Implementing database design (and manipulation) categorically

Bob Rosebrugh

Department of Mathematics and Computer Science
Mount Allison University

NIST Comp CT Workshop / 2015-09-29

Outline

Databases and CompCT?

- Databases have had strong theory since 1970's (Codd, et al)
- Database theory adopted CT early
  - commutative diagrams in Bancilhon/Spyratos (1981)
  - join query as a pullback (Akihiko 1983)
  - graph functor semantics (Lellahi-Spyratos 1991)
  - many categorical meta-models
    - Rosebrugh-Wood 1991 (Indexed categories)
    - Johnson-Dampney 1993 (Commutative diagrams)
    - Diskin et al 1995... (Sketch semantics)
    - Piessens-Steegmans 1995 (Sketch semantics)
    - Johnson-Rosebrugh-Wood 1996... (Sketch semantics)
    - Spivak et al, 2009... (Functor semantics)
    - and several others
- Multiple implementations
  - FQL (Spivak-Wisnesky 2012...)
  - Cateno?
  - EASIK (Rosebrugh et al 2006...)
  - others?

Aside: Database of Categories

- What about making categories themselves the objects in a database?
  - Database of Categories 1995: C program (later Java)
  - Idea: Store categories, and eventually functors
  - Queries on their properties e.g. limits
  - Re-implement Walters-Carmody Kan extension computation
- Development ended about 2005

Modern database systems

- Database systems today are self-described as “relational” or “object-relational”
- DBMS need to include:
  - storage management
  - query optimizers
  - journalling/backup and recovery
  - access control
- Popular systems include MySQL, Oracle, PostgreSQL
- SQL is the de facto standard language for *both*
  - Data definition: specify schema via tables/constraints
  - Query language: specify data entry and update

Database schema design

- Good design seeks to prevent redundancy, inconsistency in schemas
- *Data (meta-)models* prescribe what type *and* constraint information builds the schema
- Entity-Relationship-Attribute (ERA): the most popular
  - natural and simply understood type graph, very limited constraints
  - design diagram translates into relational database schema
  - commercial (and some free) apps allow creation of ERA diagrams, export to SQL
- Sketch Data Model (SkDM): types/constraints from a (mixed) EA sketch
  - natural and simply understood graphical paradigm: uses a type graph, finite limit and sum constraints
  - design sketch translates into relational database schema (with constraints)
  - EASIK creates EA sketches, generates database schemas and allows SQL export and DBMS connections for in-app data manipulation

Database states and queries

- Database state for a schema is the (currently) stored information
  - In the relational model this is a set of relations (tables)
  - ERA speaks of 'entity sets', 'relationship sets', usually aims at relational implementation
  - Models of EA sketches: functors preserving constraints
- Queries extract information from a database state, define access control and more
  - Relational queries expressed in "relational algebra", return relations
  - ERA has no query language
  - EA sketches have a theory: expresses queries

Sketch data model

- Database schema is syntax: a (mixed) sketch
- Database state is semantics: a model (functor)
- Models form a category
- Suitable class of sketches is Entity-Attribute (EA)

Sketch data model

- Database schema is syntax: a (mixed) sketch
- Database state is semantics: a model (functor)
- Models form a category
- Suitable class of sketches is Entity-Attribute (EA)

- Theory developed by M. Johnson, RR, RJ Wood; articles on our web pages
- Main contributions:
  - Criterion for view updatability via (op)cartesian arrows; leading to Bidirectional Transformation (BX) theory
  - Various models of incomplete information clarified
  - Translations among data meta-models

Mixed sketches and models

Mixed sketches: powerful syntax modeling tool
(defined by Ehresmann, 1960's)
Can describe multi-sorted algebra and more
Semantics (aka models) defines a category

Mixed sketches and models

Mixed sketches: powerful syntax modeling tool
(defined by Ehresmann, 1960's)
Can describe multi-sorted algebra and more
Semantics (aka models) defines a category
A mixed sketch $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ has

- G a (finite) *directed graph*
- $\mathbf{D}$ a set of *commuting diagrams*
- $\mathcal{L}$ a set of *cones*
- $\mathcal{C}$ a set of *cocones*

