ModelGen

1. ModelGen, before it was time, 1995-1997
2. ModelGen, an idea, January 2003
3. ModelGen, a partial implementation, November 2003

4/12/2003

A Multiple-Data-Model Approach to the Management of Heterogeneous Schemes

Paolo Atzeni, Riccardo Torlone

ca 1996

Overview

- Goal
  - a model-independent data dictionary
  - a component of an integrated (flexible, open) CASE tool
- Motivation
  - many data models exist

Many models

- with different features and goals
  - semantic models and logical models:
    - E-R, functional, (conceptual) object
    - relational, network, object
  - general purpose models (for all seasons) and problem oriented models (for specific contexts: DW, statistical, spatial, temporal)
- Variations of models
  - models with different levels of abstraction
  - versions within a family …

The E-R model

- It is not really a model, but a family of models:
  - choose n books (or methodologies or tools);
  - each will claim that adopts the Entity-Relationship model;
  - you will find m > n versions of the E-R model
- Indeed:
  - Binary vs n-ary
  - With or without attributes for relationships
  - With or without external identification
  - With or without generalizations (total vs partial, overlapping vs disjoint, …)

Why should we handle different models

- a methodology is chosen and a independent tool is available, and their models differ:
  - each CASE tool uses a different model
  - each methodology uses a different model
  - models in tools and methodologies are often not related (the ``impedance mismatch'')
- designers of a complex project work with their favorite models, but their work has to be exchanged, reused and integrated;
- specific sub-problems are handled with different models, specific for each
- the results of independent design activities have to be integrated

An ideal goal

- An environment that:
  - allows the definition of any possible model
  - given two models M1 and M2, and a scheme S1 of M1 (the **source** scheme and model), generates a scheme S2 of M2 (the **target** scheme and model), corresponding (**equivalent**) to S1

Is the goal realistic?

- what does “any possible model" mean?
- what does “corresponding" (or “equivalent") mean?

What does “any possible model" mean?

- each model has its own constructs
- each model gives the definition (the semantics) of the constructs in a different way
- each model introduces specific features that have no counterpart in other models

Could there be a “universal" description of models?

- first order logic?
- set theory?

- maybe, but how could we handle the descriptions?

What does “equivalent” mean?

- Various notions of equivalence have been proposed, different from one another
- Given a notion of equivalence, there are cases where, fixed the models and the source scheme, there is no equivalent scheme in the target model; example:
  - the source scheme is an E-R model with cardinality constraints and the target E-R model without the
- Also there are cases where there are two or more “corresponding“ target schemes; example:
  - the source scheme is an E-R model with is-a relationships and the target an E-R model without them

However, the situation is not that bad

- The constructs in the various models are rather similar: they can be classified into a small number of categories (“metaconstructs")
- That is:
  a metamodel approach

E-R model (a version)

- entity: objects in the domain of interest
- relationship: pairs (or n-tuples) of entities
- domain: set of values
- attribute : function from entities (or relationships) to domains

Relational model

- domain : set of values
- relation : subset of the cartesian product of domains (n-tuples) of values

Functional model (a version)

- domain : set of values
- object in the domain of interest
- function : from objects to objects or domains

A classification (Hull & King, 1987)

- Lexical types : sets of printable values
  - Domain
- Abstract types
  - Entity type , set of objects in the world
  - Class , set of objects in the system
- Aggregation : a construction based on (subsets of) cartesian products
  - Relationship in the E-R model
  - Relation in the relational model
- Function
  - Attribute in the E-R model
  - Function in a functional data model
- Grouping
- Hierarchies

Summarizing

- We can fix a set of metaconstructs of interest (each with a set of possible variants):
  - lexical, abstract, aggregation, function, ...
  - the set can be extended if needed, but this will not be frequent
- Then a model can be defined in terms of the metaconstructs its constructs refer to

Some models

- The relational model
  - a lexical construct, called domain
  - an (n-ary) aggregation construct, called relation
- (A simple version of) the E-R model
  - a lexical construct, called domain
  - an abstract construct, called entity
  - an (n-ary) aggregation construct, called relationship
  - a (monovalued monadic) function construct (from an entity or a relationship to a domain)

Translations in this framework

- The constructs corresponding to the same metaconstruct (e.g. entity in the E-R model and class in an object model both corresponding to abstract) have the same "meaning"
- Translations can refer to metaconstructs, rather than to constructs (which are model specific)
- A translation from a source model to a target model would have to replace constructs in the source (and not in the target) with constructs in the target
- Translations can be built by composing elementary transformations
  - each of them would eliminate some constructs (or "patterns" thereof) and possibly introduce new ones
  - a translation from a source model to a target one would eliminate some constructs and introduce new ones

Elementary steps

- There can be a set of predefined basic translations
- They are assumed to be correct
- We have studied properties of compositions of basic translations:
  - a correct translation from M1 and M2 produces schemes that contain only constructs that are allowed in M2

Examples of basic translations

- eliminate n-ary aggregations; replace them with binary ones (and abstracts)
- eliminate binary aggregations; replace them with functions
- eliminate functions to abstracts; replace them with aggregations
- eliminate complex attributes; replace them with simple attributes and abstracts

Translations, how many?

