Pergamon

Expert Systems With Applications, Vol. 9, No. 4, pp. 491–501, 1995

0957-4174/95 $9.50 + .00

0957-4174(95)00019-4

Domain-Knowledge-Guided Schema Evolution for Accounting Database Systems

JIA-LIN CHEN

Information and Computing Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

DENNIS MCLEOD

Department of Computer Science, University of Southern California, Los Angeles, CA 90089, USA

DANIEL O'LEARY

School of Business, University of Southern California, Los Angeles, CA 90089, USA

Abstract—*The static meta-data view of accounting 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 accounting database evolution is clear: a static meta-data view of an accounting database cannot support next generation dynamic environment where system migration, organization reengineering, and heterogeneous system interoperation are essential. This paper presents a knowledge-based approach and mechanism to support dynamic accounting database schema evolution in an object-based data modeling context. When an accounting database schema does not meet the requirements of a firm, the schema must be changed. Such schema evolution can be realized via a sequence of evolution operators. As a result, this paper considers the question: what heuristics and knowledge are necessary to guide a system to choose a sequence of operators to complete a given evolution task for an accounting database? In particular, we first define a set of basic evolution schema operators, employing heuristics to guide the evolution process. Second, we explore how domain-specific knowledge can be used to guide the use of the operators to complete the evolution task. A well-known accounting data model, REA model, is used here to guide the schema evolution process. Third, we discuss a prototype system, REAtool, to demonstrate and test our approach.*

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 cannot support either next generation dynamic database applications such as interactive multi-media information systems (Christodoulakis *et al.*, 1984). In addition, database evolution is necessary to accommodate an environment wherein system migration, organizational reengineering, and heterogeneous operations occur.

There are at least three reasons why 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 incomplete requirement analysis, or even erroneous schema design. Second, the current schema may not

Requests for reprints should be sent to Jia-Lin (Norman) Chen, Mail stop 50B3238, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA.

meet new requirements. This type of database schema may be termed an "obsolete schema"; such obsolete schema may be caused by new requirements for structural changes, e.g., company reorganization, and/or process changes, e.g., business reengineering. Third, there are likely to be implementation compromises, made for cost-benefit reasons. This type of database schema will be referred to as a "compromized schema."

1.2. Research Approach

When an accounting database schema does not meet the current requirements of a firm, the schema must be changed. One important issue is how the contents of the database itself can be adapted to a new schema. The classical way to deal with this is to employ conversion programs; this can of course be time consuming, nontransferable (e.g., database specific) and, costly. An alternative approach is to develop a set of general evolution operators to handle the data adaptation. These schema evolution operators would be used to manipulate the original schema into a new schema, and the populated database would be modified accordingly. Although a set of schema evolution operators is very powerful in conducting an evolution task, how to use these operators to reach a desirable target schema remains an important research issue.

This paper discusses a knowledge-based approach to guide schema evolution. To evolve a database schema by using evolution operators requires two types of knowledge: (1) a domain model that can suggest a potential target schema; and (2) domain-specific heuristics that can guide a user in the choice of a sequence of operators to evolve the current schema to the potential target schema. By using these two types of knowledge, we can build a knowledge-based system to guide schema evolution processes. This allows us to address our research question:

> What heuristics and domain knowledge are necessary to guide a user in the choice of a sequence of operators to complete a given evolution task for accounting database systems?

This question is explained in the following three sections. First, we define a set of basic schema evolution operators and generic evolution principles. Second, we explore what kind of domain-specific knowledge and heuristics that can be used to guide the evolution operators in completing evolution tasks for accounting database systems. The REA accounting data model (McCarthy, 1982) is used here to guide the schema evolution process of an accounting database system. Third, we discuss a tool built to choose and apply the schema 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.

2. BACKGROUND

2.1. Object-Based Data Model

A basic Object-Based Data Model (OBDM) is used here as a framework for our approach to schema evolution. OBDM adopts the most basic and common modeling constructs from such object-based data models as the Extended Entity-Relationship Model (Teorey, Yang, & Fry, 1986), the Semantic Data Model (Hammer & McLeod, 1981), and the Orion Object-Oriented Data Model (Banerjee et al., 1987). The modeling constructs of OBDM include class, attribute, class hierarchy, inheritance, and their associated constraints.

A class is a set of objects of the same type. An instance is an object belonging to a class. There are two kinds of classes: value classes and abstract classes. A value class is a basic type of data such as Booleans, integers, reals, and strings. Abstract classes are further classified as entity classes and relationship classes. In OBDM, these two types of abstract classes have no significant difference.

An attribute of a class is a function mapping the class to another class, the domain class of an attribute. A key of a class is an attribute or a set of attributes that can be used to identify instances within the class. Attributes and their inverses may form a one-to-one, a one-to-many, or a many-to-many mapping.

A class has one superclass except the root class which has none. A class may have one or more subclasses. Subclassing forms a specialization hierarchy: (1) a subclass inherits attributes of its superclass, and (2) the set of instances of a subclass is a subset of the set of instances of its superclass.

