MDM: a Multiple-Data-Model Tool for the Management of Heterogeneous Database Schemes

Paolo Atzeni
Riccardo Torlone

DDS, Sezione Informatica — Università di Roma Tre
Via Vasca Navale, 84 — 00146 Roma Italy
email: atzeni@dis.uniroma1.it
torlone@iasi.rm.cnr.it

November 1996

Part I

Concepts

Goal

- a model-independent data dictionary
- a component of an integrated (flexible, open) CASE tool

Motivation

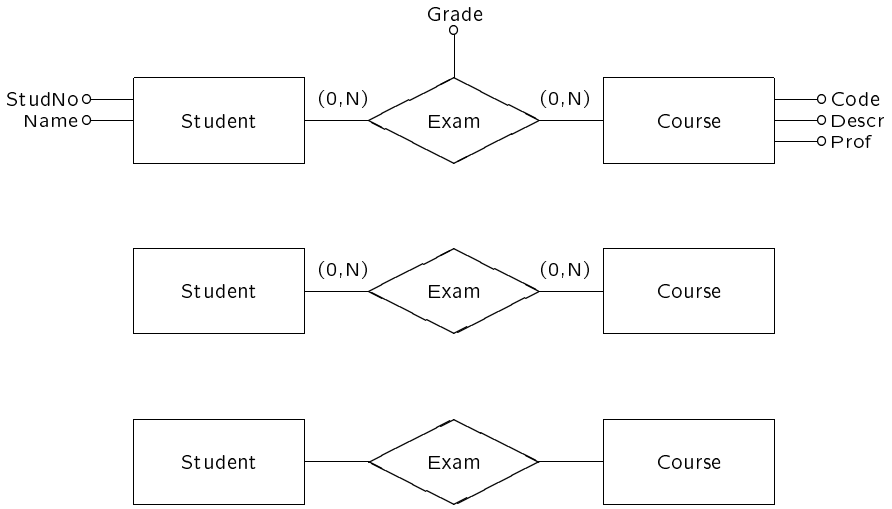
- many data models exist

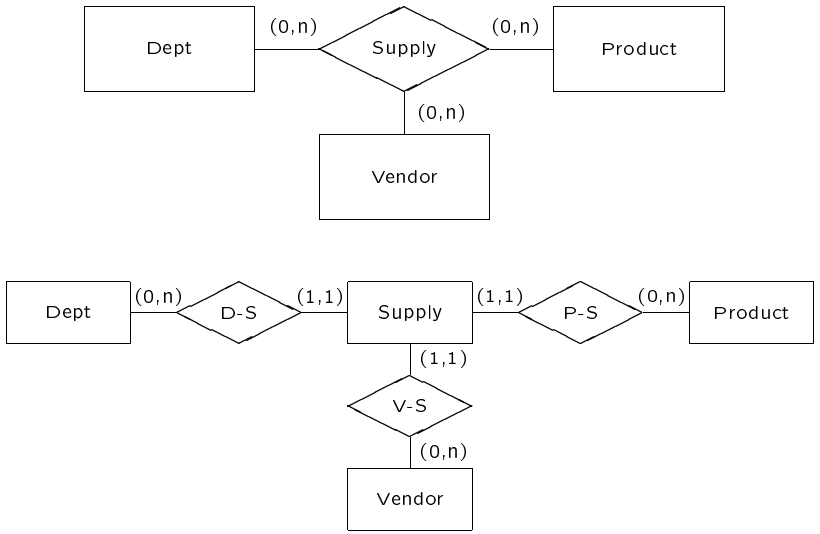
many models, with different features and goals:

- 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 applications; example: statistical)
- models with different levels of abstraction
- variations of models, with different structuring capabilities

even within the most common model (better, family of models):

- choose $\mathbf{n}$ books (or methodologies or tools);
  each will claim that adopts the Entity-Relationship model;
  you will find $\mathbf{m} > \mathbf{n}$ versions of the E-R model





reasons for having to 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 favourite models, but their work has to be exchanged, reused and integrated;
- specific subproblems 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 $\mathbf{M}_1$ and $\mathbf{M}_2$, and a scheme $\mathbf{S}_1$ of $\mathbf{M}_1$ (the **source** scheme and model), generates a scheme $\mathbf{S}_2$ of $\mathbf{M}_2$ (the **target** scheme and model), corresponding (**equivalent**) to $\mathbf{S}_1$

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

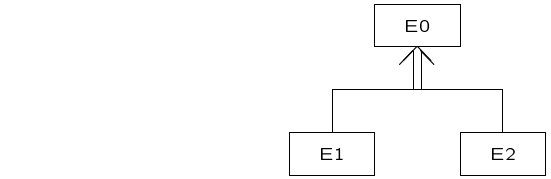
could there be a "universal" description of models?

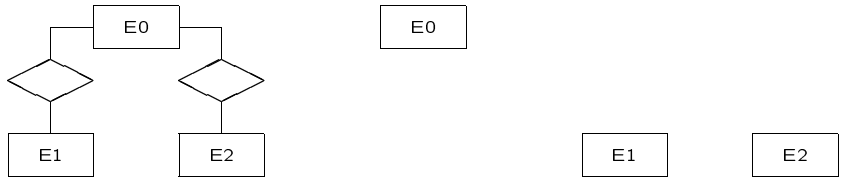
- first order logic?
- set theory?
- maybe, but how could we handle the descriptions?

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;
  for example, the source scheme is an E-R model with cardinality constraints and the target E-R model without them
- or there are cases where there are two or more "corresponding" target schemes;
  for example, the source scheme is an E-R model with is-a relationships and the target an E-R model without them

various choices





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):

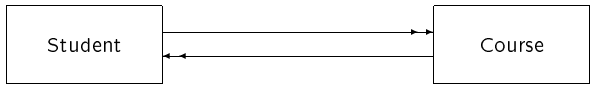
- **lexical types**: sets of printable values
  - domain
- **abstract types**
  - *entity type*, set of objects in the world
  - *class*, set of objects in the system
- **aggregation**: a construction based on (subsets of) cartesian products
  - *relationship* in the E-R model
  - *relation* in the relational model
- **function**
  - *attribute* in the E-R model
  - *function* in a functional data model
- **grouping**
- **hierarchies**

Therefore

- we can fix a set of metaconstructs of interest (each with a set of possible variants; will clarify later):
  - 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

- 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)

- we can assume that the constructs corresponding to the same metaconstruct (for example, **entity** in the E-R model and **class** in a complex object model both corresponding to **abstract**) have the same "meaning"



- therefore, 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 and possibly introduce new ones
- a translation from a source model to a target one would eliminate some constructs (in the source model and not in the target one) and introduce new ones (allowed in the target model)

- more precisely, translations operate (eliminate, introduce) on groups of constructs ("**patterns**")

- 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 $\mathbf{M}_1$ and $\mathbf{M}_2$ produces schemes that contain only constructs that are allowed in $\mathbf{M}_2$

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

- if we have $\mathbf{n}$ different models, how many translations do we need?
- apparently, $\mathbf{n}^2$
- in fact, only $\mathbf{n}$, 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 $\mathbf{M}_{\text{SUPER}}$ to a target model $\mathbf{M}_{\text{TARGET}}$ eliminates all constructs that are not allowed in $\mathbf{M}_{\text{TARGET}}$
- therefore, each translation from $\mathbf{M}_{\text{SUPER}}$ to $\mathbf{M}_{\text{TARGET}}$ is also a (possibly redundant) translation from any other model to $\mathbf{M}_{\text{TARGET}}$

- details in the description of the tool and of the application

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)