Runtime Schema And Data Translation

Giorgio Gianforme, PhD

Università Roma Tre

Model Management

Highlights

- New approach to metadata management
- Models and mappings are abstractions
  - They can be manipulated by model-at-a-time and mapping-at-a-time operators
- Generic approach
  - Operators can be applied to any model and mapping
  - Single implementation of operators

ModelGen

Why?

- Humans need models to represent every kind of knowledge
- To share knowledge humans have to unify models

What?

- Given two data models M_1 and M_2 and a schema S_1 of M_1 (source schema and model), we generate a schema S_2 of M_2 (the target schema and model), corresponding to S_1 and, for each database D_1 over S_1, we generate an equivalent database D_2 over S_2

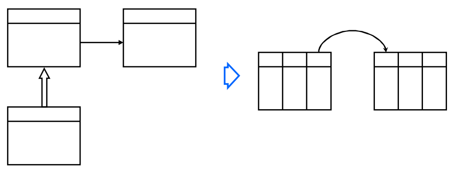
How?

- We use a framework that allows the definition of any possible model and the definition of translations from a model to another

Scenario

Example

- Given an Object-Oriented schema, we generate a corresponding Relational schema

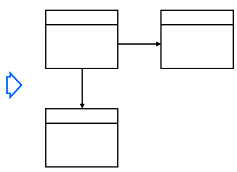


Scenario

Example

- Given an XML file, conform to an XSD file, we generate a corresponding Object-Relational schema

```
<?xml … >
< … >
 <tag1>
  <tag2> value2 </tag2>
  <tag3>
   <tag4> value4 </tag4>
   <tag5> value5 </tag5>
  </tag3>
  …
 </tag1>
</…>
```



Uniform Representation

Model

- Abstract representation of the domain

Construct

- A construct represents a concept of the domain

MetaModel

- Description of a model in terms of:
  - Constructs constituting it
  - Constraints on constructs
  - Relationships between constructs

Uniform and Generic Representation

SuperModel

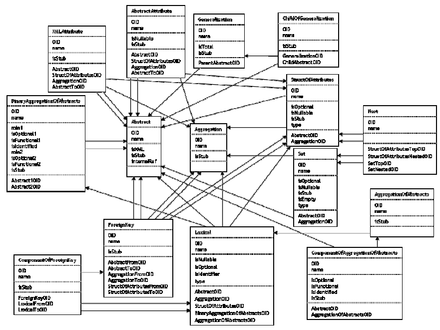
- ✓ Constructs in the various models are rather similar and can be classified into a few categories
- ✓ Reduced set of constructs
- ✓ Supermodel is a model that includes all constructs

Translation

- ✓ Can be defined on constructs
- ✓ Elementary translations to be combined

Uniform and Generic Representation

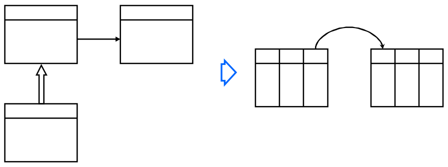
SuperModel



Translation

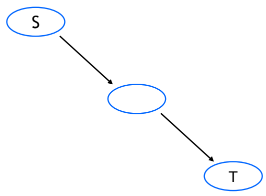
Example

- ✓ Object-Oriented to Relational
  - Eliminate generalization
  - Transform classes to tables, fields to columns, and references to foreign keys



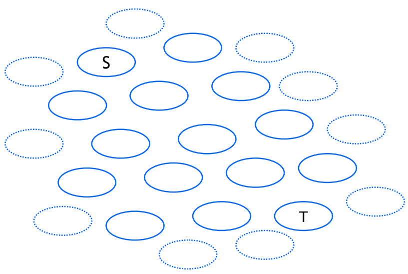
Space of Models

Which Direction?



Space of Models

Which Direction?