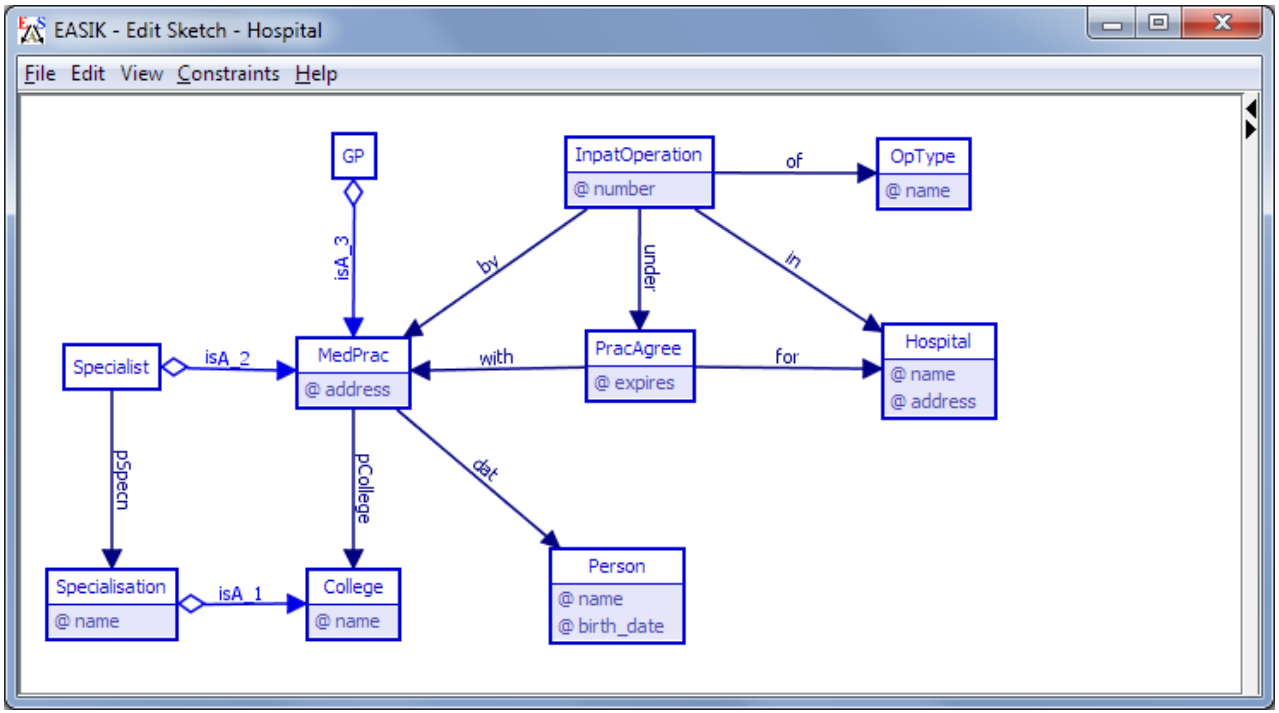
Mixed sketches and models

Mixed sketches: powerful syntax modeling tool
(defined by Ehresmann, 1960's)
Can describe multi-sorted algebra and more
Semantics (aka models) defines a category
A mixed sketch $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ has

- G a (finite) *directed graph*
- $\mathbf{D}$ a set of *commuting diagrams*
- $\mathcal{L}$ a set of *cones*
- $\mathcal{C}$ a set of *cocones*

