Implementing a Categorical Information System*

Michael Johnson[1] and Robert Rosebrugh[2]

[1] Department of Computer Science
Macquarie University
mike@ics.mq.edu.au

[2] Department of Mathematics and Computer Science
Mount Allison University
rrosebrugh@mta.ca

Abstract. The authors have proposed using category-theoretic sketches to enhance database design and integration methodologies. The algebraic context is called the Sketch Data Model (SkDM) and mathematically describes databases, views and their updates, and other database concepts. The system described here is a freely available graphical Java environment with a module that compiles a design incorporating algebraically specified constraints into database schemas that interface directly with modern database management systems. It therefore supports, inter alia, rapid prototyping.

Keywords: Semantic data model, category theory, graphical database design.

1 Introduction

Although the database management systems (DBMS) in wide use for the past dozen years have all been "relational", the most popular design method remains the Entity-Relationship-Attribute (ERA) diagram. That this is so is not surprising given that the latter is a natural and simply understood graphical paradigm. Despite this design-implementation disconnect, there are straightforward procedures and a variety of both commercial and freely available software applications that allow creation and manipulation of ERA diagrams and then translate these designs into relational database schemas. See [8] for some examples.

The system described here similarly implements the database design concepts of the *Sketch Data Model (SkDM)* [4] which is based on categorical universal algebra. The SkDM extends both the entity-relationship model [1] and the functional data model of Shipman [6]. Entities and attributes are modeled using a simple graphical language. Relationships among entities are expressed, and may also be constrained, using concepts from category theory. SkDM constraints can express, among other things, the selection, projection, join and (disjoint) union operations of relational algebra. Formally, these ideas are expressed using a special case of

* Research partially supported by grants from the Australian Research Council and NSERC Canada.

the class of categorical theories called *sketches*. The particular sketches we use are called Entity-Attribute (EA) sketches and are described below.

For the Sketch Data Model, our EASIK system supports graphical definition and manipulation of EA-sketches and automatic compilation into SQL database schemas. It is capable of graphically specifying EA sketches, storing their description in XML documents and generating relational (SQL) database schemas that implement the SkDM designs, including constraints.

EASIK is the first system that supports SkDM modelling. It produces an SQL database schema which can be loaded into a DBMS. Furthermore the resulting database will enforce the SkDM constraints. Having such a system is crucial to validating the SkDM project. The current system has two main modules. The first module is a graphical engine that allows point-and-click construction and manipulation of an EA sketch. From the graphically presented EA sketch the system generates an XML document that encodes the sketch, including constraints. The second module compiles a design stored as an XML document to an SQL database schema that implements the design. The SkDM has been the subject of extensive theoretical studies by the authors and collaborators, for example [3,4,5]. M. Johnson and C. N. G. Dampney have been engaged for several industrial consultancies using the SkDM [2].

2 An Example

Before describing the EASIK implementation, we provide a short tour of the sketch data model with the aid of an EASIK screen. Many other examples can be found in [4] and papers cited there, including the case-studies and consultancies. We assume familiarity with standard data modeling concepts such as ERA diagrams and relational database schemas and algebra. Some familiarity with the basic language of categories applied to Computer Science (for example in [7]) will be helpful.

A conference program committee might use a database with information on its own members, authors, articles submitted, and their status. Among the rules we assume are the following: a person in the database is an author or committee member (not both); there is an assignment of a single committee member as first reader for each paper; papers may have several authors and vice versa; for accepted papers one or more of the authors is recorded as a presenter.

An EA sketch has four components which we outline for a database schema to support the program committee example. For a formal definition of EA sketch see [5].

The first component is a directed graph G, like that in Figure 1. We note some similarities with an Entity-Relationship diagram that might describe the same application domain, and some important differences.

Nodes of G represent entities, but there are no “relationship” nodes. The authorship entity allows several authors to be among the “authorship” of a paper and one author to have authorship of several papers. In an ER diagram authorship might be a relationship from author to paper. Here that is expressed by the