Details

Our Constructs

- ✓ Oid, name, boolean properties and references

Our Translation Rules

- ✓ Datalog with oid invention (Skolem functors)
  - Names of predicates are names of constructs
  - Names of arguments may be OIDs, names, names of references and properties

Remarks

- ✓ References between constructs are mandatory
- ✓ Oids are necessary “only” for technical reasons
- ✓ Constraints on boolean properties are well represented by propositional formula

Constructs

Simple Entity Relationship

- ✓ One construct for entities
  - No properties and no references
- ✓ One construct for attributes of entities
  - Properties: is identifier, is nullable
  - A reference towards the corresponding entity
- ✓ One construct for binary relationships
  - Properties for minimum and maximum cardinalities
  - Two references towards the involved entities

Datalog

Examples

- ✓ Whenever B, produce H
  - H ← B
- ✓ Generate a new Aggregation for each Abstract
  - Aggregation (
    OID: SK1(oid),
    Name: name)
    ←
    Abstract (
    OID: oid,
    Name: name);

Datalog

Examples

- ✓ Generate a new Lexical of Aggregation for each Lexical of Abstract
  - Lexical (
    OID: SK2(oid),
    Name: name, isIdentifier: isId, …,
    AggregationOID: SK1(absOid))
    ←
    Lexical (
    OID: oid,
    Name: name, isIdentifier: isId, …,
    AbstractOID: absOID),
    Abstract (
    OID: absOID);

Reasoning

Idea

- ✓ To make system able to reason on models automatically, we can use a *compact* representation of models and rules

Models

- ✓ Set of constructs with an associated formula
  - ER with no null values and no attributes on relationships:
    ER* = { Entity (*true*), Relationship (*true*),
    attributeOfEntity (NOT isNullable),
    attributeOfRelationship (*false*) }

Reasoning

Rules

- ✓ Signature of rule to describe:
  - Applicability
  - Result of application
  - Mapping between input and output

A (OID: …
a_1: *true*,
a_2: *x*)
←
B (OID: …
b_1: *true*,
cOID: *y*),
C (OID: *y*,
c_1: *false*,
c_2: *x*);

Reasoning

Rules

- ✓ Signature of rule to describe:
  - Applicability
  - Result of application
  - Mapping between input and output

A (OID: …
a_1: *true*,
a_2: *x*)
←
B (OID: …
b_1: *true*,
cOID: *y*),
C (OID: *y*,
c_1: *false*,
c_2: *x*);

Reasoning

Rules

- ✓ Signature of rule to describe:
  - Applicability
  - Result of application
  - Mapping between input and output

A (OID: …,
a_1: *true*,
a_2: *x*)
←
B (OID: …
b_1: *true*,
cOID: *y*),
C (OID: *y*,
c_1: *false*,
c_2: *x*);

Reasoning

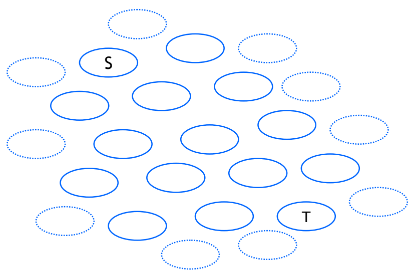
Rules

- ✓ Signature of rule to describe:
  - Applicability
  - Result of application
  - Mapping between input and output

A (OID: …
a_1: *true*,
a_2: *x*)
←
B (OID: …
b_1: *true*,
cOID: *y*),
C (OID: *y*,
c_1: *false*,
c_2: *x*);

Space of Models

Which Direction?



Reasoning

Formal System

- ✓ Compact representation of models and rules
  - Based on logical formulas
- ✓ Reasoning on data models
  - Union, intersection, difference of models and schemas
  - Applicability and application of rules and programs
- ✓ Sound and complete
  with respect to the Datalog programs
  - The application of a program to a schema can generate all (completeness) and only (soundness) constructs belonging to the application of the signature of that program to the model of that schema

Reasoning

Main Application

- ✓ It is possible to find automatically a sequence of basic translations to perform the transformation of a schema from a model to another, under suitable assumptions