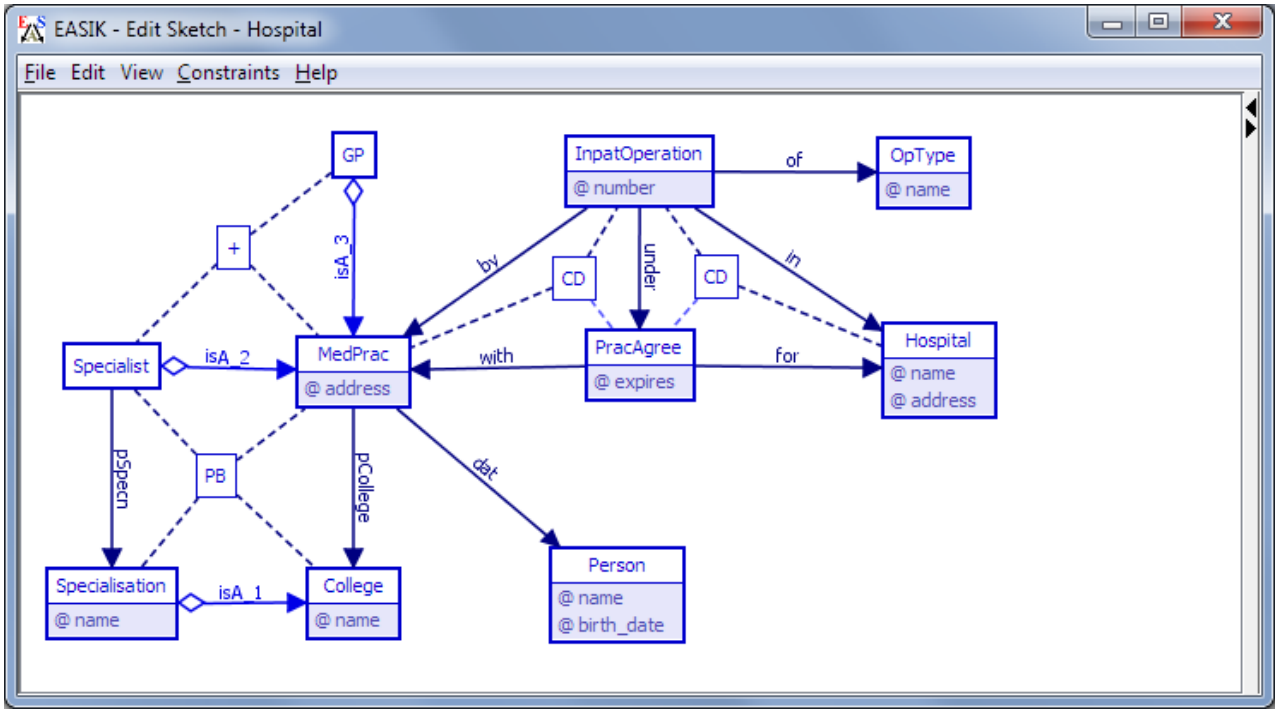
A model in a category $\mathbf{S}$ = a sketch morphism from $\mathbb{E}$ to $\mathbf{S}$:
i.e. a functor from $C(\mathbb{E})$ (category generated by $G, \mathbf{D}$) to $\mathbf{S}$
sending $\mathcal{L}, \mathcal{C}$ to (co)limit (co)cones

Example 1: Health Admin Database



Arrows for foreign keys;
Monos (including primary keys) are pullback constraints (not shown)

Example 1: Health Admin Database



ERA (and relational) data model cannot express:
Square, triangles commute; square is a pullback;
Medical Pract'ner $\cong$ GP $+$ Specialist

EA sketches

An *EA-sketch* $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ is a finite limit, finite sum sketch with

EA sketches

An *EA-sketch* $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ is a finite limit, finite sum sketch with

- G finite
- a specified empty-base cone in $\mathcal{L}$ (vertex is called 1); arrows with domain 1 called elements
- attributes are vertices of (sum) cocones of elements; non-attributes called entities
- An EA sketch is keyed if each entity E has a specified monic arrow $k_E : E \rightarrowtail A_E$ to an attribute A_E

EA sketches

An *EA-sketch* $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ is a finite limit, finite sum sketch with

- G finite
- a specified empty-base cone in $\mathcal{L}$ (vertex is called 1); arrows with domain 1 called elements
- attributes are vertices of (sum) cocones of elements; non-attributes called entities
- An EA sketch is keyed if each entity E has a specified monic arrow $k_E : E \rightarrowtail A_E$ to an attribute A_E

- $\mathbf{D}$, $\mathcal{L}$ and $\mathcal{C}$ express constraints often *not expressible* in other data models
- the main constraints of relational algebra are included

EASIK design and features

"Entity-Attribute Sketch Implementation Kit"

- Java for portability; design input through the GUI
- Target is automatic database schema definition in SQL
- Entities have system-generated primary key
- User may select a path of edges to create constraints
- Constraints representable graphically and are...
- Exported as triggers and procedures to the SQL schema
- Database schema (with constraints) exported to SQL
- Drivers for DBMS connection and manipulation included

EASIK Development

- Version 1 released 2006:
  - Java graphical interface for EA sketch design
  - Automatic SQL code generation
  - Database schema enforces the constraints
  - Exportable to a MySQL database

EASIK Development

- Version 1 released 2006:
  - Java graphical interface for EA sketch design
  - Automatic SQL code generation
  - Database schema enforces the constraints
  - Exportable to a MySQL database
- Version 2 released 2009:
  - Overview with multiple EA sketches
  - MySQL or PostgreSQL export
  - Data entry and retrieval (queries) within the interface (DBMS connections)
  - Basic support for views