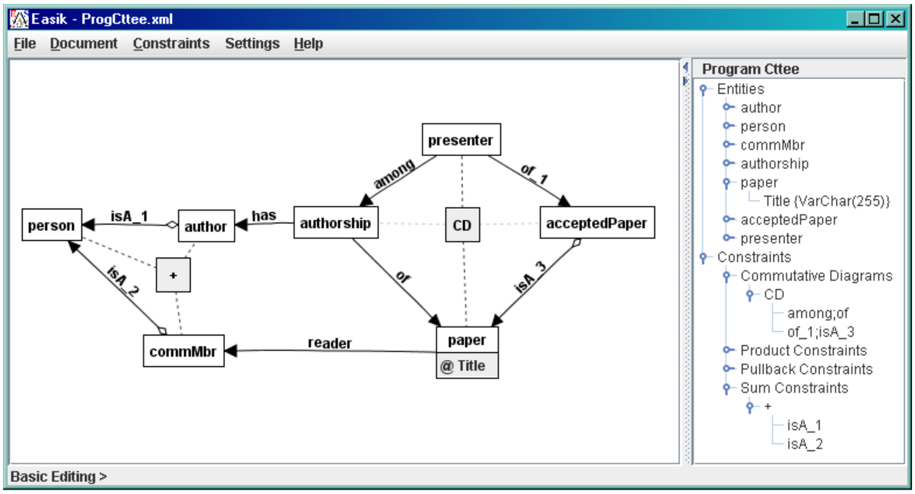

Fig. 1. EASIK screen for part of a Program Committee database

directed edges from authorship. Instead of being modeled as a relationship set in the ER fashion, authorship is also modeled as an entity set. Thus, the directed edges from authorship are modeled by functions specifying for each authorship who is the author and which paper they wrote. The is_a relationships are denoted here by edges indicated $\diamond\!\!\longrightarrow$. The other directed edges in G, for example that from paper to commMbr, are modeled by functions (and can be thought of as methods or, from the database perspective, as integrity constraints). Given an instance of their source type they return an instance of their target type.

The other three components of an EA sketch do not appear in ER diagrams and they express database constraints. In EASIK they are indicated graphically.

The second component is a set of *commutative diagrams.* A commutative diagram is a pair of paths in G with common source and target. They are used to specify equality of function constraints. In our example, the two paths from presenter to paper are a commuting diagram. This represents a real-world constraint: Each presenter instance has an authorship of a paper and each presenter instance has an acceptedPaper which is *the same* paper. On the screen, this diamond of edges encloses a CD icon for which the details are also found among constraints in the right panel of the screen. In contrast, the two paths from authorship to person (around the parallelogram) do *not* form a commutative diagram.

The last two components of an EA sketch express database constraints graphically. They require a node of G to have a value in a database state that depends on values of other nodes in the state.

The third component of an EA sketch is a set of *finite cones* in G. For details of their syntax and semantics we refer the reader to [4] or [5], but we do note here that is_a edges are expressible using finite cones.

The fourth component of an EA sketch is a set of *finite discrete (or sum) cocones.* A discrete cocone has a single *vertex* node, a set of *base* nodes a path *to* the vertex node *from* each base node. In Figure 1 there is a discrete cocone with vertex person. Its base vertices are commMbr and author. The links to the vertex from the base nodes are the is_a edges. On the screen, these nodes link with a + icon for which the details are also found in the right panel of the screen. This cocone expresses that (the elements of the entity set) person are exactly the disjoint union (sum) of (the elements of the entity sets) commMbr, and author. Note that we thus enforce the constraint that committee members are not allowed to be authors.

We do not show most of the attributes—the "A" in EA—in the screen graph of Figure 1, but they are definitely a part of the EA sketch. Attributes are (often large) fixed value sets. Examples in this case are the name of a person, the title of a paper, the year-joined of a committee member, and so on. On the right panel attributes and their data-types are listed with their entity. They may optionally be shown on screen UML style, as we do for paper. Formally an attribute is the vertex of a sum cocone whose linking paths are called *elements*. In every state of the database, an attribute's value is exactly the disjoint union of its elements.

The theory of the sketch data model considers categories of *models* of EA sketches. A model is prescribed by value sets for entities and attributes and functions among them prescribed by the edges of G which satisfy the constraints. Our interest is instead to translate the sketch into an SQL database schema which maintains the constraints on entity sets prescribed by the sketch.

3 Implementation of the SkDM with EASIK

We begin the description of our implementation with some of the design desiderata for EASIK:

The graphical front end is written in Java for portability. All user input is gathered through the GUI environment. Saved files are XML documents. EASIK supports the database design and then handles the generation of an SQL database schema. Access to the database schema and data manipulation (input and queries) is via a user selected database management system. (Thus, database implementation requires an ambient DBMS; export of the SQL schema as text is also available.) Entities have a system-generated primary key. Keys are definable within the sketch and are in the exported SQL schema. Attributes must be based on data types for the platform specified by the user and are specified in an understandable format without adding clutter. Commutative diagrams and other constraints are representable graphically and exported as triggers and procedures to the SQL schema. The user may select a path of edges of any length to create constraints. The user may add, delete, edit and rename all sketch elements. Database schema export is accomplished automatically and includes all information about constraints, primary and foreign keys. Drivers for interactions with database platforms are included.

