Management of schema translations in a model generic framework

Paolo Atzeni
Università Roma Tre

Joint work with
Paolo Cappellari, Giorgio Gianforme (Università Roma Tre)
and Phil Bernstein (Microsoft Research)
partially based on a paper in the proceedings of EDBT 2006

San Rafael Glacier, November, 2006

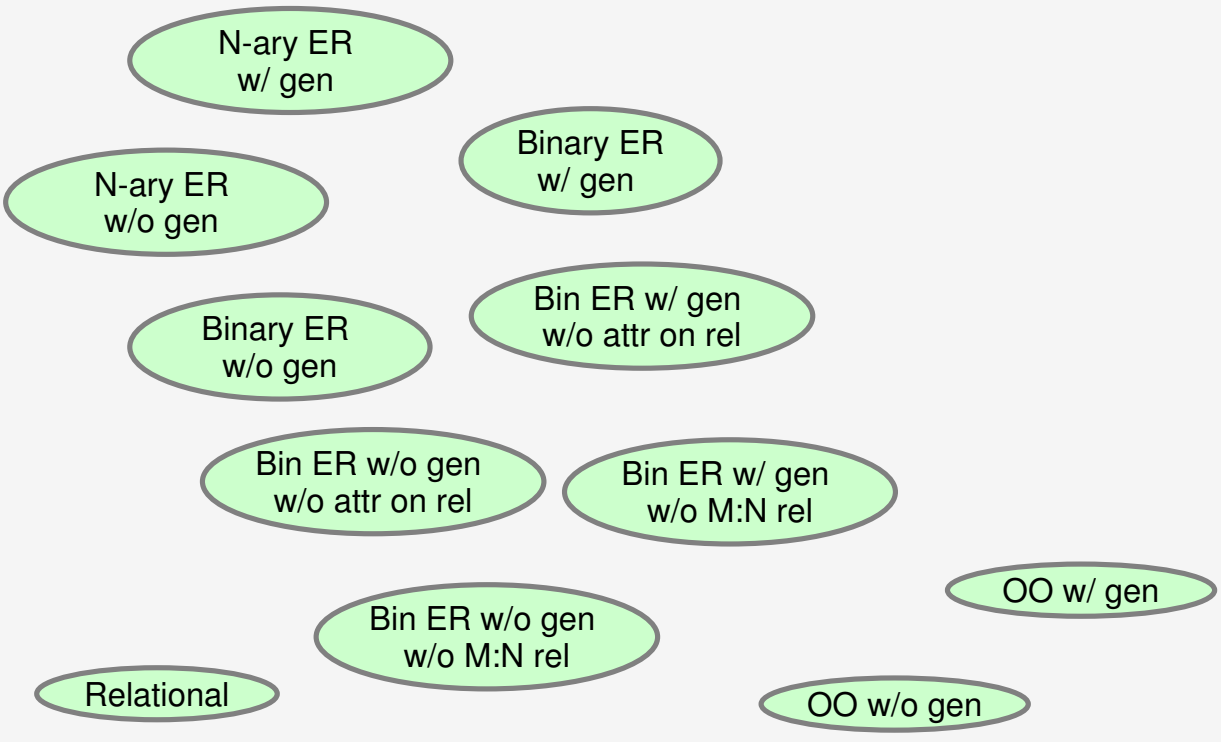
The problem

- **ModelGen** (a model management operator)
  - given two data models M1 and M2, and a schema S1 of M1 (the ***source*** schema and model),
    - generate a schema S2 of M2 (the ***target*** schema and model), corresponding to S1
    - and, for each database D1 over S1, generate an equivalent database D2 over S2

Old and new work

- Previous work on ModelGen exists (Atzeni & Torlone, 1996)
- New work (EDBT paper and more)
  - translation of both schemas **and data**
    - data-level translations generated, from schema-level ones
  - a visible, multilevel and (in part) self-generating dictionary
  - high-level, visible and customizable translation rules in Datalog with OID-invention
  - mappings between elements generated as a by-product (materialization of Skolem functions)
  - reasoning techniques on models and rules

Many different models



A metamodel approach

- The constructs in the various models are rather similar:
  - can be classified into a few categories (Hull & King 1986):
    - Lexical: set of printable values (domain)
    - Abstract (entity, class, …)
    - Aggregation: a construction based on (subsets of) cartesian products (relationship, table)
    - Function (attribute, property)
    - Hierarchies
    - …
- We can fix a set of metaconstructs (each with variants):
  - lexical, abstract, aggregation, function, ...
  - the set can be extended if needed, but this will not be frequent
- A model is defined in terms of the metaconstructs it uses

