Schema Evolution for Object-Based Accounting Database Systems

Jia-Lin Chen and Dennis McLeod
Department of Computer Science
and
Daniel O'Leary
School of Business

University of Southern California
Los Angeles, CA 90089, USA

Abstract When an (accounting) database schema does not meet the requirements of a firm, the schema must be changed. Such schema evolution can be considered as realizable via a sequence of operators. This research proceeds in the following three steps. First, we define a set of basic evolution schema operators and employ the evolution heuristics to guide the evolution process. Second, we explore how domain-specific knowledge can be used to guide the use of evolution operators to complete the evolution task. A well-known accounting data model is used here to guide the schema evolution process. Third, we discuss a tool built to implement the evolution operators, using the evolution heuristics and domain-specific knowledge.

1. Introduction

1.1 Motivation

The static meta-data view of database management is that the schema of a database is designed before the database is populated and remains relatively fixed over the life cycle of the system. However, the need to support database evolution is clear: a static meta-data view of a database can support neither next generation dynamic database applications such as interactive multi-media information systems [5] nor traditional database applications such as accounting information systems.

There are at least two reasons that a database schema would need to change. First, the current schema may not meet the original requirements. Such a schema may be called a "premature schema," resulting from erroneous schema design or incomplete requirement analysis. Second, the current schema may not meet new requirements. This type of database schema may be termed an "obsolete schema"; such obsolete schema may be caused by changes in the real world and/or changes of users' views or perceptions thereof.

In a traditional setting, a database administrator (DBA) and programmers would spend substantial time to correct a premature schema and update an obsolete schema even after the database has been populated. However, with the proliferation of databases within organizations, end-users today often act as DBAs. For example, often an accounting database may be directly maintained by its users (accounting specialists or clerks). Unfortunately, these users may lack the database knowledge and programming skills required to change the database schema.

1.2 Research Approach

When an accounting database schema does not meet the requirements of a firm, the schema must be changed. One important issue is how the data can be adapted to a new schema. A classical way to deal with this is to write a conversion program to manipulate the data to fit the new schema. An alternative approach is to develop a set of evolution operators to handle the data adaptation. Database schema evolution can be considered as realizable via a sequence of operators. These schema evolution operators manipulate the original schema into a new schema, and the populated database is modified accordingly. We can consider two key questions here:

- Can we find a set of schema evolution operators that can be effectively used by an end user?
- What heuristics are necessary to guide a user in the choice of a sequence of operators to complete a given evolution task?

This research addresses the above two questions in the following three steps. First, we define a set of basic schema evolution operators and employee evolution heuristics to guide evolution process. Second, we explore how domain-specific knowledge can be used to guide the evolution operators to complete tasks. The REA accounting data model [9] is used here to guide the schema evolution process of an object-based accounting database system. Third, we discuss a tool built to implement the evolution operators, using the evolution heuristics and domain-specific knowledge. The tool provides a user-friendly interface to guide a non-expert user to complete evolution tasks.

This paper is organized as follows. Section 2 provides background on related research and introduces an object-based data model and its schema evolution operators. Section 3 introduces an object-based REA accounting model and discusses how the REA model is used to guide schema evolution. Section 4 describes the architecture and implementation of a schema evolution administration tool, REAtool, and shows the look and feel of its prototype. The last section, Section 5, summaries this research, and discusses some future research directions.

2. Background

2.1 Related Research

ORION [2], ENCORE [13], and GemStone [12] use object-oriented data models and support evolution mechanisms; ORION and ENCORE employ a screening approach and Gemstone uses a conversion approach. They define modeling invariants and rules as schema evolution constraints. The semantics of schema evolution operators is used

to maintain the evolution constraints. PKM [8] identifies a rich set of evolution patterns that can be used in a conceptual evolution process. OSAM* Schema Tailoring Tool [11] is based on OSAM* data model [14] and allows a non-expert user to redesign an OSAM* schema by tailoring its old schema. The tailoring process is accomplished through evolution operations. These operations maintain the schema constraints. If the modeling constraints are violated, the operations will be aborted.

Thus, several researchers have formulated schema evolution constraints as invariants and rules of object-oriented data models. The semantics of a set of evolution operators are then defined, based on the evolution constraints they proposed. However, their research did not employ heuristics or domain knowledge to guide an end-user in completing evolution tasks. In the research described in this paper, we indeed employ heuristics and domain knowledge to structure the evolution process.