2.2. Related Research

ORION (Banerjee et al., 1987), ENCORE (Skarra & Zdonik, 1986), and GemStone (Penny & Stein, 1987) use object-based data models and address the schema evolution problem. These systems define modeling invariants and rules as schema evolution constraints. The set of schema evolution operators is defined to change the database scheme without violating the evolution constraints. These schema evolution operators are designed for a programming system, not for database users or administrators, to manage the schema evolution. Based on an object-based data model, called OSAM* (Su, Krishnamurthy, & Lam, 1989), Schema Tailoring Tool (Navathe et al., 1990) allows a non-expert user to redesign an OSAM* schema by tailoring its old schema. The tailoring process is accomplished through schema evolution operators. The schema evolution operators are designed for a database administration user, but no guidance is provided to use these operators.

PKM (Li & McLeod, 1989) identifies a rich set of conceptual schema evolution operations. These evolution operations are guided through the evolution heuristics.

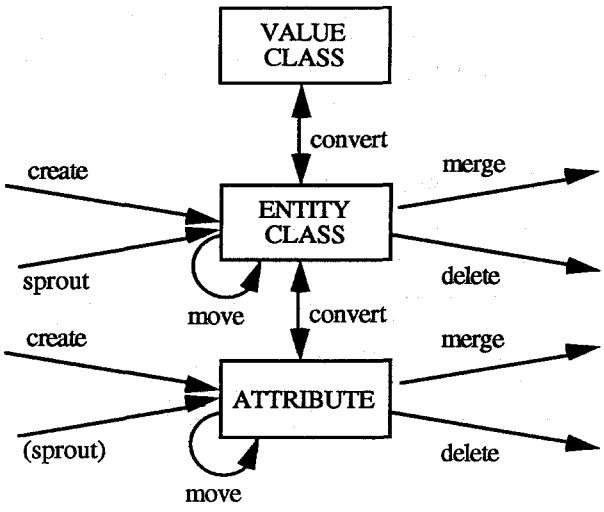


FIGURE 1. Schema evolution operators.

Although no specific domain knowledge is used to guide the schema evolution, the generic database modeling knowledge is used by these evolution guiding heuristics. On the other hand, the knowledge-based approach to guide view creation and integration has proved very promising. For example, Storey and Goldstein (1990) use a conceptual data model to guide the view creation; McCarthy and Rockwell (1989) use the accounting domain model to guide the view integration. In the research described in this paper, we apply the knowledge-based approach to guide schema evolution operations by employing domain-specific data model and heuristics.

3. CONCEPTUAL SCHEMA EVOLUTION

3.1. Schema Evolution Operators

In our approach to database evolution, a set of conceptual evolution operators is used. These operators are relatively generic and domain independent. Specifically, we define four groups of schema evolution operators based on OBDM: (See Figure 1).

(1) Schema Enhancement Operators **create** and **sprout**: The operator **create** creates a new class or attribute. The new class can be a generalization of several classes or a specialization of some class. The operator **sprout** generates a new class with its instances having a one-to-one correspondence with those of its source class.

(2) Schema Reduction Operators **merge** and **delete**: The operator **merge** folds a class into another class or an attribute into another attribute; **merge** deletes meta-data, but keeps the data unchanged. The operator **delete** removes data and associated meta-data.

(3) Schema Restructure Operator **move**: The operator **move** changes the structure of a class hierarchy by moving classes or by moving attributes.

(4) Schema Conversion Operator **convert**: The operator **convert** converts a modeling construct among a value class, an entity class, and an attribute.

3.2. Basic Principles for Schema Evolution

The evolution operators are guided by three primary principles: consistency, propagation, and preservation. Schema evolution operators must keep a schema consistent after they have been applied. This is the consistency principle of schema evolution. To maintain database consistency in schema evolution, evolution operators are ruled by modeling constraints to propagate the changes to related parts of a schema. This is called the propagation principle of schema evolution. Since a propagation effect could change the data and meta-data a user does not intend to change, careful propagation control is necessary. This is called the preservation principle of schema evolution.

The above three principles are realized by using a set of generic evolution heuristics to guide the evolution process. For example, when a subclass is merged into a superclass, its attributes and the values defined by these attributes could be lost. One way to avoid this type of loss of meta-data and data is to move the attributes of the subclass to a superclass. In what follows we focus on how domain knowledge and domain-specific heuristics are used to guide the evolution process.

4. SCHEMA EVOLUTION GUIDED BY DOMAIN KNOWLEDGE

4.1. Object-Based REA Accounting Model

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

The REA accounting model is a generalized accounting framework used to capture the interaction of economic resources, economic events and economic agents for accounting systems (McCarthy, 1982). 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, or distribution. Purchase and cash disbursement are examples of economic events. Economic agents are persons and parties who participate in the economic events, e.g., vendors. Economic units are a subset of economic agents and are inside participants, e.g., cashiers and buyers.