The metamodel approach, example

- The ER model:
  - Abstract (called Entity)
  - Function from Abstract to Lexical (Attribute)
  - Aggregation of abstracts (Relationship)
  - …
- The OR model:
  - Abstract (Table with ID)
  - Function from Abstract to Lexical (value-based Attribute)
  - Function from Abstract to Abstract (reference Attribute)
  - Aggregation of lexicals (value-based Table)
  - Component of Aggregation of Lexicals (Column)
  - …

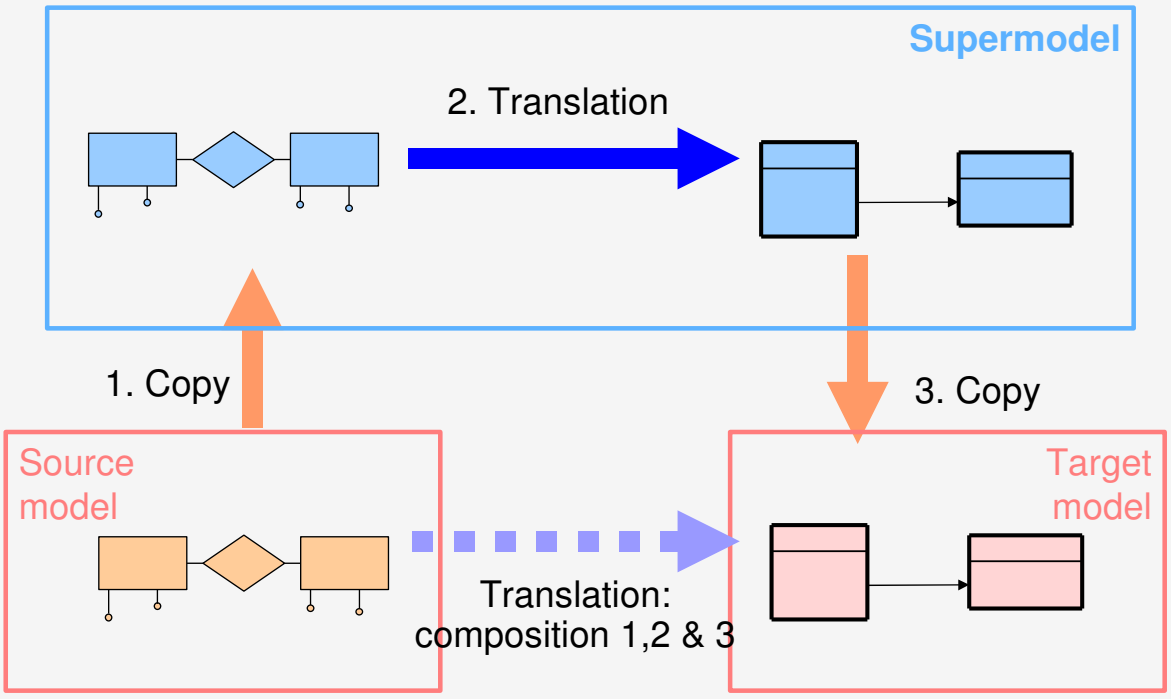
The supermodel

- A model that includes all the meta-constructs (in their most general forms)
  - Each model is subsumed by the supermodel (modulo construct renaming)
  - Each schema for any model is also a schema for the supermodel (modulo construct renaming)

The metamodel approach, translations

- The constructs in the various models are rather similar:
  - can be classified into a few categories (“metaconstructs")
  - translations can be defined on metaconstructs,
    - and there are “standard”, accepted ways to deal with translations of metaconstructs
    - they can be performed within the supermodel
  - each translation from the supermodel SM to a target model M is also a translation from any other model to M:
    - given n models, we need n translations, not n^2