2.2. Object-Based Data Model and Schema Evolution Operators

A basic Object-Based Data Model (OBDM) is used here. Modeling constructs of OBDM such as class, class hierarchy, attribute, inheritance, and their associated constraints can be found in the literature of object-based data models [1, 4, 7].

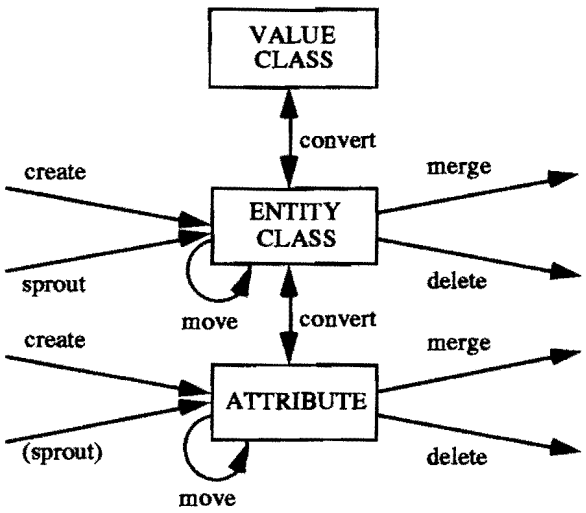

Figure 2.1 Schema Evolution Operators

Based on OBDM, we define four groups of schema evolution operators: 1) Schema Enhancement Operators **create** and **sprout**; 2) Schema Reduction Operators **merge** and **delete**; 3) Schema Restructure Operator **move**; and 4) Schema Conversion Operator **convert**. (See Figure 2.1) The operator **create** creates a new class or attribute. The new class can be a generalized class of several classes or a specialization of some class. The operator **sprout** generates a new class with its instances having a one-to-one mapping to its source class. The operator **merge** merges a class into

another class or an attribute into another attribute. The operator **merge** deletes the meta-data such as classes or attributes, but keeps the data unchanged. The operator **delete** will delete data as well as meta-data. The operator **move** changes the structure of class hierarchy by moving classes or by moving attributes. The operator **convert** converts a modeling construct among a value class, an entity class, and an attribute.

These schema evolution operators should obey modeling constraints to keep a schema consistent after they have been applied. This is the consistency principle of schema evolution. For example, the instances of a subclass should be the instances of its superclass. To keep database consistent in schema evolution, evolution operators will be ruled by modeling constraints to propagate the changes to related parts of a schema. This is called a propagation effect. Since a propagation effect could change the data and meta-data of a schema that a user does not intend to change, a propagation effect must be controlled to maintain the losslessness of the data and meta-data. This is called the preservation principle of schema evolution. For example, when a subclass is merged into its superclass, its attributes and the values defined by these attributes could be lost. One way to avoid this kind of loss of meta-data and data is to move these attributes of the subclass to a superclass. The preservation principle is realized by using a set of evolution heuristics to guide an evolution process. The details of these evolution heuristics are described in [3].

3. Schema Evolution Guided by Domain Knowledge

3.1 Object-Based REA Accounting Model

Domain-specific knowledge can be used to guide a non-expert user to conduct schema evolution tasks. In particular, this research explores an well-known accounting model to guide the schema evolution in the context of accounting information systems.

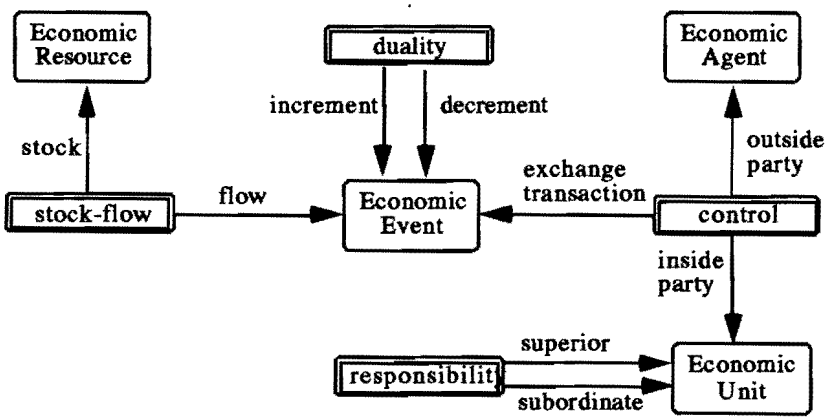

Figure 3.1 The Object-Based REA Accounting Model

The REA accounting model is a generalized accounting framework to capture the interaction of economic resources, economic events and economic agents for