There 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; 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 an exchange transaction part), an economic agent (as an outside party), and an economic unit (as an inside party). For example, Purchase Supply is a control relationship that associates Purchase (an event with the role of exchange transaction part), Vendor (an agent with the role of outside party), and Buyer (a unit with the role of 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.

The REA model was originally described in an entity-relationship representation (Chen, 1976). Since this paper employs an object-based approach, the REA entities are modeled as entity classes and their relationships are modeled as relationship classes in the OBDM.

There are no significant difference between entity classes and relationship classes in the OBDM. Furthermore, the role of an entity, which is pertinent to a relationship in an ER representation, is modeled as an ordinary attribute of the relationship class in the OBDM. This transformation of the REA model eases the evolution process by simplifying modeling constructs. 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. An object-based representation of the REA model is summarized in Figure 2.

4.2. REA Guidance for Schema Evolution

The REA accounting model is used to guide the 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 is an instance of one of the classes of the REA model. For example, the class Purchase Payment is an instance of the meta-class Duality. A family of classes in an accounting database will be said to be REA-compliant if and only if:

- the class is an instance of one of the 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. If all classes of an accounting database are REA-compliant, then the schema of this accounting database is REA-compliant.

The concept of REA-compliance provides an instrument to verify whether a current schema is REA-compliant. Moreover, it also suggests a potential target schema for a current schema if the current schema is not REA-compliant. There are two contexts where the REA model is used in the schema evolution of an accounting database: (1) A current schema is already REA-compliant. The schema must be maintained as REA-compliant while schema evolution is required. This case is called REA-to-REA evolution. In this case, the REA model is used to verify whether an evolution operation keeps the current schema REA-compliant. (2)

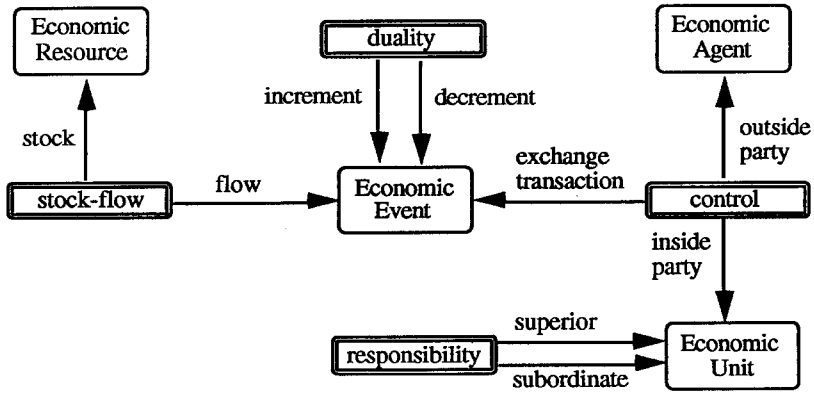


FIGURE 2. The object-based REA accounting model.

A current schema is not REA-compliant. The schema is required to evolve in order to be REA-compliant. This case is called non-REA-to-REA evolution. In this case, the schema description and the evolution heuristics based on the REA model are used to guide a system for selecting a sequence of evolution operations, so that the target schema will be REA-compliant after this sequence of evolution operations. The guidance for non-REA-to-REA evolution is the focus of this paper.

4.2.1. REA Schema Description. To guide schema evolution from the current schema to a potential REA-compliant target schema, we need to build a connection between the current schema and its potential target schema. REA schema description provides this connection by describing a class of the current accounting schema as an instance of an REA meta-class. For example, class Purchase is described as an economic event and class Purchase Payment as a duality relationship. This description provides the necessary information to check the difference between the current schema and the potential target schema, and allows the system to trigger evolution heuristics to guide the schema evolution.

Since the current schema and the potential target schema could be complicated, the schema description is divided into four types of tasks. Each of them is centered around an REA relationship and its associated entity classes. These four tasks are: (1) Stock-Flow Description Task, including the descriptions for a stock-flow relationship, an economic resource, and an economic event; (2) Duality Description Task, including the descriptions for a duality relationship and two economic events; (3) Control Description Task, including the descriptions for a control relationship, an economic event, an economic agent, and an economic unit; and (4) Responsibility Description Task, including the descriptions for a responsibility relationship and two economic units.

A complete schema description of a given task could be redundant. Some sort of partial description is enough to provide the necessary information for a system to accomplish an REA schema description. There are two methods for a user to specify a partial description:

(1) REA Relationship-Driven Schema Description Method. This description method starts with a description of an REA relationship. While an REA relationship is specified for accounting database classes, the system will add REA entity descriptions accordingly. The REA Schema Description part of Session I in Section 4.3 provides an example.

(2) REA Entity-Driven Schema Description Method. This description method starts with an REA entity description. While REA entities of a given task is described for accounting database classes, the system will add REA relationship descriptions accordingly. The REA Schema Description part of Session II in Section 4.3 provides an example.