Generic translation environment



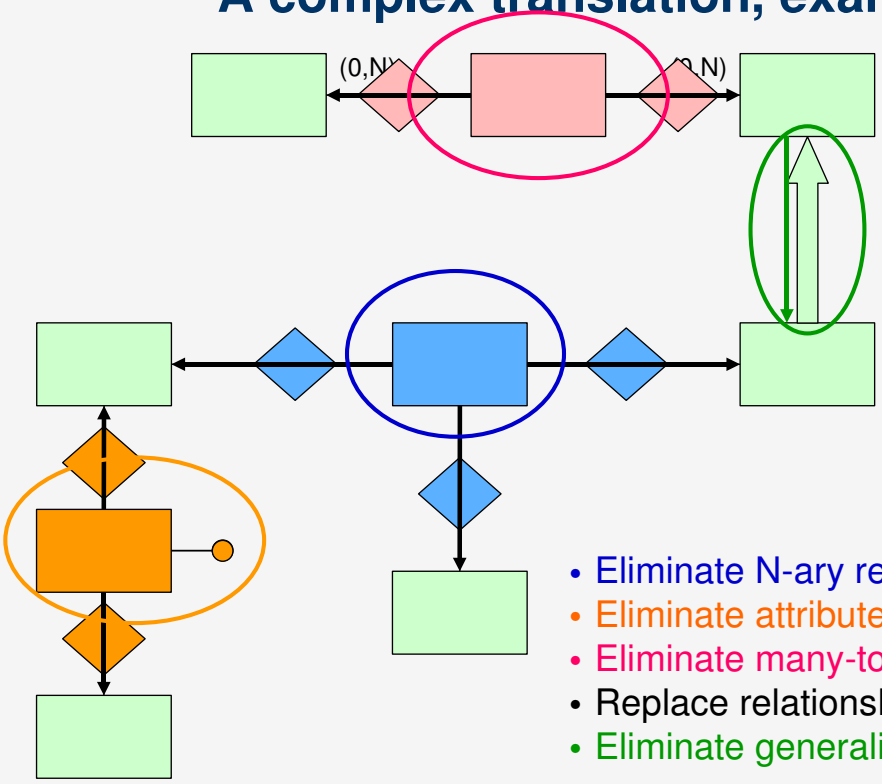
Translations within the supermodel

- We still have too many models:
  - Combining all variants of constructs within few families of models (e.g., ER), we get hundreds of models!
  - The management of a specific translation for each model would be hopeless

Translations, the approach

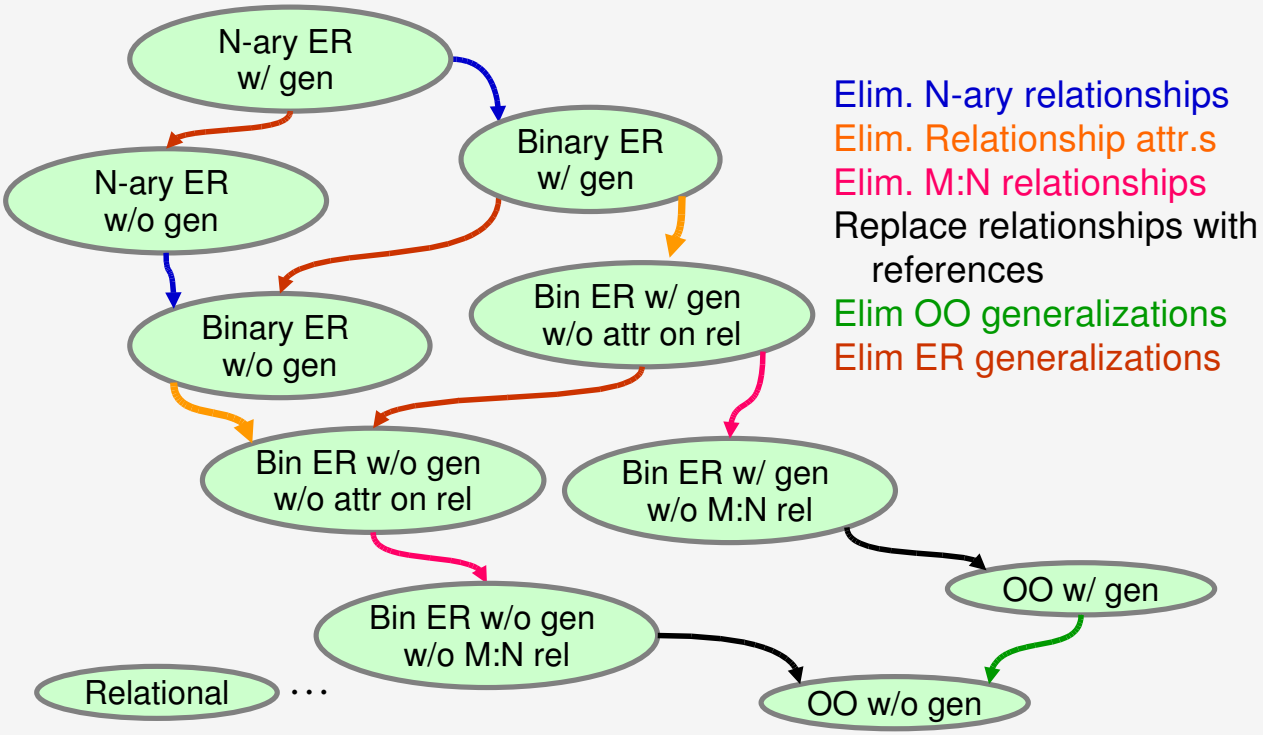
- Elementary translation steps to be combined
- Each translation step handles a supermodel construct (or a feature thereof) "to be eliminated" or "transformed"
- A translation is the concatenation of elementary translation steps