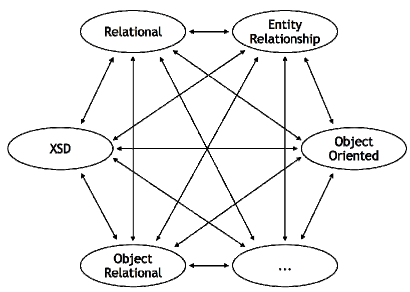
Observations

- ✓ Few “family” of models
  - ER, OO, Relational…
  - Each family has a progenitor
- ✓ Two kind of Datalog programs
  - Reduction
  - Transformation

Reasoning

Observations

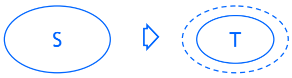
- ✓ Few “family” of models



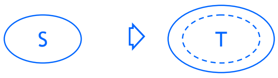
Reasoning

Observations

- ✓ Two kind of Datalog programs
  - Reduction



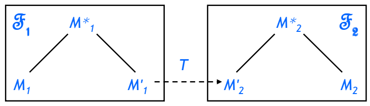
  - Transformation



Reasoning

Automatic Translation

- ✓ 3-steps transformation
  - Reduction within the source family
  - Transformation from the source family to the target family
  - Reduction within the target family



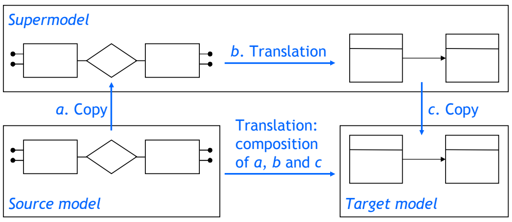
Supermodel & Rules

Enhanced Supermodel

- ✓ The usefulness of the MIDST proposal depends on the expressive power of its supermodel
  - The set of (families of) models handled
  - Accuracy and precision of the representation of such models
- ✓ Improvement of the expressive power of the supermodel has been performed by introducing new constructs
- ✓ New constructs imply definition of new rules

Transformation

In Theory



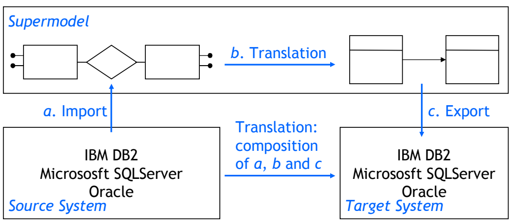
Exercise

Who is Missing?

- ✓ Instances
- ✓ The tool automatically generate instance level rules, from schema level rules
  - …but fails in some cases

Transformation

In Practice



MIDST tool

Guided Tour

MIDST
Model Independent Schema and Data Translation

MIDST tool

Where is the Problem?

- ✓ Off-line translation
  - The source DB is imported, translated, and exported
- ✓ This is not always feasible

MIDST tool

Where is the Problem?

- ✓ Off-line translation
  - The source DB is imported, translated, and exported
- ✓ This is not always feasible

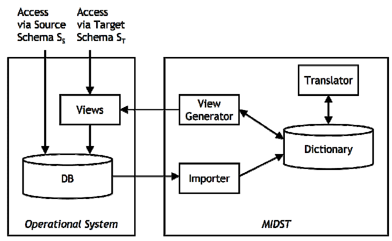
Idea

- ✓ Data not moved from the source system
- ✓ Translations performed directly on it

On-line Operator

2-steps Algorithm

- ✓ Generate views according to the constructs
- ✓ Specialize them according to the operational system



Motivating Examples

Data Migration

- ✓ Data migration is the process of transferring data between storage types, formats, or computer systems
- ✓ The need to migrate data can be driven by multiple business requirements
  - Storage migration
  - Database migration
  - Application migration
  - …

Motivating Examples

Data Migration

- ✓ Using MIDST
  - Define a virtual schema over the source system, that is also compatible with the target system
  - Define on the target system this virtual schema
  - Prepare the migration, updating the applications
  - Produce a set of statements that populate tables of the target system with data extracted from (the views over) the source schema

Motivating Examples

Data Migration via XML

- ✓ XML was thought for the Web, but is largely used for exchanging data
- ✓ Using MIDST, it is possible to migrate data via XML
  - Generate statements that create an XML representation of the original data
  - Import this XML document in the new system