accounting systems [9]. Economic resources are scarce assets such as inventory or cash under the control of an enterprise. Economic events are phenomena that reflect changes in economic resources resulting from production, exchange, consumption, and distribution. Purchase and cash disbursement are the examples of economic events. Economic agents are persons and parties who participate in the economic events, e.g. vendor. Economic units are a subset of economic agents and are inside participants, e.g. cashier and buyer. (See Figure 3.1)

Their are four types of relationships between these REA entities:

1) Stock-flow relationship: This relationship is used to connect an economic resource and an economic event. The stock part of the relationship is an economic resource; and the flow part of the relationship is an economic event. For example, the stock-flow relationship between Inventory and Purchase has Inventory as its stock part and Purchase as its flow part.
2) Duality relationship: A duality relationship links two events. One event is an increment part of the relationship and the other corresponding event would be a decrement part of the relationship. For example, Purchase Payment is a duality that links the event, Purchase, as its increment part and the event, Cash Disbursement, as its decrement part.
3) Control relationship: A control relationship is a three-way association among an economic event (as exchange transaction part), an economic agent (as outside party), and an economic unit (as inside party). For example, Purchase Supply is a control relationship that associates Purchase (an event with a role as its exchange transaction part), Vendor (an agent with a role as its outside party), and Buyer (a unit with a role as its inside party).
4) Responsibility relationship: This relationship indicates one economic unit as its superior part and the other economic unit as its subordinate part. For example, the relationship "works for" is a responsibility that has Cashier as its subordinate part and Treasurer Department as its superior part.

REA model was originally described in an entity-relationship representation. Since this paper employs an object-based approach, the REA entities are modeled as classes and their relationships are modeled as associative classes in an object-based model. Since the evolution operators discussed here are based on the object-based data model, the object-based REA model will be used directly to guide the use of these evolution operators.

3.2 REA Guidance and Schema Evolution

The REA accounting model is used to guide schema evolution of an object-based accounting database. From the viewpoint of an accounting database schema, REA classes are meta-classes. A class of an accounting database are an instance of one of classes of the REA model. For example, the class Purchase Payment is an instance of the meta-class Duality. A class in an accounting database is said to be REA-compliant if and only if:

- The class is an instance of one of REA meta-classes;
- It inherits all the attributes from this REA meta-class; and
- The values of these attributes of the class are defined.

For example, Class **Purchase Payment** is defined as an instance of REA meta-class **Duality**. The class inherits the attributes, **increment** and **decrement**, from the meta-class **Duality**. Furthermore, its increment part is defined as **Purchase** and its decrement part is defined as **Cash Disbursement**. Both of them are events. Hence, Class **Purchase Payment** is REA-compliant. While all classes of an accounting database are REA-compliant, the schema of this accounting database is REA-compliant.

There are two contexts where the REA model is used to guide schema evolution: 1) A schema is not REA-compliant. The schema is required to evolve to be REA-compliant. This case is called non-REA-to-REA evolution. 2) A schema is already REA-compliant. The schema must be maintained to be REA-compliant while schema evolution is required. This case is called REA-to-REA evolution.

Both cases of REA-to-REA and non-REA-to-REA evolution involve the following three tasks:

- REA Description Tasks - Specify an accounting database class as an instance of an REA meta-class. For example, Purchase is described as an economic event or Purchase Payment as a duality relationship.
- Evolution Operation Tasks - Apply evolution operators to manipulate an accounting database schema.
- REA Verification Tasks - Verify if an accounting database schema is REA-compliant.

The following three different methods will use the above tasks to guide evolution process: (The next section, Schema Evolution Scenario, will give an example for each method introduced here.)