EASIK Development

- Version 1 released 2006:
  - Java graphical interface for EA sketch design
  - Automatic SQL code generation
  - Database schema enforces the constraints
  - Exportable to a MySQL database
- Version 2 released 2009:
  - Overview with multiple EA sketches
  - MySQL or PostgreSQL export
  - Data entry and retrieval (queries) within the interface (DBMS connections)
  - Basic support for views
- Version 3 released March 2015:
  - More general views; can implement constraints
  - View updates restricted to known safe cases
  - Constraint handling better tested and refined
  - Demo to follow

EASIK modules

- graphical engine for point-click editing/display of an EA sketch
- generate an XML document that encodes the sketch
- compile the design XML document to SQL database schema
- connect to database server for database creation, data entry and retrieval
- *all accessible* via the GUI

Example 2

Conference program committee:

- information on cttee members, authors, articles submitted, status
- some of the rules:
  - person is an author or committee member (not both)
  - assignment of single committee member as first reader
  - one or more authors recorded as presenter
- do not show most attributes (can be added later)

EASIK screen...

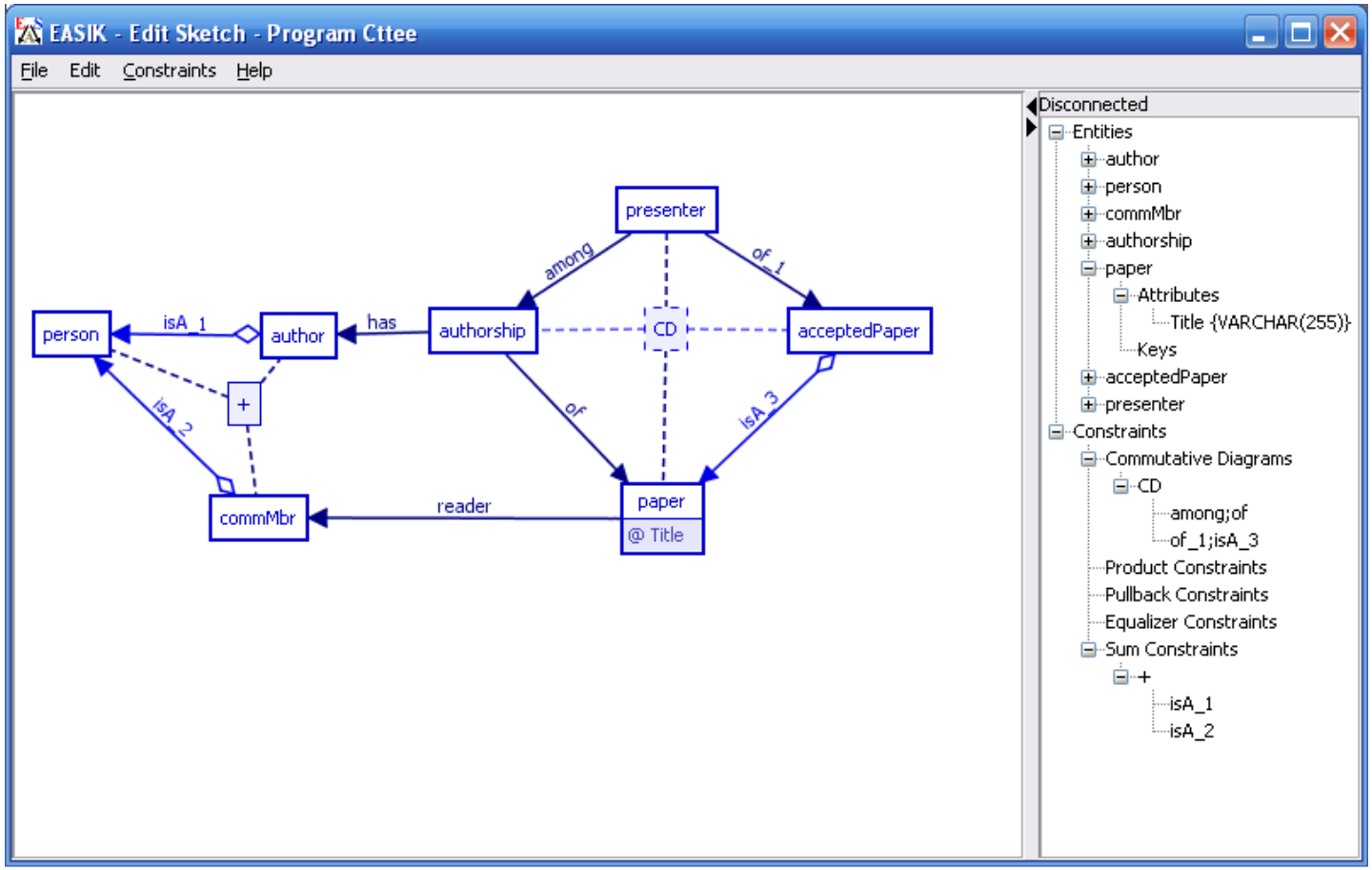


Figure: EASIK screen for part of a Program Committee database

EASIK demo

- Create a sketch from overview
- Add entities and edges
- Add constraints
- Export to SQL
- Constraint operation