Motivating Examples

Object/Relational Mapping

- ✓ How can “communicate” objects and relations?
- ✓ Two directions
  - From a relational schema to object-oriented wrappers
  - From a set of software objects to a relational schema definition
- ✓ Using MIDST
  - Different object-relational technologies
  - Customizable mapping

Motivating Examples

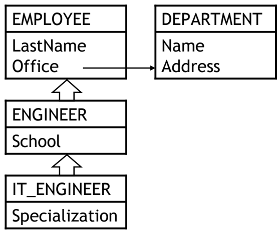
Updating Relational Views

- ✓ Views can be used to expose different schemas over the same data to different users/applications
- ✓ In general they are not updatable
- ✓ Using MIDST
  - Define views
  - Introducing “reverse mappings”, MIDST knows the origin of data and can forward updates to original data

Running Example

Source Schema

- ✓ An object relational schema



Running Example

Target Model

- ✓ Relational model
  - DEPARTMENT (DEPARTMENT_OID, Name, Address)
  - EMPLOYEE (EMP_OID, LastName, DEPARTMENT_OID_fk)
  - ENGINEER (ENG_OID, School, EMP_OID_fk)
  - IT_ENGINEER (IT_ENG_OID, Specialization, ENG_OID_fk)

Running Example

Translation

- ✓ A schema-level translation should perform three tasks
  - Elimination of the multiple levels of generalizations
  - Replacement of references with foreign keys
  - Transformation of the typed tables into value-based ones

Running Example

Runtime Translation in MIDST

- At each step, the tool produces a set of views
  - CREATE VIEW ENG_A … AS
    (SELECT … SCHOOL, … EMP_OID
    FROM ENG
    );

Generating Views

General Approach

- Analysis of Datalog schema rules
- Production of system generic statements
- Coding of statements in SQL-like language
- Specialization of encoded statements

Generating Views

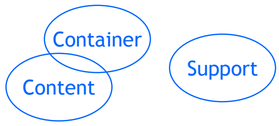
View

- CREATE VIEW … AS
  SELECT …
  FROM …
- How to fill blank spaces?

Generating Views

Classification of Constructs

- Container constructs
  - Sets or structured objects in the operational system
- Content constructs
  - Elements of more complex constructs
- Support constructs
  - Elements that model relationships and constraints between constructs, without storing data



Generating Views

Classification of Rules

- Container generating rules
- Content generating rules
- Support generating rules

Generating Views

Idea

- Define a view for each container
- Fields of a view derive from contents related with the corresponding container
- Support constructs do not affect view definitions

Generating Views

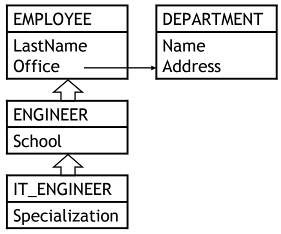
Example

- ✓ H ← B
  - If H is a container, create a view for each instantiation of B
  - If H is a content, add a field to a certain view

Generating Views

Running Example

- ✓ Elimination of generalizations
  - Container generating rule:
    copy abstracts
  - Content generating rules:
    copy lexicals;
    copy abstract attribute;
    replace generalizations
    with references



Generating Views

Problem

- ✓ Provenance of data
  - Where to derive values from?
  - How to generate new values?

Generating Views

Solution

- ✓ Provenance of data is encoded in Skolem functors
  - H (OID: SK(…), …) ← …
  - If SK has an oid of a content as a parameter, then the value for H comes from the instance of construct having that oid
    - Lexical (
      OID: SK2(oid),
      …)
      ←
      Lexical (
      OID: oid,
      …),
      Abstract (…);

Generating Views

Solution

- ✓ Provenance of data is encoded in Skolem functors
  - H (OID: SK(…), …) ← …
  - If SK has no parameter referring to a content, then we add an annotation to the corresponding rule
    - AbstractAttribute (
      OID: SK2 (genOID),
      …)
      ←
      Generalization (
      OID: genOID,
      …),
      … ;
    - In this case we can use the internal tuple OID