1) REA Relationship-Driven Schema Evolution Method: This evolution method starts with an REA relationship description task. While an REA relationship is specified for an accounting database class, the system will evoke the related operators (i.e. an evolution operation task) to complete the schema evolution required by the specification and then evoke the REA verification task to examine if the current schema is REA-compliant.
2) REA Entity-Driven Schema Evolution Method: This evolution method starts with an REA entity description task. While an REA entity is specified for an accounting database class, the system will evoke the related operators (i.e. an evolution operation task) to complete the schema evolution required by the specification and then evoke the REA verification task to examine if the current scheme is REA-compliant.
3) Operation-Driven Schema Evolution Method: This evolution method starts with applying an schema evolution operator. While the accounting database schema is manipulated by evolution operators, the REA verification task is evoked to examine if this evolution operation meet the requirements of REA-compliantness.

3.3 Schema Evolution Scenario

This section describes a schema evolution scenario and demonstrates how three evolution methods can be used in the case of non-REA-to-REA evolution. The

example used here is a primitive accounting database for inventory purchases and it is not compliant to the REA model. Its schema is shown in Figure 3.2. The task is to evolve this non-REA-compliant schema into an REA-compliant schema. The scenario of this non-REA-to-REA evolution contains several sessions where each session corresponds to a major schema evolution task. After five sessions, this evolution process reaches its target schema, an REA-compliant schema. The target schema is shown in Figure 3.3.

Each session contains several steps, where each step corresponds to an REA specification task or an evolution operation task. An REA verification task is the final step of each session and will not be shown in the following discussion. To illustrate this evolution process, we rename the classes of the starting schema to meet the class names of its target schema, i.e. "Purchase Record" becomes "Purchase" and "Payment Record" becomes "Cash Disbursement." (See Figure 3.2)

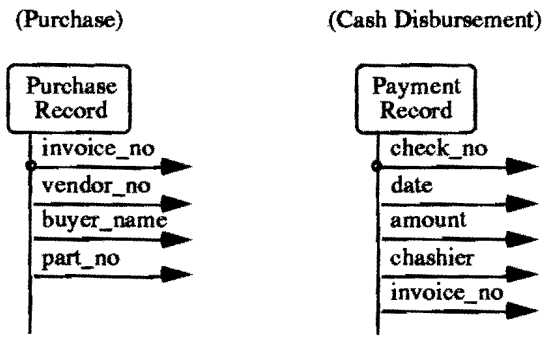

Figure 3.2 Initial Schema of Inventory Purchase

Session I. Evolution Session for Purchase Payment In this session, an REA relationship-driven schema evolution method is used.

Step 1 A user specifies a duality between Class **Purchase** and Class **Cash Disbursement.**

```
[duality instantiate: "Purchase Payment"
increment: "Purchase"
decrement: "Cash Disbursement"]
```

Step 2 The system specifies Class **Purchase** and Class **Cash Disbursement** as events. Then, Classes **Purchase** and **Cash Disbursement** become the instances of events.

```
[Event instantiate: "Purchase"]
[Event instantiate: "Cash Disbursement"]
```

Step 3 The domain of Attribute **invoice_no** of Class **Cash Disbursement** will merge with Class **Purchase**. First, the domain of Attribute **invoice_no** will evolve from a class which contains atomic data values (a "value class") to one which contains abstract objects (an "entity class). Then, this entity class will merge with Entity Class **Purchase**. Attribute **Invoice Number** will evolve to Class **Purchase Payment**. Then, Class **Purchase Payment** becomes an instance of Duality.