XML export

At EASIK's center is the XML document produced
from the GUI image of the sketch.
This XML document has all sketch data, including constraints
(and incidentally rendering coordinates)
Hence can *encode*

- a finitely presented category **C** (as EA sketch)
- a functor from **C** to finite sets (as model)
- in the EASIK XML output
  - Entity is an `<entities>` element
  - Attributes are attributes of an entity
  - Edge is an `<edges>` element; `normal`, `injective` or `partial`
  - Path elements have `<edgerefs>`
  - Constraint elements have types (commutative diagram, pullback,...)

XML export

Example of generated XML for Program Committee:

First some header data

```
 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
<easketch>
<header>
<title>Program Cttee</title>
<author>M. Johnson</author>
<author>R. Rosebrugh</author>
<description/>
<creationDate>2008-05-07T14:59:00</creationDate>
...
</header>
```

XML export

Then entities and attributes...

```
 <entities>
<entity name="author" x="144" y="128"/>
<entity name="person" x="10" y="127"/>
<entity name="commMbr" x="114" y="235"/>
<entity name="authorship" x="243" y="125"/>
<entity name="paper" x="365" y="228">
<attribute attributeTypeClass="...Varchar" name="Title"
size="255"/>
</entity>
...
</entities>
```

XML export

Edges...

```
 <edges>
<edge cascade="cascade" id="of" source="authorship"
target="paper" type="normal"/>
<edge cascade="cascade" id="of_1" source="presenter"
target="acceptedPaper" type="normal"/>
<edge cascade="cascade" id="reader" source="paper"
target="commMbr" type="normal"/>
...
<edge cascade="cascade" id="isA_2" source="commMbr"
target="person" type="injective"/>
...
</edges>
```

XML export

and constraints...

```
 <constraints>
<commutativediagram isVisible="true" x="373" y="126">
<path codomain="paper" domain="presenter">
<edgeref id="among"/>
<edgeref id="of"/>
</path>
<path codomain="paper" domain="presenter">
<edgeref id="of_1"/>
<edgeref id="isA_3"/>
</path>
</commutativediagram>
<sumconstraint isVisible="true" x="125" y="169">
<path codomain="person" domain="author">
<edgeref id="isA_1"/>
</path>
...
</constraints>
</easketch>
```

SQL export basics

EASIK automatically generates SQL data definition
for export to text or a DBMS connection.

- Table for each entity; column for each attribute
- Edge is foreign key for source entity on primary key of target table
- Injective edges use `UNIQUE`

Example of generated SQL code:

```
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 ));
```

Enforcing EA constraints

EASIK SQL data definition includes (certain) limits/sums

- automatic generation of constraint enforcement (vs ad hoc post-definition)
- constraint code is triggers and stored procedures
- consistency, non-redundancy follow

Example: insert tuple in domain table of commutative diagram. Invokes trigger to ensure that values match after following two paths. Note call to `ProgCttee_commDiag0`

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

The stored procedure is routine imperative code.

EASIK and views

A difficult and much studied database issue:
When can an update to a view be *correctly* propagated?

EASIK implements views, and from Version 3 supports some correct updates.

- In the Sketch Data Model(SkDM), views are sketch morphisms
- SQL views are single tables; updatability very narrow
- EASIK views are (limited) SkDM views
  - So far essentially a subsketch
  - View entities may be SQL views on sketch entity
  - Some support for enforcing updatability

Summary

- Sketch Data Model has expressive syntax (EA sketch), semantics
- EASIK provides graphical interface for EA sketch design
- EASIK exports XML description of finitely presented category
- To do
  - improve view management
  - implement integration
  - refactor for potential users

Design/Development Team:

- Rob Fletcher - 2005, Vera Ranieri, Kevin Green - 2006
- Jason Rhinelander, Andrew Wood - 2008/9
- Christian Fiddick - 2012, Sara van der Laan - 2013, Federico Mora - 2014
- Bob Rosebrugh `www.mta.ca/~rrosebru`