Generating Views

Running Example

- CREATE VIEW ENG_A AS
  (SELECT SCHOOL, REF(ENG_OID) AS EMP_OID
  FROM ENG
  );

View Generation Algorithm

Input

- ✓ A schema level translation
- ✓ A classification of the constructs

Output

- ✓ SQL statements defining views

View Generation Algorithm

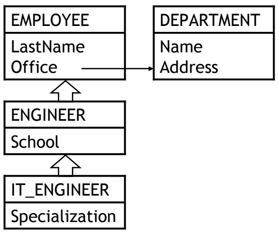
Definitions

- ✓ Containers(T)
  - The set of containers-generating rules of a translation T
- ✓ Contents(T)
  - The set of content-generating rules of a translation T
- ✓ Content(R,T)
  - The set of rules (of a translation T) generating contents for a container generated by R (R∈Containers(T))
- ✓ Abstract view
  - A pair (R, content(R,T), with R∈Containers(T)

View Generation Algorithm

Running Example

- ✓ Elimination of generalizations (T)
  - Containers(T):
    copy abstracts
  - Contents(T):
    copy lexicals;
    copy abstract attribute;
    replace generalizations
    with references



View Generation Algorithm

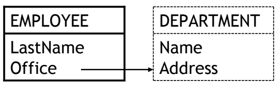
And Now?

- ✓ Instantiate abstract view
- ✓ Translate each instantiated abstract view into a view generating statement
- ✓ Specialize view generating statements according to a specific system's syntax

View Generation Algorithm

Running Example

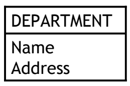
- ✓ Elimination of generalizations
  - V_1 = (EMP $\rightarrow_{\text{copy-abstract}}$ EMP ,
    { EMP(lastname) $\rightarrow_{\text{copy-lexical}}$ EMP(lastname),
    EMP(office) $\rightarrow_{\text{copy-AbstractAttribute}}$ EMP(office) })



View Generation Algorithm

Running Example

- ✓ Elimination of generalizations
  - V_2 = (DEPT $\rightarrow_{\text{copy-abstract}}$ DEPT,
    { DEPT (name) $\rightarrow_{\text{copy-lexical}}$ DEPT (name),
    DEPT (address) $\rightarrow_{\text{copy-lexical}}$ DEPT (address) })

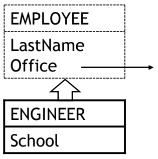


View Generation Algorithm

Running Example

✓ Elimination of generalizations

- V_3 = (ENG $\rightarrow_{copy\text{-}abstract}$ ENG ,
  { ENG(school) $\rightarrow_{copy\text{-}lexical}$ ENG(school),
  Gen(EMP;ENG) $\rightarrow_{elim\text{-}gen}$ ENG(EMP)})

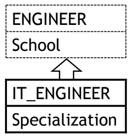


View Generation Algorithm

Running Example

✓ Elimination of generalizations

- V_4 = (IT_ENG $\rightarrow_{copy\text{-}abstract}$ IT_ENG,
  { IT_ENG (specialization) $\rightarrow_{copy\text{-}lexical}$ IT_ENG (specialization),
  Gen(ENG; IT_ENG) $\rightarrow_{elim\text{-}gen}$ IT_ENG(ENG)})



View Generation Algorithm

Running Example

✓ Elimination of generalizations

- CREATE TYPE OR_DEMO_1.DEPARTMENT_t AS (
  NAME varchar(50),
  ADDRESS varchar(50))
  NOT FINAL MODE DB2SQL WITH FUNCTION ACCESS REF USING INTEGER;
- CREATE VIEW OR_DEMO_1.DEPARTMENT of OR_DEMO_1.DEPARTMENT_t MODE DB2SQL (REF is OIDDEPARTMENT USER GENERATED) AS
  SELECT CAST(CAST(OR_DEMO.DEPARTMENT.OIDDEPT AS INTEGER) AS REF(OR_DEMO_1.DEPARTMENT_t)),
  OR_DEMO.DEPARTMENT.NAME,
  OR_DEMO.DEPARTMENT.ADDRESS
  FROM OR_DEMO.DEPARTMENT;