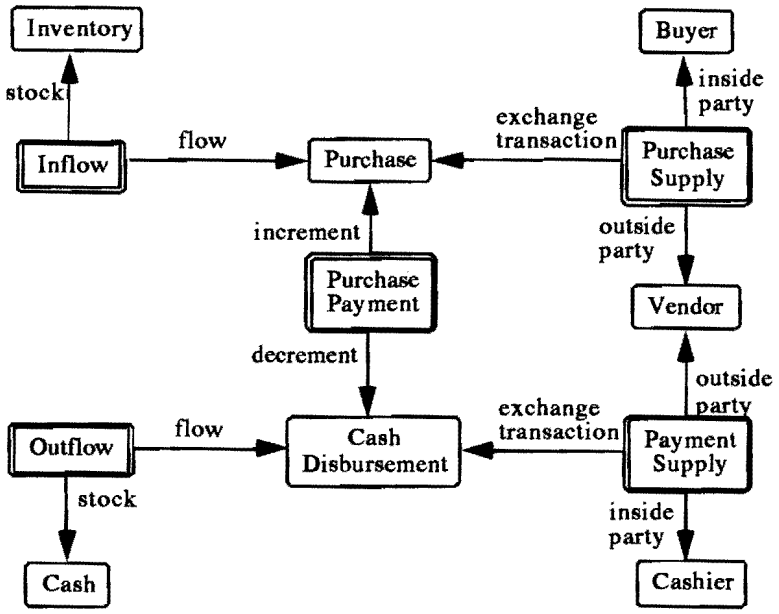

Figure 3.3 Target Schema of Inventory Purchase

Session II. Evolution Session for Purchase Supply In this session, an REA entity-driven schema evolution method is used.

Step 1 The user specifies Buyer as a unit and Vendor as an agent.

```
[Unit instantiate: "Buyer"]
[Agent instantiate: "Vendor"]
```

Step 2 Since the system already has Classes **Purchase** and **Cash Disbursement** as events and REA control is a three-way relationship connecting an event, a unit and an agent, the system creates a control relationship. Since there are two events in the current schema, the user must decide which one participates in this control relationship and also name the control. Here, Class **Purchase** is chosen by the user.

```
[control instantiate: "Purchase Supply"
exchange_transaction: "Purchase"
inside_party: "Buyer"
outside_party: "Vendor"]
```

Step 3 According to the previous REA description, class **Purchase** will sprout itself and generate a new class "Purchase Supply." Attributes **vendor_no** and **buyer_name** of Class **Purchase** will be moved to Class **Purchase Supply**. The domains of these attributes will be converted to Entity Classes **Vendor** and **Buyer**, respectively.

Session III. Evolution Session for Payment Supply (This session uses the same method as the Session II, and is omitted here.)

Session IV. Evolution Session for Inflow Class In this session, an operation-driven schema evolution method is used.

Step 1 A user applies schema evolution operators to convert the domain of Attribute **part_no** of Class **Purchase** into a class and merges this class with Class **Inventory**. The user also renames Attribute **part_no** as "Inflow." Then, the user converts Attribute **Inflow** to an associative class (viz., a class whose instances model relationships). Class **Inflow** becomes a two-way relationship connecting Class **Purchase** and Class **Inventory**.

Step 2 The user describes Class **Inventory** as a resource and Class **Inflow** as an REA relationship stock-flow. Since Class **Purchase** is an Event, Class **Inflow** generated by schema operators should be an REA-compliant class.

Session V. Evolution Session for Outflow Class (This session uses the same method as the Session IV, and is omitted here.)

4. REAtool: a Schema Evolution Guidance Tool

4.1 The Architecture of REAtool

A Schema Evolution and Administration Tool, SEAtool for short, is an experimental prototype that implements the proposed schema evolution methodology and assists a database user, designer, or administrator with the schema evolution. It also can employ domain-specific knowledge, such as REA accounting model, to guide a user to complete evolution tasks. The version of SEAtool that uses the REA accounting model to guide schema evolution is called REAtool. The architecture of REAtool is shown in Figure 4.1. Three main modules of REAtool are SEAshell, SEAengin and SEAbase.

SEAshell provides necessary context information and guides a user through a dialog to complete an evolution task. It also gives feedback to allow a user to validate evolution operations. SEAshell supports the interaction required by evolution

operation tasks. SEAshell (REA Option) also supports the interaction required by REA description tasks.