4.2.2. REA Evolution Heuristics. Evolution will be accomplished using a variety of heuristic rules. A heuristic rule for guiding the evolution operation has three parts. (1) Trigger is the difference between the current schema and the target schema suggested by the REA model. (2) Condition is the current schema characteristics that are relevant to an evolution operation. (3) Action is the selected evolution operation. Based on the REA description supplied by a user, the system checks the difference between the current schema and the target schema. The difference is used to trigger evolution heuristic rules. If the condition required by a heuristic rule meets the current schema, a schema evolution operator will be activated. The selected schema evolution operation will further change the current schema. The change of the current schema will change the difference between the current schema and the target schema. This updated difference will further trigger evolution heuristic rules until there is no difference between the current schema and the target schema. The following is a set of REA-based heuristics:

A1: Value Conversion Heuristic

[Trigger] A class X is missing and it is an REA entity class.

[Condition] There exists a value attribute whose domain Y is equivalent to the missing class X.

[Action] Apply the operator Value-to-Entity Convert to convert the value class Y into an entity class.

Example: See Evolution Operation Step 1 and Step 2 of Session II in Section 4.3.

A2: Attribute Conversion Heuristic

[Trigger] A class X is missing and it is one of these classes: stock-flow, duality, or responsibility.

[Condition] There exists an attribute h who connects two REA entity classes Y and Z.

[Action] Apply the operator Attribute-to-Entity Convert to convert the attribute h into an entity class.

Example: See Evolution Operation Step 3 of Session I in Section 4.3.

A3: Entity Sprout Heuristic

[Trigger] A class X is missing and it is class control.

[Condition] There exists an attribute h of a class Y that is one of the classes Event, Agent, or Unit; and the attribute h has class Z as its domain, where Z is one of the classes Event, Agent, or Unit, but not Y.

[Action] Apply the Class Sprout to generate the class control from the class Y.

Example: See Evolution Operation Step 3 of Session II in Section 4.3.

A4: Attribute Move Heuristic

[Trigger] An attribute h of the class control is missing. The domain of the attribute h is one of the classes Event, Agent, or Unit.

[Condition] There exists an attribute i of the class X, which is the domain of an attribute of the control. The domain of the attribute i is the domain of the attribute h.

[Action] Apply the operator Attribute Move to move the attribute i from the class X to class control. The attribute i will become the attribute h.

Example: See Evolution Operation Step 4 of Session II in Section 4.3.

A5: Value Link Heuristic

[Trigger] A class X is missing and it is one of these classes: stock-flow, duality, or responsibility.

[Condition] There exists no attribute between class Y and class Z, where either Y is a Resource and Z is an Event if X is a stock-flow, or both Y and Z are Events if X is a duality, or both Y and Z are Units if X is a responsibility. And a non-key attribute of class Y has a value domain W that is equivalent to the domain of the key attribute of the class Z.

[Action] Apply the operator Value-to-Entity Convert to convert the value class W to an entity class X.

Example: See Evolution Operation Step 1 of Session I in Section 4.3.

A6: Class Merge Heuristic

[Trigger] The class X and class Y are redundant.

Kit[Condition] SitEThe class X and class Y have the same key attribute.

[Action] Apply the operator Class Merge to merge the classes X and Y.

Example: See Evolution Operation Step 2 of Session I in Section 4.3.

4.3. Schema Evolution Scenario

This section describes a schema evolution scenario and demonstrates how REA description and evolution heuristics can be used to guide 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. In the example, it is assumed that due to growth of the company, this primitive schema is unable to support more sophisticated and complicated enterprise-wide accounting applications. Since a schema based on the REA model can provide a richer view for an organization, the task here is to evolve this non-REA-compliant schema into an REA-compliant schema. The scenario of this non-REA-to-REA evolution contains several short interactive sessions where each session corresponds to a major schema evolution task.

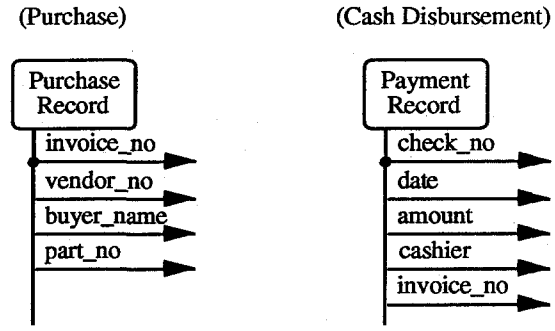


FIGURE 3. Initial schema of inventory purchase.

Each session contains an REA schema description and several steps of schema evolution operations, which are guided by evolution heuristics. 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).

4.3.1. Session I. Evolution Session for Purchase Payment. In this session, a schema evolution task for a duality relationship is selected.

REA schema description. An REA Relationship-Driven Schema Description Method is used in this session. A user describes a duality relationship between class Purchase and class Cash Disbursement. According to this user's description, the system specifies class Purchase and class Cash Disbursement as events.