A complex translation, example



- Eliminate N-ary relationships
- Eliminate attributes from relationships
- Eliminate many-to-many relationships
- Replace relationships with references
- Eliminate generalizations

Complex translations



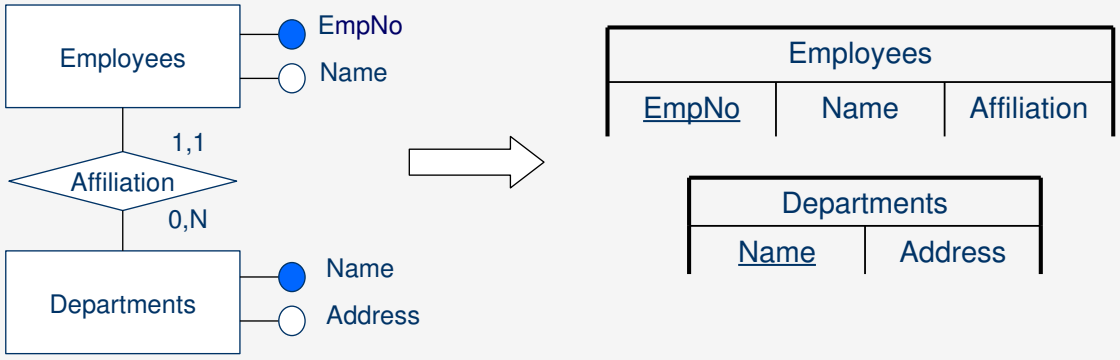
Translations

- Basic translations are written in a variant of Datalog, with OID invention
  - We specify them at the schema level
  - The tool "translates them down" to the data level
  - Some completion or tuning may be needed

A basic translation

- From (a simple) binary ER model to the relational model
  - a table for each entity
  - a column (in the table for E) for each attribute of an entity E
  - for each M:N relationship
    - a table for the relationship
    - columns …
  - for each 1:N and 1:1 relationship:
    - a column for each attribute of the identifier …

A basic translation application



A basic translation (in supermodel terms)

- From (a simple) binary ER model to the relational model
  - a table for each entity
  - a column (in the table for E) for each attribute of an entity E
  - for each M:N relationship
    - a table for the relationship
    - columns …
  - for each 1:N and 1:1 relationship:
    - a column for each attribute of the identifier …

Datalog with OID invention

- Datalog:
  - …
  - we use a non-positional notation
- Datalog with OID invention:
  - an extension of Datalog that uses Skolem functions to generate new identifiers when needed
- Skolem functions:
  - injective functions that generate "new" values (values that do not appear anywhere else; so different Skolem functions have disjoint ranges)

"An aggregation of lexicals for each abstract"

```
SM_AggregationOfLexicals(
    OID: #aggregationOID_1(OID),
    Name: n)
←
SM_Abstract (
    OID: OID,
    Name: n) ;
```

- the value for the attribute Name is copied (by using variable n)
- the value for OID is "invented": a new value for the function #aggregationOID_1(OID) for each different value of OID, so a different value for each value of SM_Abstract.OID
- the materialization of the Skolem function describes the mapping

"A component of the aggregation for each attribute of abstract"

```
SM_ComponentOfAggregation… (
    OID: #componentOID_1(attOID),
    Name: name,
    AggrOID: #aggregationOID_1(absOID),
    IsNullable: isNullable,
    IsKey: isIdent,
    Type : type )

SM_AttributeOfAbstract(
    OID: attOID,
    Name: name,
    AbstractOID: absOID,
    IsIdent: isIdent,
    IsNullable: isNullable ,
    Type : type ) ;
```

