- **Step A: Query the MetaData.** To make a transformation general and re-usable for any possible schema in the form of a transformation template, it is necessary that a user be able to query the metadata using a query language. This information can then be used to make decisions about changes to the schema. In Figure 3 this step denoted by **Step A** which retrieves all the objects from the metadata that models the properties of the **Address** object.
- **Step B: Change the Schema.** We require that all structural changes, i.e., changes to the schema, are exclusively made through the schema evolution primitives. This helps us in guaranteeing the schema consistency after the application of a transformation [CJR98b]. The information gathered in *Step A* can provide the metadata to be changed as well as provide information needed for determining how to change the metadata, both serving as input to these SE primitives. For example, **Step B** in Figure 3 shows the addition of extra attributes through the *add_attribute* primitive to the **Person** class.
- **Step C: Query the Objects.** As a preliminary to performing object transformations, we need to obtain the handle for objects involved in the transformation process. This may be objects from which we copy object values (e.g., **Address** objects in **Step C**), or objects that get modified themselves (e.g., **Person** objects in **Step D**).

⁵Note that each of these four steps are not pure but can often be composed of or inter-mingled with the other steps. For example, in **Step D** also involves the query of objects. We use the four key steps to denote the primary functionality of the step.

- **Step D: Change the Objects.** The next step to any schema transformation logically is the transformation of the objects to conform to the new schema. Through **Step C**, we already have a handle to the affected object set. **Step D** in Figure 3 shows how a query language like OQL and system-defined update methods, like *obj.set(...)*, can be used to perform object transformations.

In general, a transformation in our SERF framework uses a query language to query over the schema repository, i.e., the metadata and the application objects, as in **Steps A** and **C**. The transformation also uses the query language to invoke the schema evolution primitives for schema structure changes and the system-defined functions for updating the objects, as in **Steps B** and **D**.

In this example, we simply wanted to add attribute values to the existing objects. In some transformations, existing objects might need to be deleted or new objects might be created. OQL allows for the creation of new objects through a constructor-like statement. Deletion of objects has to be done by some system-defined methods.

3.5 Transformation Templates

In Figure 3 in the previous section, we have shown how a schema transformation can be written for inlining the **Address** class into the **Person** class using OQL, schema primitives, and system-defined object manipulating methods. It can be observed that this is a useful transformation that should be generalized so that it can now be applied to inline any Class A into any Class B. We achieve this by parameterizing the transformation and refer to this generalized schema transformation as a *template* in our framework.

```

begin template inline (className, refAttrName)
{
  refClass = element (
    select a.attrType
    from MetaAttribute a
    where a.attrName = $refAttrName
    and a.classDefinedIn = $className; )

  define localAttrs(cName) as
  select c.localAttrList
  from MetaClass c
  where c.metaClassName = cName;

  // get all attributes in refAttrName and add to className
  for all attrs in localAttrs(refClass)
    add_atomic_attribute ($className, attrs.attrName,
      attrs.attrType, attrs.attrValue);

  // get all the extent
  define extents(cName) as
  select c
  from cName c;

  // set: className.Attr = className.refAttrName.Attr
  for all obj in extents($className):
    for all Attr in localAttrs(refClass)
      obj.set (obj.Attr, valueOf(obj.refAttrName.Attr))

  delete_attribute ($className, $refAttrName);
}
end template

```

Legend: **cName**: OQL variables
\$className: template variables
refClass: user variables

Figure 4: Genralized Inline Template based on the Inline Transformation

Figure 4 shows the inline transformation in the form of a template. A template is a *named*, *parameterized* and *atomic*

schema transformation. The inline template takes two parameters *className* and *refAttrName*. The *className* is the referring class and the *refAttrName* refers to the class that is to be inlined. These variables are now used instead of the **Person** class and the **address** attribute. The template can now be instantiated by binding the formal parameters to the actual parameters **Person** and **address**. This instantiation of the template **inline (Person, address)** would result in the transformation shown in Figure 2.

A SERF template is thus a named sequence of OQL statements extended with parametrization that can be translated down to pure OQL statements during the process of instantiation.