These design criteria are met by the application which is available for download from `http://mathcs.mta.ca/research/rosebrugh/Easik/`. EASIK uses the graph display package `JGraph` and translates between XML and Java via

SAX. The Java source code and a Java archive (jar) file (including a `Help` system) are available, as is extensive documentation and several examples.

EASIK opens with a graphical canvas with functionality for the creation of entities (nodes), attributes and edges joining entities. EA sketch constraints from several classes may be specified using the graphical interface. The components of the sketch are accessible from a text panel. A stored EA sketch may be loaded. The EA sketch information (entities, edges, attributes and constraints) in the graphical display can be edited on screen.

The displayed sketch may be saved to an XML document that encodes entities, attributes, edges and all of the constraints plus the current graphical display. The XML code follows a schema written in XSD. Fragments of XML code for the above example follow:

```
<entities>
<entity name="paper" x="365" y="228">
<attribute attributeType="VarChar(255)" name="Title"/> ...
<edges>
<edge id="of" inj="false" source="authorship" target="paper"/> ...
<commutativediagram isVisible="true" x="373" y="126">
<pathref id="among;of"/>
<pathref id="of_1;isA_3"/>...
```

Generation of data description language (SQL code) for a database schema from an EA sketch uses its stored XML document. The generation procedure begins by creating a table for each entity with keys derived from the graph of the EA sketch. A column of its entity table is created for each attribute. Each edge is encoded as a foreign key created for its source entity table on the primary key of its target entity table. The point is that, for any tuple in a source table (entity set) its value under the function implementing the edge is specified in the target table (entity set). The `is_a` (injective) edges also use `UNIQUE`. An example of generated SQL code follows. Note the attribute `Title` and the foreign key for `reader`.

```
CREATE TABLE paper (paper_id INTEGER AUTO_INCREMENT ,
Title VARCHAR(255), commMbr_id INTEGER NOT NULL ,
FOREIGN KEY (commMbr_id ) REFERENCES commMbr (commMbr_id ) ,
ON UPDATE CASCADE ON DELETE CASCADE ,
PRIMARY KEY (paper_id ));
```

Cascading updates and deletes for the foreign key is a design decision entailed by automatic generation.

The constraints of the EA sketch are coded as triggers and stored procedures. For example, inserting a tuple in the domain table of a commutative diagram invokes a trigger to enforce the requirement that the values the foreign keys used to express the participating edges do match after following the two paths in the commutative diagram. In the example following, the stored procedure `ProgCttee_commDiag0` is called to traverse the paths involved.

```
CREATE TRIGGER presenterAInsertTrig AFTER INSERT ON presenter
FOR EACH ROW BEGIN call ProgCttee_commDiag0(NEW.presenter_id );END
```

The stored procedure is fairly routine imperative code.

4 Conclusions

The challenging problem solved by this early version of EASIK is automatic compilation into SQL data definition language in a way that enforces the constraints of an EA sketch. Thus its powerful constraint definition facility is available to users via a simply understood graphical data model design tool. EASIK is the first system to do this. Furthermore EASIK will support the development of large systems using the SkDM approach. Such software support is vital with the large applications common in industrial practice. The system described has some limitations. Only a single EA sketch window may be opened. Limit cones are currently required to be one of two (very common) types. Testing with commercial DBMS has been limited. Future versions will address these issues.

Database theory has worked within its own Relational Algebra for many years, but enhancing the theory with categorical universal algebra is a more recent development. The system described here provides a positive link between theory (the algebraic methodology, SkDM) and practice (the software technology of database management systems).

References

1. Chen, P.P.S.: The Entity-Relationship Model—Toward a Unified View of Data. ACM Trans. Database Syst. 2, 9–36 (1976)
2. Dampney, C.N.G., Johnson, M.: Experience in developing interoperations among legacy information systems using partial reverse engineering. In: Proceedings of the IEEE International Conference on Software Maintenance, pp. 269–272 (2003)
3. Johnson, M., Rosebrugh, R.: Sketch data models, relational schema and data specifications. In: Proceedings of CATS 2002. Electronic Notes in Theoretical Computer Science, vol. 61(6), pp. 1–13 (2002)
4. Johnson, M., Rosebrugh, R.: Fibrations and universal view updatability. Theoretical Computer Science 388, 109–129 (2007)
5. Johnson, M., Rosebrugh, R.: Constant complements, reversibility and universal view updates. In: AMAST (to appear, 2008)
6. Shipman, D.: The functional data model and the data language DAPLEX. ACM Trans. Database Syst. 6, 140–173 (1981)
7. Walters, R.F.C.: Categories and Computer Science. Cambridge University Press, Cambridge (1991)
8. `http://en.wikipedia.org/wiki/Entity-relationship` (accessed on May 6, 2008)