View Generation Algorithm

Running Example

✓ Elimination of generalizations

```
CREATE TYPE OR_DEMO_1.DEPARTMENT_t AS(
   NAME varchar(50),
   ADDRESS varchar(50))
NOT FINAL MODE DB2SQL WITH FUNCTION ACCESS REF USING INTEGER;

CREATE VIEW OR_DEMO_1.DEPARTMENT of OR_DEMO_1.DEPARTMENT_t MODE DB2SQL
   (REF is OIDDEPARTMENT USER GENERATED) AS
   SELECT CAST(CAST(OR_DEMO.DEPARTMENT.OIDDEPT AS INTEGER) AS REF(OR_DEMO_1.DEPARTMENT_t)),
      OR_DEMO.DEPARTMENT.NAME,
      OR_DEMO.DEPARTMENT.ADDRESS
   FROM OR_DEMO.DEPARTMENT;

CREATE TYPE OR_DEMO_1.EMPLOYEE_t AS(
   LASTNAME varchar(50),
   OFFICE REF(OR_DEMO_1.DEPARTMENT_t))
NOT FINAL MODE DB2SQL WITH FUNCTION ACCESS REF USING INTEGER;

CREATE VIEW OR_DEMO_1.EMPLOYEE of OR_DEMO_1.EMPLOYEE_t MODE DB2SQL
   (REF is OIDEMPLOYEE USER GENERATED, DEPT WITH OPTIONS SCOPE OR_DEMO_1.DEPARTMENT) AS
   SELECT CAST(CAST(OR_DEMO.EMPLOYEE.OIDEMP AS INTEGER) AS REF(OR_DEMO_1.EMPLOYEE_t)),
      OR_DEMO.EMPLOYEE.LASTNAME,
      CAST(CAST(OR_DEMO.EMPLOYEE.OFFICE AS INTEGER) AS REF(OR_DEMO_1.DEPARTMENT_t))
   FROM OR_DEMO.EMPLOYEE;

CREATE TYPE OR_DEMO_1.ENGINEER_t AS(
   SCHOOL varchar(50),
   to_EMPLOYEE REF(OR_DEMO_1.EMPLOYEE_t))
NOT FINAL MODE DB2SQL WITH FUNCTION ACCESS REF USING INTEGER;

CREATE VIEW OR_DEMO_1.ENGINEER of OR_DEMO_1.ENGINEER_t MODE DB2SQL
   (REF is OIDENGINEER USER GENERATED, to_EMPLOYEE WITH OPTIONS SCOPE OR_DEMO_1.EMPLOYEE) AS
   SELECT CAST(CAST(OR_DEMO.ENGINEER.OIDEMP AS INTEGER) AS REF(OR_DEMO_1.ENGINEER_t)),
      OR_DEMO.ENGINEER.SCHOOL,
      OR_DEMO_1.EMPLOYEE_t(CAST(OR_DEMO.ENGINEER.OIDEMP AS INTEGER))
   FROM OR_DEMO.ENGINEER;

CREATE TYPE OR_DEMO_1.IT_ENGINEER_t AS(
   SPECIALIZATION varchar(50),
   to_ENGINEER REF(OR_DEMO_1.ENGINEER_t))
NOT FINAL MODE DB2SQL WITH FUNCTION ACCESS REF USING INTEGER;

CREATE VIEW OR_DEMO_1.IT_ENGINEER of OR_DEMO_1.IT_ENGINEER_t MODE DB2SQL
   (REF is OIDIT_ENGINEER USER GENERATED, to_ENGINEER WITH OPTIONS SCOPE OR_DEMO_1.ENGINEER) AS
   SELECT CAST(CAST(OR_DEMO.IT_ENGINEER.OIDEMP AS INTEGER) AS REF(OR_DEMO_1.IT_ENGINEER_t)),
      OR_DEMO.ENGINEER.SPECIALIZATION,
      OR_DEMO_1.ENGINEER_t(CAST(OR_DEMO.IT_ENGINEER.OIDEMP AS INTEGER))
   FROM OR_DEMO.IT_ENGINEER;
```