- Skolem functions
  - are functions
  - are injective
  - have disjoint ranges
- the first function "generates" a new value
- the second "reuses" the value generated by the first rule

Correctness

- Usually modelled in terms of information capacity equivalence/dominance (Atzeni+ 1982, Hull 1986, Miller+ 1993, 1994)
- Mainly negative results in practical settings that are non-trivial
- Probably hopeless to have correctness in general
- We follow an "axiomatic" approach:
  - We have to verify the correctness of the basic translations, and then infer that of complex ones

Reasoning on rules

- Given a source model and a rules,
  - what is the model to which the application of the rule to a source scheme belongs?

Reasoning on rules

- In the simple example, with the two rules:
  - The source contains (at least) entities, attributes, and relationships
  - The target contains tables and columns
- However:
  - If the source has nulls forbidden for the attributes, then nulls would not appear in the target as well!

Reasoning on models and rules, in short

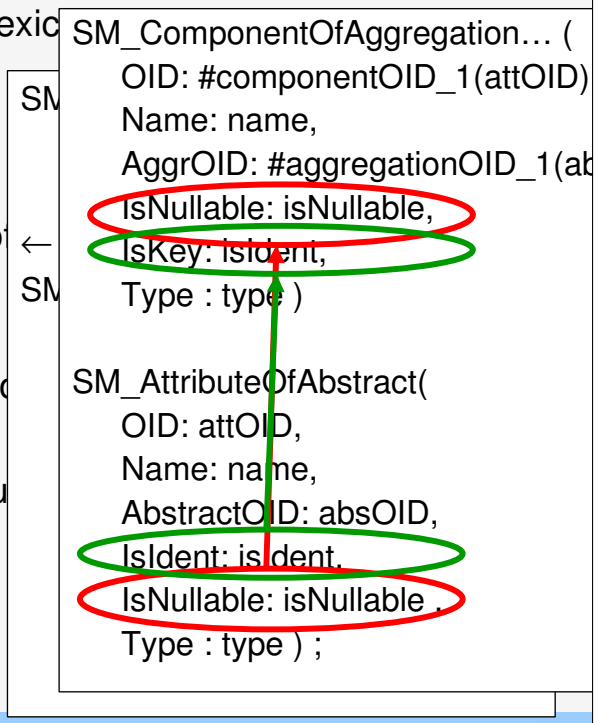
- A model
  - a set of constructs out of a universe,
  - each with a condition on its possible properties
- A Datalog rule has a signature that specifies
  - the body (i.e., the applicability of the rule)
  - the head (the construct it generates, with specific properties, if any)
  - the mapping (a description of where in the body values in the head originate from)

Models

- A universe of contstructs:
  - Abstract
  - AttributeOfAbstract(isIdent,isNullable)
  - AggregationOfAbstracts(…)
  - AggregationOfLexical
  - ComponentOfAggregationOflexicals(isKey,isNullable)
- The relational model
  - AggregationOfLexicals
  - ComponentOfAggregationOflexicals
- The relational model with no nulls
  - AggregationOfLexicals
  - ComponentOfAggregationOflexicals(not isNullable)

Rules

- The first rule ("An aggregation of lexic
  - Body: Abstract
  - Head: AggregationOfLexicals
  - Mapping: empty
- The second rule (" A component o
  attribute of abstract ")
  - Body: AttributeOfAbstract
  - Head: ComponentOfAggregati
  - Mapping:
    - Attr.IsNullable->Comp.IsNu
    - Attr.IsIdent -> Comp.IsKey



Results

- We have a notion of application of the signature of a rule to a model
- We can extract signatures from Datalog rules
- Main result:
  - The result of the application of the signature of a rule exactly characterizes the constructs (and the properties thereof) that can be generated by means of the Datalog rule
  - The result of the application of the signature of a program exactly characterizes the constructs that can be generated by means of the Datalog program

Summary

- ModelGen was studied a few years ago
- New interest on it within the "Model management" framework
- New approach
  - Translation of schema and **data**
  - Visible (and in part self generated) dictionary
  - Visible and modifiable rules, **which allow for reasoning**
  - Skolem functions describe mappings