- If we have n different models, how many translations do we need?
  - apparently, n^2
  - in fact, only n, that is, one for each model, which eliminates the constructs that are not allowed

The Supermodel

- A model that includes all the constructs (in their most general forms)
  - each translation from the supermodel SM to a target model M eliminates all constructs that are not allowed M
  - therefore, each translation from SM to M is also a (possibly redundant) translation from any other model to M

Actors on the scene (and behind it)

- designers : define schemes within existing models
- model engineers : define models by using metaconstructs and generate (and modify) translations by composing basic translations
- metamodel engineers: extend the whole system, by defining new metaconstructs and the corresponding basic translations (a nontrivial task)

ModelGen e EDBT'96

Paolo Atzeni, Phil Bernstein

The goal

- An automated procedure is presented in [Atzeni & Torlone 96] for generating a script of transformations that translates a source schema in one metamodel into a target schema in another metamodel.
- We want to develop this work into a complete specification and implementation of the ModelGen operator proposed in [Bernstein 03].

Specific objectives

- have the script generate a well-behaved model management mapping from the source schema to the target schema
- allow users to customize the transformations that are produced
- allow users to customize the target schema and mapping
- ensure that the generated schemas and mappings can be used at least for the scenarios of [Bernstein 03]
- generate instance-level semantics for the script, which translates instances of the source schema into instances of the target schema.

Strategy

- bottom-up, by evaluating ER-to-SQL and SQL-to ER translations.
- translations written in Datalog, which enabled us to write schema translations in a (precise) textual form
- we checked that we can produce rules that do the work we expect of ModelGen, and we did walkthroughs of those rules for some small prototypical examples.
- we have developed sufficient confidence in the examples that we are now ready to address the overall goal, namely, using the framework of [Atzeni & Torlone 96] to produce a specification and implementation of ModelGen.
- we have therefore started translating the most general metamodel, i.e. supermodel, of [Atzeni & Torlone 96] into the relational and datalog framework that we are currently pursuing.

"Results"

- metamodels for ER (various versions), relational (SQL), ADM
- mapping rules for tranforming an ER schema into a SQL schema
- mapping rules for (reverse engineering) a SQL schema into an ER schema
- mapping rules from ADM to ER (not yet the recursive ones, but reasonable)
- a description of the Supermodel, which is a metamodel that generalizes all of the metamodels of interest, based on the framework of [Atzeni & Torlone 96].

"Open issues"

1. A better understanding of the notion of mapping is needed in order to reconcile the "CIDR" framework with the "EDBT" one
2. Instance level tranformations are by no means trivial in the "automatic generation" framework, but they would be essential for understanding what mappings are in a general setting (for example when dealing with complex views); however, one could aim at the "syntactic verification" of instance level transformations (that is, check that what they generate is coherent with the metamodel level; Torlone proposed to use "simulation"; much work is probably needed)

"Open issues", 2

- A notion of completeness would be needed: let us say that
  - syntactic correctness means that a transformation produces only constructs that are allowed in the target model
  - semantic correctness means that the transformation produces "equivalent" schemas
  - completeness could mean that the translation uses the allowed constructs in the "best" way (e.g.: generates ternary relationships if the model allows)

Open issues, 3

- Customization is a way open problem, and could be implemented in various way (all offered at the same time):
  - at the metamodel level (all constructs of a given family)
  - at the schema level (a specific piece gets modeled in a certain way)
  - by choosing among alternatives, or by rewriting (in the latter case, which would be the language for rewriting?)

ModelGen: an implementation

Paolo Atzeni, Paolo Cappellari

"Results" in January

- metamodels for ER (various versions), relational (SQL), ADM
- mapping rules for tranforming an ER schema into a SQL schema
- mapping rules for (reverse engineering) a SQL schema into an ER schema
- mapping rules from ADM to ER (not yet the recursive ones, but reasonable)
- a description of the Supermodel, which is a metamodel that generalizes all of the metamodels of interest, based on the framework of [Atzeni & Torlone 96].

Problems

- Rules were written in a "Datalog with OID invention"
  - rather high level
  - not implemented (Bernstein tried a ConceptBase implementation, but did not work well)
  - control over rule applications would be needed

Solution idea

- rules written in SQL
- Skolem functions stored in relational databases,
  - OID invention simulated via "autoincrement" fields (alternatives exist)
- control could be separated from rules (and we ignore most of the sophisticated aspects for now: choosing rules, verifying "soundness", minimality, …)

- experiment in March, 2003
  - relational database
  - SQL rules

Problems with the solution idea

- Rules have a lot of "bureaucracy"
  - source and destination scheme
  - Skolem rules and Skolem tables
- There is even more bureaucracy in the supermodel environment
  - we need copies between the supermodel and individual models and vice versa
  - we also need "copies" within the supermodel if we have elementary rules that handle single constructs (e.g. if we eliminate n-ary relationships, entities and their attributed had to be simply copied)