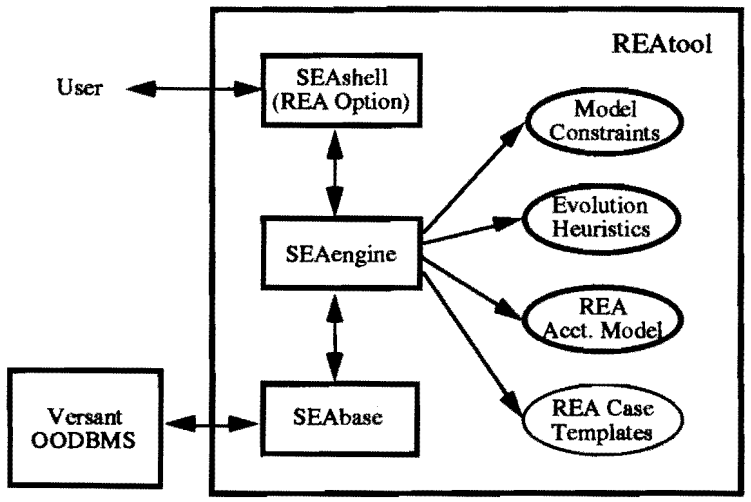

Figure 4.1 The Architecture of SEAtool

The SEAengine module accepts the requests issued by a user from SEAshell and uses two kinds of generic knowledge to guide schema evolution:

- Model Constraints: The constraints required by Object-Based Data Model should be maintained to keep database schema consistent.
- Evolution Heuristics: Schema evolution is guided by Preservation Principle to minimize the loss of the data and meta-data.

The SEAengine module in REAtool uses two additional kinds of domain-specific knowledge to guide schema evolution:

- Domain Knowledge: The REA accounting model is used as a generic model to guide evolution process.
- Templates: Knowledge specific to a sub-domain can be used to guide schema evolution, for example, industry-specific information about REA schema. The current version of SEAtool does not include it.

The SEAbase module defines the internal data structure to store the data and meta-data of a database. SEAbase is built on the top of Versant Object-Oriented DBMS and uses function calls provided by Versant libraries [15]. SEAshell and SEAengine are implemented in Objective-C and SEAbase is implemented in C++. SEAtool prototype is developed under the NeXTSTEP programming environment [10].

4.2 REA Task Guidance

An evolution task is guided by the Task Guidance Panel (TGP), portion of SEAshell. TGP has four components:

1) Schema Browser. There are three kinds of browsers to be used. Network Browser and Hierarchical Browser show the semantic relationship and hierarchical structure of classes. REA Browser shows REA meta-classes and their instances.
2) Context Display illustrates graphic objects involved in an evolution process. Therefore, a user can get a comprehensive and coherent control of the evolution process.
3) Task Catalog. A list of evolution tasks is listed in an organized way to allow a user to choose. Three major kinds of tasks are: REA description, operation specification and REA verification. For example, a user can choose to do the task of duality description.
4) Working Space. Evolution tasks are guided and completed here. Working Space consists of a stack of Dialog Pages. A user is guided by the system to fill Dialog Pages step by step to complete an evolution task.

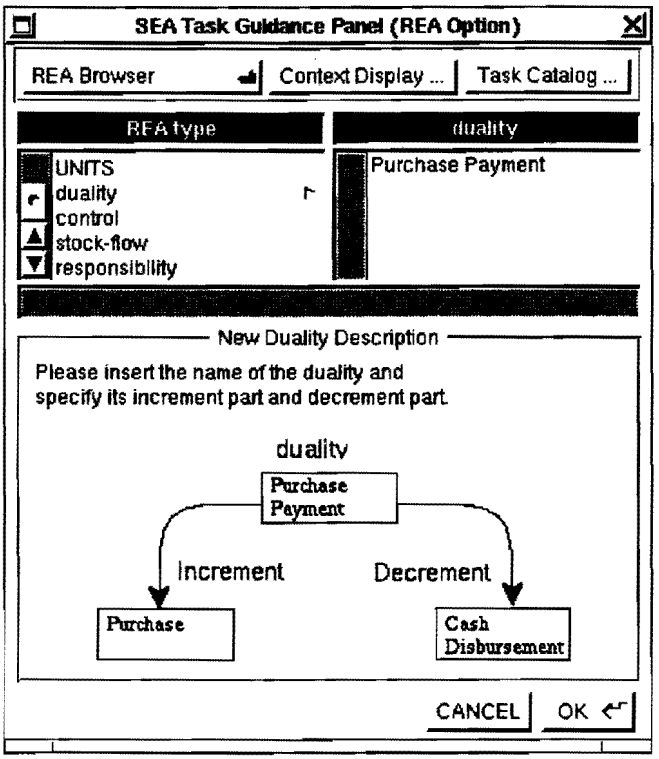

Figure 4.2 Snapshot of REAtool's Task Guidance Panel

A snapshot of Task Guidance Panel is shown in Figure 4.2 to demonstrate the look and feel of SEAtool. Assume a user has chosen the task New Duality Description from Task Catalog. A Dialog Page for describing a new duality is placed in Working Space. The user is first asked to supply the name of the new duality. The user is also asked to specify the Increment and Decrement parts of this duality. After the user

clicks the OK button to confirm the REA description, evolution operators will be evoked to complete the evolution task. After that, a newly created duality is shown in the REA Browser.