The BNF for a SERF template ⁶ is given as:

```

template ::= begin - template template_name
          ([parameter]*)
          template_statements
          end - template;
template_statements ::= template_statement |
                       template_statement
                       template_statements
template_statement ::= define_query | query
define_query ::= define identifier as query
query ::= query
           restricted_query
restricted_query ::= query([function]*) |
                  λ
function ::= system_function(basic_query*) |
            schema_primitive(parameter*)
parameter ::= string_literal parameter |
              string_literal
basic_query ::= nil | true | false | literal

```

During the instantiation process the SERF template specification is translated to pure OQL statements. The SERF template BNF restricts the query language to only invoke a limited set of operations and functions. This set of operations includes the schema evolution primitives and some system-defined operations for setting the values of objects. While this may limit the ability of users to employ some powerful foreign functions ⁷, it is essential for assuring the well-foundedness of our template framework. In particular, it assures some level of portability of our templates if method calls are restricted to a standard generic set of object methods, such as, *setValue* or *simply set*.

Secondly, this also implies that no method other than a schema evolution primitive can change the structure of the application schema. This allows us to perform the consistency and bind checks both before and after the instantiation of the template. Figure 5 shows the steps for the execution of a template. When a template is created, the user can assign a name to it and also specify its parameters. A syntax check is performed for both the OQL queries and the *system_function* in the template. At the time of instantiation of the template, the user needs to provide the bindings for the parameters. A second syntax check at this point

⁶In the BNF *restricted_query* denotes the OQL query limited to invoking only object updates and the schema evolution primitives

⁷A given realization of our framework could relax this restriction by carefully permitting the use of some foreign functions within the constraints of the well- foundedness of our framework

validates the bindings and the template is then executed.

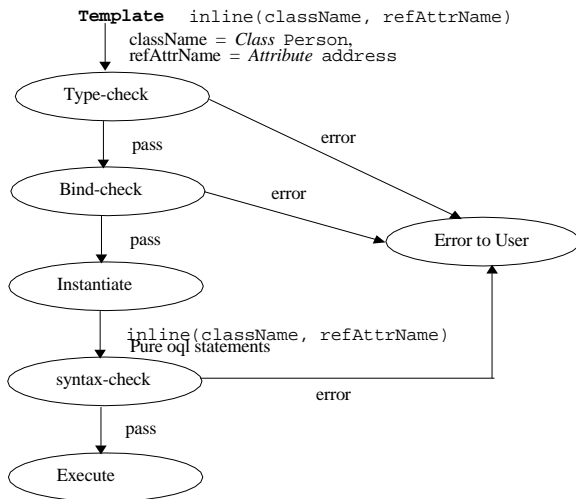


Figure 5: Steps for the Execution of a Template

In summary, the templates provide users not only with the advantages achieved by our schema transformations, i.e. a user can specify their own semantics for transformations, but also allows *reusability* of these schema transformations by parameterizing them. This thus makes the templates reusable and applicable to any application schema. Moreover, our SERF template BNF allows us to guarantee the consistency of the schema after the application of our schema transformations (For more details see [CJR98b]).

Further we propose the development of a library of such templates. We can envision that a template library could become an important resource in the schema evolution community much like the standard libraries in the programming environment.

4 ODMG Standard for SERF

In this section we give a brief description of the ODMG Standard and show how the ODMG Standard meets the system requirements as listed in Section 3.3.

4.1 ODMG Standard

The ODMG standard is based on the continuing work for OODB systems undertaken by the members of the Object Database Management Group (ODMG). The major components of the ODMG standard as applied to the SERF framework are described in the subsequent sections.

4.1.1 ODMG Object Model

The ODMG Object Model is based on the OMG Object Model for object request brokers, object databases and object programming languages [Cea97, Clu98]. For the purpose of the SERF framework we limit our description of the ODMG Object Model to the Java's binding of the object model.

Types, Objects and Literals. The basic modeling primitives for an ODMG compliant database are *objects* and *literals* or *immutable objects* which are categorized by their *types* implying that there are *object types* and *literal types*.

Each object has a unique object identifier which persists through the lifetime of the object and serves as a means of reference for other objects. Literals, on the other hand, do not have object identifiers and a change to a literal results in a new literal. An object can optionally also have one or more user-assigned name(s) and this is termed as *persistence by reachability*.

Inheritance. Although ODMG defines multiple inheritance, Java's binding of ODMG Model supports only single inheritance. A subclass, therefore inherits the range of states and behavior from its superclass. Moreover, an object can be considered as an instance of its class as well as its superclass.

Extent of Types Although Java's binding of the ODMG model does not as yet support the notion of extents, we have found it to be a necessary extension to the binding.

ODMG Schema A schema is composed of a set of object and literal type definitions, a class hierarchy and a set of named objects.

4.1.2 ODMG Schema Repository

MetaData is descriptive information that defines the schema of a database. It is used by the OODB system at initialization time to define the structure of the database and at run-time to guide its access to the database. MetaData is stored in a *Schema Repository*, which is also accessible to tools and applications using the same operations that apply to user-defined types, like OQL. ODMG defines a Schema Repository which is stored in the form of *meta-objects* interconnected by relationships that define the schema graph.