Schema evolution operation. After the REA schema description of the current schema is accomplished, the evolution heuristics will guide the evolution operators in the following three steps. Figure 4 shows the schema changes caused by the schema evolution operation in each step of this session.

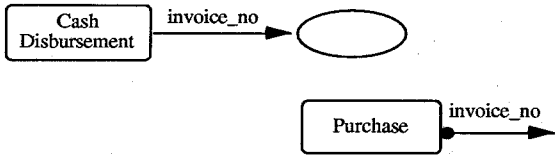
Step 1: Value-to-entity convert. The duality task which includes Event Cash Disbursement and Event Purchase misses the relationship duality. This situation triggers the Value Link Heuristic (A5). The required condition of the heuristics meets the condition of the current schema, where event Cash Disbursement has an attribute invoice_no and this attribute happens to be the same as the key attribute of Event Purchase. The operation Value-to-Entity Convert is applied to the value class, which is the domain of the atrribute invoice_no of event Cash Disbursement.

Step 2: Class merge. The domain of attribute invoice of event Cash Disbursement is class Invoice, which is redundant to event Purchase. The

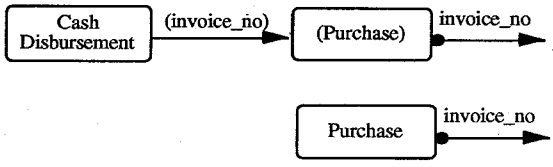
situation triggers Class Merge Heuristic (A6). The required condition of the heuristics meets the condition of the current schema, where event Purchase and event Invoice have the same key attribute invoice_no. The operation Class Merge is applied to class Invoice and merges it into event Purchase.

Step 3: Attribute-to-entity convert. The duality task which includes event Cash Disbursement and event Purchase misses the relationship duality. This situation triggers the Attribute Conversion Heuristic (A2). The required condition of the heuristics meets the condition of the current schema, where event Cash Disbursement has an attribute invoice that connects to class Purchase. The operation Attribute-to-Entity Convert is applied to the attribute invoice and converts it to a duality, named Purchase Payment.

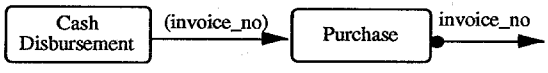
4.3.2. Session II. Evolution Session for Purchase Supply. In this session, a schema evolution task for a control relationship is selected.



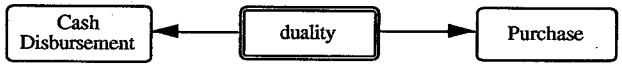
(a) (Part of) Initial Schema



(b) (Part of) Schema after the Operation Value-to-Entity Convert



(c) (Part of) Schema after the Operation Class Merge



(d) (Part of) Schema after the Operation Attribute-to-Entity Convert

FIGURE 4. Schema evolution task for a duality relationship.

REA schema description. An REA Entity-Driven Schema Description Method is used in this session. The user specifies Buyer as a unit and Vendor as an agent. 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.

Schema evolution operation. After the REA schema description of the current schema is accomplished, the evolution heuristics will guide the evolution operators in the following five steps:

Step 1: Value-to-entity convert. An entity class Buyer is missing. This situation triggers the Value Conversion Heuristic (A1). The required condition of the heuristics meets the condition of the current schema, where the domain of attribute buyer_name is equivalent to class Buyer. The operation Value-to-Entity Convert is applied to the value class, the domain of the attribute buyer_name. The newly converted entity class becomes the class Buyer.

Step 2: Value-to-entity convert. An entity class Vendor is missing. This situation is the same as the situation of the last step, and therefore, it triggers the Value Conversion Heuristic (A1). The required condition of the heuristics meets the condition of the current schema, where the domain of attribute vendor_no is equivalent to class Vendor. The operation Value-to-Entity Convert is applied to the value class, the domain of the attribute vendor. The newly converted entity class becomes the class Vendor.

Step 3: Class sprout. In the task of Purchase Supply, class control is missing. This situation triggers the Entity Sprout Heuristic (A3). The required condition of the heuristics meets the condition of the current schema, where event Purchase has an attribute buyer pointing to unit Buyer. The operation Class Sprout is applied to event Purchase and generates a control, Purchase Supply.

Step 4: Attribute move. The unit Buyer is not found as the domain of an attribute of the control Purchase Supply. This situation triggers the Attribute Move Heuristic (A4). The required condition of the heuristics meets the condition of the current schema, where the event Purchase has an attribute buyer and the event Purchase is a domain of the attribute exchange transaction of the control Purchase Supply. The operation Attribute Move is applied to the attribute buyer and moves it to the control Purchase Supply.