Towards a more feasible solution

- The goal:
  - remove the bureaucracy
  - "automate" as much as possible

Remove the bureaucracy

- We would like rules that are as close as possible to the Datalog form

```
Rel_ColumnOfTable(OID(aOID), OID(eOID), AttrName, nullable) :-
     ER_AttributeOfEntity(aOID, AttrName, eOID, nullable)
```

```
INSERT INTO Rel_ColumnOfTable (
            columnOID, name, tableOID,
            isNullable, isKey, schemaOID)
  SELECT    var2.columnOID ,
            var1.name ,
            var3.tableOID ,
            var1.isNullable,
            var1.isID,
            ?1 AS schemaOID
 FROM       ER_AttributeOfEntity var1,
            Rel_ColumnOfTable_SK var2,
            Rel_Table_SK var3
WHERE       var1.attributeOID = var2.attributeOID
   AND      var1.entityOID = var3.entityOID
   AND      var1.schemaOID = ?2
   AND      var2.schemaOID = ?1
   AND      var3.schemaOID = ?1
```

```
INSERT INTO Rel_ColumnOfTable (
            columnOID, name, tableOID,
            isNullable, isKey)
  SELECT    columnOID_1 (var1.attributeOID) ,
            var1.name ,
            tableOID_1(var1.entityOID) ,
            var1.isNullable,
            var1.isID
FROM        ER_AttributeOfEntity var1
```

Remove the bureaucracy, 2

- Skolem functors can be automatically generated
  - expansion of the term
  - addition of variable in the range list
  - addition of condition
- This is feasible if we have a description of Skolem functions
- If so, we also have that
  - rules that insert tuples in Skolem tables can be generated

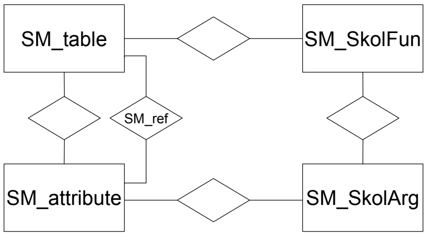
Remove the bureaucracy, 3

- "Copy rules" can also be automatically generated if we have a description of the models

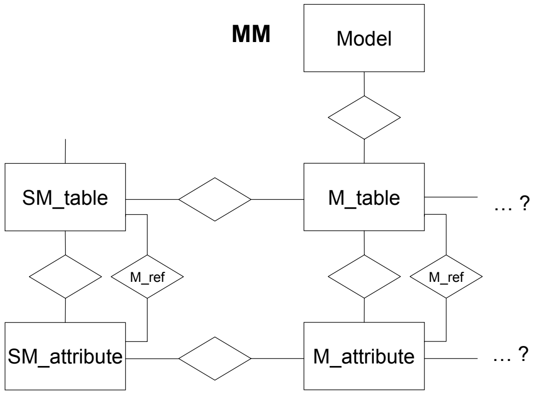
The dictionary

- MSM: a description of the supermodel (the "metasupermodel")
- SM: the supermodel (as described by MSM)
- MM: a descritpion of the models (the metamodel), each defined in terms of MSM constructs
- M: the models (as described by MM)

MSM



MM



The dictionary, in relational terms

- MSM_
  - The supermodel dictionary (4-5 tables, including MSM_Tables)
- SM_
  - The supermodel tables (a table for each tuple of MSM_Tables)
- Models
  - A tuple for each model
- MM_
  - The dictionary of models, similar to and related to MSM
    - Tuples in MM tables are, for each model "subsets and projections" (or joins, in the future) of those in MSM, plus references to Models
- M_XX_
  - Model tables (a table for each tuple in MM_tables)

Lo strumento: definizione di un modello

- Inserimento di tupla in Models
- “popolamento” di MM con riferimento da una parte alla tupla inserita in Models e dall’altra a costrutti del SM (cioè a tuple in MSM)
- Generazione di M_XX a partire da MM

Lo strumento: il processo di traduzione

1. Traduzione (o meglio, copia) verso il SM
2. Traduzione vera e propria (applicazione di sequenze regole che eliminano i costrutti non ammissibili nel modello target sostituendoli con costrutti previsti); richiede:
   1. una disponibilità di regole di traduzione elementari
   2. una capacità di “ragionamento” riguardo alla selezione delle regole
   3. la capacità di applicare sequenze di regole
3. Traduzione (o meglio copia) verso il modello target

Note

- Definizione modelli
  - passi 1 e 2: serve sostanzialmente una procedura interattiva (e una batch) di inserimento dei dati; essenziale è la verifica (esplicita o implicita) dei vincoli di riferimento;
  - passo 3: procedura di generazione (genera istruzioni SQL di creazione e le esegue)
- Traduzione
  - passi 1 e 3: ok o quasi (unici dubbi sul supermodello sofisticato, che potrebbe essere rinviato)
  - passo 2: è il cuore del sistema
    - 2.1 richiede la scrittura di regole, e qui la rappresentazione ad alto livello è essenziale
    - 2.2: è necessario individuare modalità di descrizione sintetica delle regole e dei modelli
    - 2.3: è sostanzialmente disponibile

More …

- Gestione del test
- Salvataggio e ripristino
- Report
- Interfaccia grafica …