4.1.3 ODMG's Object Query Language - OQL

As part of its standard, ODMG has defined an object query language OQL which supports the ODMG data model. OQL is similar in format and features to SQL 92 but has extensions for some object-oriented notions like complex objects, object identity, path expressions, polymorphism, operation invocation and late binding. In this section we describe a small subset of the language that is used for the examples in the paper. For a complete description of OQL the reader is referred to [Cea97].

Selection. As a stand-alone query language, OQL supports the querying over *any* kind of object (i.e., individual object instances, collections and even the metadata in the schema repository) starting from their names which act as entry points to the database. OQL supports querying with and without object identifiers.

Creation. OQL supports the creation of objects both with and without identity.

Path Expressions. ODMG as mentioned in Section 4.1.1 supports the naming of objects and also the reachability of other objects through this named object (i.e., persistence by reachability).

Method Invocation. OQL can call a method with or without parameters anywhere the result type of the method matches the expected type in the query. And although OQL does not have any explicit support for updating the objects, for example the capability to invoke methods allows a user to invoke application-specific update methods through the query language.

4.2 Why ODMG?

The analysis of the requirements for the underlying OODB system revealed the ODMG standard to be a perfect fit for the SERF Framework.

Object Model. The ODMG object model encompasses the most commonly used object models and standardizes the features into its own object model, thus increasing the portability and applicability of our prototype.

Schema Evolution Primitives. Moreover, the most commonly found taxonomy of schema evolution primitives (as found in commercial databases [Tec94, Tec92, BMO⁺89, Obj93, BKkk87, Inc93]) are directly applicable to the ODMG object model.

Schema Repository. ODMG also defines metadata and MetaObject Protocols in the shape of the ODL Schema Repository and explicitly states that this Repository should be accessible to tools and applications using the same operations that apply to the user-defined types [Cea97].

Query Language. ODMG also defines a query language, OQL, as part of its standard that is a superset of SQL-92 in terms of its querying capabilities. In addition OQL also provides programming-language-like constructs such as *for loop*, *iterators* etc..

And, although OQL does not have any built in features for updating objects, it has the capability to invoke system-defined functions. This implies that it can update objects through system-defined set methods and also it can invoke schema evolution primitives. OQL thus meets all the requirements of a query language as set forth by the SERF framework.

We have done an extensive case study of the different types of schema evolution operations found in literature and have found OQL to be sufficient in expressing all of the transformations [Jin98].

5 Conclusions

Summary In this paper we have presented an extensible schema evolution framework for OODBs to address the limitations of a fixed set of schema evolution operations. We offer re-use of these transformations through the key concept of this framework that is a *template*. The transformation templates are written using a query language, like OQL, and use the schema evolution primitives and the meta-knowledge provided by the underlying system. They are a generic, named and parameterized module. We believe the notion of transformation templates is a very powerful idea and to our knowledge this is the first time it has been introduced in the schema evolution domain. Users of any specific OODB will no longer be limited to the schema evolution primitives that are offered by the OODB. Instead, a user

will be able to shop around for the best schema evolution template library for their own use.

Implementation Status We are in the process of developing a prototype of the SERF Framework - OQL-SERF. OQL-SERF is built using Object Design Inc.'s Persistent Storage Engine Pro 2.0 (PSE Pro 2.0) as the underlying OODB system. It is based on the ODMG standard and is written in 100% Pure Java. In particular, we have used an extension of Java's binding of the ODMG object model and we have built our own binding of the ODMG Schema Repository using Java [CJR98a].

We have built a Schema Repository consistent with the ODMG specification, and a Schema Evolution facility for PSE Pro 2.0. We are currently in process of developing a fully operational OQL Engine for PSE Pro 2.0. The Framework modules, i.e., the Template Module and the User Interface are under development as well. For more implementation details refer [CJR98a]. We hope to have a functional OQL-SERF by the end of the year.

Contributions In summary, the main contributions of this work are:

- Identification of problem - we have identified the limitation of the current OODBs in terms of schema evolution support.
- Solution - as a solution we have presented the novel idea of templates. The concept of templates exists in the programming language domain. To the best of our knowledge we are the first to introduce and adapt this in the context of schema evolution.
- Consistency - in addition we have shown that templates can preserve the structural consistency of the schema, if the underlying schema evolution primitives are invariant preserving [CJR98b].
- Implementation - we have partially implemented a prototype to show the feasibility of our approach and in the process have come up with our own minimal and complete taxonomy of schema evolution primitives [CJR98a].
- Evaluation - we have verified that our proposed notion of transformation templates is powerful by a thorough case study. Due to space restrictions, this case study is not included here but can be found in [Jin98].