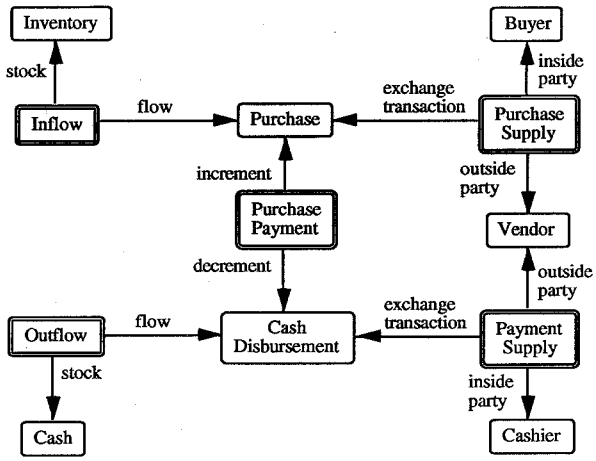


FIGURE 5. Target schema of inventory purchase.

Step 5: Attribute Move. The unit Vendor is not found as the domain of an attribute of the control Purchase Supply. This situation triggers the Attribute Move Heuristic (A4). The required condition of the heuristics meets the condition of the current schema, where the event Purchase has an attribute vendor_no and the event Purchase is a domain of the attribute exchange transaction of the control Purchase Supply. The operation Attribute Move is applied to the attribute vendor_no and moves it to the control Purchase Supply.

There are three more sessions. The task of Session III, Evolution Session for Payment Supply, is a task for the control relationship. The REA schema description and the heuristic guidance of the evolution operations of this task are similar to the task of Session II. The tasks of Session IV, Evolution Session for Inflow, and Session V, Evolution Session for Outflow, are tasks for the stock-flow relationship. The REA schema description and the heuristic guidance of evolution operations of these tasks are similar to the task of Session I. The details of these three sessions are omitted here. After five short sessions, this evolution process reaches its target schema, an REA-compliant schema. The target schema is shown in Figure 5.

5. REAtool: A SCHEMA EVOLUTION GUIDANCE TOOL

5.1. The Architecture of REAtool

A Schema Evolution Administration Tool, SEAtool for short, is an experimental prototype that implements the conceptual-level operators and basic guiding principles for schema evolution. It can also employ domain knowledge, such as REA accounting model and REA evolution heuristics, to guide a user in the completion of evolution tasks. The version of SEAtool that uses the REA domain knowledge to guide schema evolution is called REAtool. REAtool is divided into three layers: SEAshell, SEAengine and SEAbase. Major modules of REAtool are shown in Figure 6.

SEAshell is the user interface layer of REAtool. It provides the necessary context information and guides a user through a dialog to select an evolution task, which can be a direct evolution request or a REA schema description. It also gives feedback to allow a user to validate evolution operations. Section 5.2 will show further how SEAshell interacts with the user.

SEAengine is the intelligent layer of REAtool. It accepts direct evolution requests from SEAshell issued by the user and uses two kinds of generic knowledge to guide schema evolution:

- Data Model Constraints: Object-Based Data Model provides the constraints for the usage of schema evolution operators.
- Basic Evolution Principles: Schema evolution is guided by basic evolution principles to maintain schema consistency, handle evolution propagation, and minimize the loss of data and meta-data.

SEAengine also accepts REA schema description and uses two additional kinds of domain knowledge to guide schema evolution:

- Object-Based REA Model: The REA accounting model suggests a potential target schema to guide the evolution process.
- REA Evolution Heuristics: A set of REA heuristic rules is used to guide the evolution operators in order to reach a REA-compliant target schema.

SEAengine has three major modules: (1) Schema Comparison Analyzer, comparing the current schema and the potential target schema suggested by the REA model; (2) Evolution Operation Planner, executing evolution operations with a copy of the current schema in Evolution Space; and (3) Evolution Heuristic Manager, using the heuristics to guide evolution operations. The

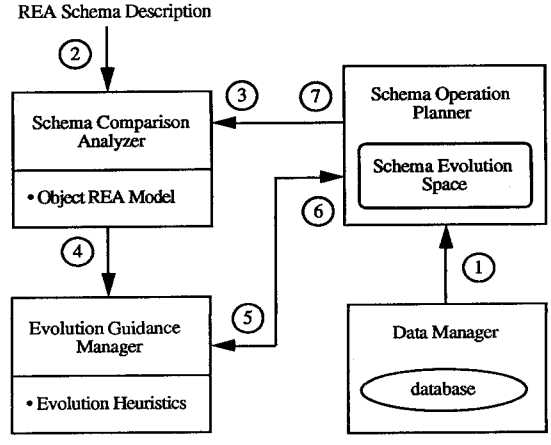


FIGURE 6. Major modules and information flow of REAtool.

basic information flow of SEAengine is shown in Figure 6. Each detail step is listed below:

(1) A copy of the current database schema is sent to the Evolution Operation Planner.
(2) The Schema Comparison Analyzer accepts an REA schema description from a user through SEAshell.
(3) Schema Comparison Analyzer receives the current schema from Evolution Operation Planner.
(4) Schema Comparison Analyzer detects the difference between the current schema and the potential target schema. The schema difference is sent to Evolution Heuristic Manager to trigger the REA evolution heuristics.
(5) Evolution Heuristic Manager checks the state of the current schema in Evolution Operation Planner with the condition of a triggered rule.
(6) While the condition of a heuristic rule is satisfied, an evolution operation is issued and, thereafter, updates the current schema.
(7) The updated current schema and its REA description is fed back to Schema Comparison Analyzer and another cycle starts.