5. Concluding Remarks

In this research, we have defined a set of basic evolution schema operators, and have employed evolution heuristics to guide the evolution process. We have also explored the use of domain-specific knowledge to guide the use of the evolution operators. The REA accounting model has been used on our research as an example of such domain-specific knowledge. The SEAtool and REAtool experimental prototypes demonstrate our reseults.

As a future research direction, we will explore more specific domain knowledge to guide schema evolution. The REA accounting model has been successfully used to guide an evolution process, but different types of firms may use different types of accounting databases. For example, the accounting database of a manufacturing-type firm may be quite different from that of a service-type firm [6]. The REA model is a general accounting model, which does not capture some specific sub-domain knowledge. This specific knowledge for some type of firms can be used as a "template" to guide an evolution process. In the architecture of REAtool, the sub-domain template is a source of knowledge that can further constrain and guide the evolution process.

References

1. Banerjee, J., Chou, H-T., Garza, J., Kim, W., Woelk, D., Ballou, N., and Kim, H., "Data Model Issues for Object-Oriented Applications," ACM TOOIS, 5:1, January, 1987.

2. Banerjee, J., Kim, W., Kim, H., and Korth, H., Semantics and Implementation of Schema Evolution in Object-Oriented Databases, Proc. ACM/SIGMOD Annual Conference on Management of Data, San Francisco, California, May 1987.

3. Chen, J.-L. "Heuristic-Based Conceptual Database Schema Evolution," Technical Report, University of Southern California, September 1994.

4. Chen, P. P., "The entity-relationship model: Toward a unified View of data.," ACM Transactions on Database Systems, 1:9-36, 1976.

5. Chiristodoulakis, S., Vanderbroek, J., Li, J., Wan, S., Wang, Y., Papa, M., and Bertino, E., "Development of a Multimedia Information System for an Office Environment." In Proc. of the 10th Int'l Conf. on VLDB, 1984, Pp. 261-271.

6. Grabski, S. and Marsh, R., "Integrating Accounting and Advanced Manufacturing Information Systems: An ABC and REA-Based Approach," AIS Research Symposium, 1994, Phoenix, AZ.

7. Hammer, M. and McLeod, D., "Database Description with SDM: A Semantic Database Model", ACM TODS 6, 3, September 1981, 351-387.

8. Li, Q. and McLeod, D., Conceptual Database Evolution Through Learning, Object-Oriented Databases and Applications, Gupta, R. and Horowitz, E. (Editors), Prentice-Hall, 1989.

9. McCarthy, W., "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Environment," Accounting Review, July 1982, pp. 554-578.

10 NeXTSTEP Version 3, NeXT Computer, Inc., 1990. (note: NeXTstep is a registed trademark of NeXT Computer, Inc.)

11. Navathe, S. B., Geum, S., Desci, D. K., and Lam, H., "Conceptual Design for Non-database Experts with an Interactive Schema Tailing Tool", Proc. of the 9th Int'l Conf. on the Entity-Relationship approach, Lausanne, Switzerland, Oct., 1990.

12. Penney, D. J. and Stein, J., Class Modification in the GemStone Object-Oriented DBMS. In Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, pages 111-117, 1987.

13. Skarra, A. H., and Zdonik, S. B., The Management of Changing Types in an Object-Oriented Database, Proc. ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland, Oregon, September 1986.

14. Su, S., Krishnamurthy, V. and Lam, H., "An Object-Oriented Semantic Association Model (OSAM*)," AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications, Kumara, S., Kashyap, R., and Soyster, A. (Eds.). American Institue of Industrial Engineering, 1989.

15. Versant ODBMS, "Versant System Manual," Versant Object technology, 1992, Menlo Park, CA. (note: Versant is a trademark of Versant Object Technology Co.)