We are in the process of developing our prototype. In the future, we plan to develop tools that assist with the maintainance, search and consistency checking of templates.

Acknowledgments. The authors would like to thank students at the Database Systems Research Group at WPI for their interactions and feedback on this research. In particular, we are grateful to Anuja Gokhale, Parag Mahalley, Swathi Subramanian, Jayesh Govindrajana, Stacia De Lima, Stacia Weiner, Xin Zhang and Ming Li for their input on the implementation of OQL-SERF.

References

- [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.
- [BMO⁺89] R. Bretl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, and M. Williams. The GemStone Data Management System. In *Object-Oriented Concepts, Databases and Applications*, pages 283–308. ACM Press, 1989.
- [Bré96] P. Bréche. Advanced Primitives for Changing Schemas of Object Databases. In *Conference on Advanced Information Systems Engineering*, pages 476–495, 1996.
- [Cea97] R.G.G Cattell and et al. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann Publishers, Inc., 1997.
- [CJR98a] K.T. Claypool, J. Jin, and E.A. Rundensteiner. OQL_{SERF}: An ODMG Implementation of the Template-Based Schema Evolution Framework. Technical Report WPI-CS-TR-98-14, Worcester Polytechnic Institute, July 1998.
- [CJR98b] K.T. Claypool, J. Jin, and E.A. Rundensteiner. SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework. Technical Report WPI-CS-TR-98-9, Worcester Polytechnic Institute, May 1998.
- [Cla92] S.M. Clamen. Type Evolution and Instance Adaptation. Technical Report CMU-CS-92-133R, Carnegie Mellon University, School of Computer Science, 1992.
- [Clu98] S. Cluet. Designing OQL: Allowing objects to be queried. *Journal of Information Systems*, 23(5):279–305, 1998.
- [FFM⁺95] F. Ferrandina, G. Ferran, T. Meyer, J. Madec, and R. Zicari. Schema and Database Evolution in the O₂ Object Database System. In *Int. Conference on Very Large Data Bases*, 1995.
- [FMZ94a] F. Ferrandina, T. Meyer, and R. Zicari. Correctness of Lazy Database Updates for an Object Database System. In *Proc. of the 6th Int'l Workshop on Persistent Object Systems*, 1994.
- [FMZ94b] F. Ferrandina, T. Meyer, and R. Zicari. Implementing Lazy Database Updates for an Object Database System. In *Proc. of the 20th Int'l Conf. on Very Large Databases*, pages 261–272, 1994.
- [Inc93] Itasca Systems Inc. Itasca Systems Technical Report. Technical Report TM-92-001, OODBMS Feature Checklist. Rev 1.1, Itasca Systems, Inc., December 1993.
- [Jin98] J. Jin. An Extensible Schema Evolution Framework for Object-Oriented Databases using OQL. Master's thesis, Worcester Polytechnic Institute, May 1998.
- [KGBW90] W. Kim, J. F. Garza, N. Ballou, and D. Woelk. Architecture of the ORION Next-Generation Database System. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.
- [Lau97] S.-E. Lautemann. Schema Versions in Object-Oriented Database Systems. In *Int. Conference on Database Systems for Advanced Applications (DASFAA)*, pages 323–332, 1997.
- [Ler96] B.S. Lerner. A Model for Compound Type Changes Encountered in Schema Evolution. Technical Report UM-CS-96-044, University of Massachusetts, Amherst, Computer Science Department, 1996.
- [Obj93] Object Design Inc. *ObjectStore - User Guide: DML. ObjectStore Release 3.0 for UNIX Systems*. Object Design Inc., December 1993.
- [PO95] R.J. Peters and M.T. Ozsü. Axiomatization of Dynamic Schema Evolution in Objectbases. In *IEEE Int. Conf. on Data Engineering*, pages 156–164, 1995.
- [RLR98] E.A. Rundensteiner, A. Lee, and Y.-G. Ra. Capacity-Augmenting Schema Changes on Object-Oriented Databases: Towards Increased Interoperability. In *Object-Oriented Information Systems*, 1998.
- [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*, 9(4):600–624, September 1997.
- [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.
- [Tec92] Versant Object Technology. *Versant User Manual*. Versant Object Technology, 1992.
- [Tec94] O₂ Technology. *O₂ Reference Manual, Version 4.5, Release November 1994*. O₂ Technology, Versailles, France, November 1994.