SEAbase layer defines a data manager 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 (Versant Object Technology, 1992). SEAshell and SEAengine are implemented in Objective-C and SEAbase is implemented in C++. SEAtool prototype is developed under the NeXTSTEP programming environment (NeXT Computer Inc., 1990).

5.2. Task Guidance of REAtool

An evolution task is guided by the Task Guidance Panel (TGP), portion of SEAshell. A user interacts with TGP to submit a direct evolution request or a REA schema description. TGP has four components:

(1) Schema Browser. There are three kinds of browsers that can 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 comprehensive and coherent control of the evolution process.
(3) Task Catalog. A set of direct evolution operation tasks and REA schema description tasks organized in a hierarchical menu to allow a user to choose. 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.

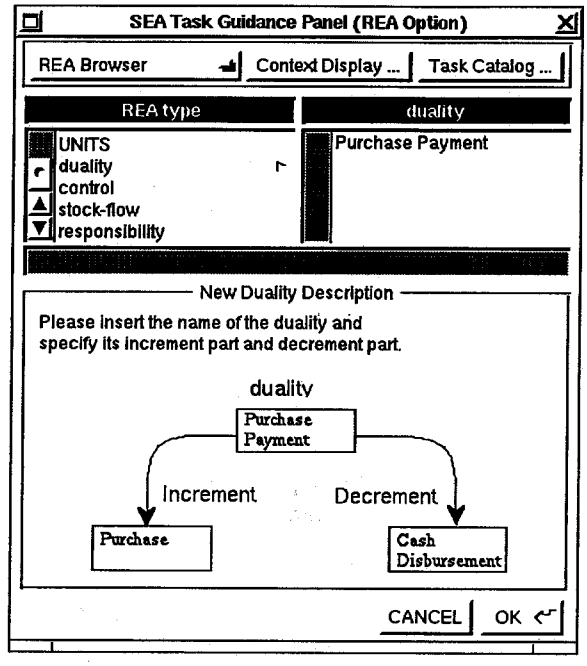


FIGURE 7. Snapshot of REAtool's task guidance panel.

A snapshot of the Task Guidance Panel is shown in Figure 7 to demonstrate the look and feel of REAtool. Assume a user has chosen the task Duality Description from the Task Catalog. A Dialog Page for describing a duality is placed in Working Space. The user is first asked to supply the name of the 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.

6. CONCLUDING REMARKS

6.1. Research Results

In this research, we have defined a set of basic schema evolution operators, and built a knowledge-based system to guide the usage of this set of schema evolution operators. The knowledge-based system employs such domain knowledge as a domain model and a set of domain-specific heuristics. The REA accounting model has been used on our research as an example of such domain knowledge. The experimental prototype, REAtool, demonstrates our results.

6.2. Future Research Direction

This research uses only the REA model as domain knowledge. While the REA accounting model has been successfully used to guide an evolution process, different types of firms may use different types of accounting databases. For example, the accounting database of a

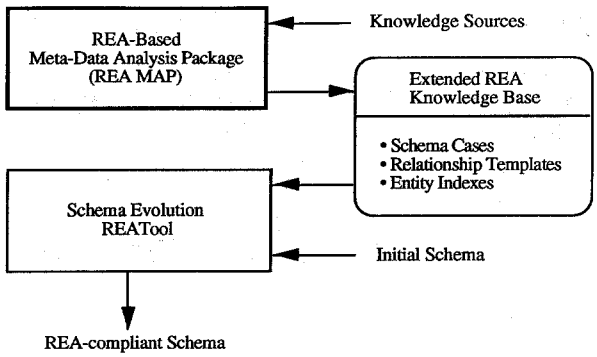


FIGURE 8. Architecture of REA knowledge base and REA-based tool set.

manufacturing firm may be quite different from that of a service firm (Grabski & Marsh, 1994). The REA model is a general accounting model, which does not capture some specific domain knowledge.

As a future research direction, we will explore more specific domain knowledge to guide schema evolution. McCarthy and Rockwell (1989) and Rockwell (1992) classify the accounting knowledge sources into three levels of generality: Principles Level Knowledge, Industry Level Knowledge, and Company Level Knowledge. We could employ these three levels of knowledge sources to facilitate accounting database evolution. To guide schema evolution, we provide an additional dimension that is based on the levels of granularity for knowledge utilization:

- Schema Case Knowledge. This type of knowledge includes abstract schema cases from different types of industries and concrete schema cases from various companies, that can be further classified by the type of its industry.
- Relationship Template Knowledge. This includes knowledge of various of templates that are organized as REA relationships and whose participant entities are described by the REA description.
- Entity Index Knowledge. This knowledge includes a list of accounting terms, and their synonyms, used to name classes and attributes of an accounting database.

These three levels of knowledge will enhance the current REAtool in at least three ways: (1) Schema Case Knowledge will assist in the choice of database schema by providing a schema "library" with additional knowledge, about specific and relevant schema, to guide evolution operations; (2) Relationship Template Knowledge will find related classes of each relationship in an evolution task and describe these related classes in REA terminology; and (3) Entity Index Knowledge will resolve unknown classes and attributes of the current schema and link them to Schema Case Knowledge and Relationship Template Knowledge. To acquire these three levels of knowledge, we will build an REA-based Meta-data Analysis Package, called REA MAP, to analyze the different sources of accounting knowledge, such as the Encyclopedia of Accounting Systems (Pescow, 1976), a collection of accounting database schema of various companies, and an accounting lexicon, etc. Figure 8 shows the architecture of the REA knowledge base and the tool set of REAtool and REA MAP.

Our findings on a knowledge-based system to facilitate the evolution of databases also have implications for the evolution of knowledge bases themselves. For example, in some cases evolution of knowledge bases can be accomplished using a similar guiding framework, by which the current knowledge elements—such as schema cases, relationship templates, and even heuristic rules—can be restructured to fit the requirements of a "normalized" target system. However, that discussion is substantial and beyond the scope of this paper.

REFERENCES

Banerjee, J., Chou, H.-T., Garza, J., Kim, W., Woelk, D., Ballou, N., & Kim, H. (1987). Data model issues for object-oriented applications. *ACM Transactions on Office Information Systems*, **5**(1), 3–26.

Banerjee, J., Kim, W., Kim, H., & Korth, H. (1987). Semantics and implementation of schema evolution in object-oriented databases. *Proceedings of the ACM/SIGMOD Annual Conference on Management of Data*, San Francisco, California, 311–322.

Chen, P. P. (1976). The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, **1**(1), 9–36.

Christodoulakis, S., Vanderbroek, J., Li, J., Wan, S., Wang, Y., Papa, M., & Bertino, E. (1984). Development of a multimedia information system for an office environment. *Proceedings of the 10th International Conference on Very Large Data Bases*. VLDB Endowment, Saratoga, California, 261–271.

Grabski, S., & Marsh, R. (1994). Integrating accounting and advanced manufacturing information systems: An ABC and REA-based approach. *AIS Research Symposium*, Phoenix, AZ.

Hammer, M., & McLeod, D. (1981). Database description with SDM: A semantic database model. *ACM Transactions on Database Systems*, **6**(3), 351–387.

Li, Q., & McLeod, D. (1989). Conceptual database evolution through learning. In R. Gupta and E. Horowitz (Eds.), *Object-oriented databases and applications*, (pp. 62–74). Prentice-Hall.

McCarthy, W. E. (1982). The REA accounting model: A generalized framework for accounting systems in a shared environment. *Accounting Review*, **57** (July), 554–578.

McCarthy, W. E., & Rockwell, S. R. (1989). The integrated use of first-order theories, reconstructive expertise, and implementation heuristics in an accounting information system design tool. *Proceedings of the 9th International Workshop on Expert Systems and their Applications*, Avignon, France, 537–548.

NeXTSTEP, (1990). *NeXTSTEP version 3*, NeXT Computer, Inc. (Note: NeXTSTEP is a registered trademark of NeXT Computer, Inc.)

Navathe, S. B., Geum, S., Desci, D. K., & Lam, H. (1990). Conceptual design for non-database experts with an interactive schema tailing tool. *Proceedings of the 9th International Conference on the Entity-Relationship Approach*, Lausanne, Switzerland, 25–42.

Pescow, J. K. (Ed.), (1976). *The encyclopedia of accounting systems*. Englewood Cliffs, NJ: Prentice-Hall.

Penney, D. J., & Stein, J. (1987). Class modification in the gemstone object-oriented DBMS. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*. 111–117.

Rockwell, S. R. (1992). The conceptual modeling and automated use of reconstructive accounting domain knowledge. Ph.D. Dissertation of Michigan State University.

Skarra, A. H., & Zdonik, S. B. (1986). The management of changing types in an object-oriented database. *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*. Portland, Oregon, 483–495.

Storey, V. C., & Goldstein, R. C. (1990). An expert view creation system for database design. *Expert Systems Review for Business and Accounting*, **2**(3), 19–45.

Su, S., Krishnamurthy, V., & Lam, H. (1989). An object-oriented semantic association model (OSAM*). In S. Kumara, R. Kashyap, & A. Soyster (Eds.), *AI in industrial engineering and manufacturing: Theoretical issues and applications*. American Institute of Industrial Engineering, Norcross, GA.

Teorey, T., Yang, D., & Fry, J. (1986). A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, **18**(2).

Versant ODBMS (1992). *Versant system manual*. Versant Object Technology, Menlo Park, CA. (Note: Versant is a trademark of Versant Object Technology Co.).