A Vision for Management of Complex Models

Presentation by Ali
Discussion by Monir

March 9, 2009
Basic Terminology

- **Models**
  - SQL schema
  - OO interface
  - UML model
  - XML DTD

- **Mappings**
  - DTD-to-DTD
  - SQL schema-to-SQL schema
  - ER-to-SQL Schema
Applications of Mapping

- **DB design** by mapping ER model to SQL schema
- **Web site design** via models that map DB to page layout
- **Program design** by generating templates or code from a UML model
- **Generate data warehouse loading programs** from mappings of data sources to DW schema
Motivation

- Need for:
  - Transformation of data from one model to another
  - Managing change in models

- Aim:
  - Reduce programming work
Discussion question

• Assume you just have a little background about database, now do you think, if having a Model Management System is feasible? Why or why not? For example think about finding an algorithm to find the best possible matching between two schemas?? Talk about the problems in designing.
Proposal

- Models and mappings are objects
- Define generic high-level operations on models and mappings
A model is a directed graph with one root.

A mapping is a model each of whose nodes connects nodes of two other models.

A Model for Model Management

Relational Schema

Emp
- E#
- Dept#
- Name
- Addr

map1
- =

XSD

Emp
- E#
- Dept#
- Name
- First
- Last
- Phone

Relational Schema

Emp
- E#
- Dept#
- Name
- Addr

A mapping is a model each of whose nodes connects nodes of two other models.
Challenge

• Developing a mechanism for representing models
Some Operations

- Match($M_1$, $M_2$, $\cong$, $map$)
- Compose($map_1$, $map_2$)
- Merge($M_1$, $M_2$, $map$)
- Enumerate($M$)
Mapping between models

- Types:
  - Best mapping
  - Best mapping consistent with prior knowledge
  - Extend the partial mapping

- Challenge:
  - Developing an algorithm for finding extended mappings
Match

- Match($M_1$, $M_2$, $\cong$) returns the best mapping between $M_1$ and $M_2$, w.r.t. to $\cong$.
Composition

- Notation – $\text{map3} = \text{map1} \cdot \text{map2}$
  - Map1: $M_1 \rightarrow M_2$
  - Map2: $M_2 \rightarrow M_3$
  - Map3: $M_1 \rightarrow M_3$

- Easy for single-valued functions
  - just use ordinary function composition
Merge($M_1$, $M_2$, map)

- Move content of $M_2$ into $M_1$ model
  - Use map to guide the Merge
- If it’s a union, just add children of $M_2$’s root under $M_1$’s root
Merge (cont’d)

- To avoid copying target object $m_2$ that’s already in $M_1$, connect $m_2$ to $m_1$ in map

- Challenge:
  - Proposing a semantic For Merge
Example

1. $map_2 = \text{Match(dtd1, dtd2)}$

2. $map_3 = map_1 \cdot map_2$

3. $<map_4, rdb2 > = \text{Copy}(map_3^{-1})$
Final Challenges

- Designing an algebra of useful operations
Discussion

- Thinking about the operators used in the MMS, which operators do you think can be performed easier and which do you think are more difficult. Please rank them and say your reasons.
  - 1) Enumerate  2) Match  3) Merge  4) Compose
Model Management 2.0: Manipulating Richer Mappings
Why is mapping hard?

- Heterogeneity
- Impedance mismatch
And it is getting harder

- More data models
  - XSD, RDF, OWL

- More programming languages
Two Types of Mappings

- **Engineered Mapping:**
  - precisely specified.
  - tested for applications.

- **Approximate Mapping:**
  - imprecision is tolerable
  - there is no well defined notion of correct answer.
Discussion

- This paper talks about two kinds of mapping: engineered mapping and approximate mapping, and after that it focus on engineered mapping. Now can you think about the applications of approximate mapping?? Which applications do you think that just need approximate mapping instead of engineered mapping??
Model Management 1.0

- Research focus on more powerful operations
- Hence better tools

- Good News
  - Lots of progress on operations
  - Some practical applications

- Bad News
  - Still waiting for the first reasonably-complete practical implementation
Version 1.0: Mapping is a structure
Version 2.0: Just use the expression

\[
\pi_{\text{EID}}(\text{Empl}) = \pi_{\text{SID}}(\text{Staff}) \land \\
\pi_{\text{EID,Name}}(\text{Empl}) = \pi_{\text{SID,Name}}(\text{Staff}) \land \\
\pi_{\text{EID,AID,Concat(Street,City,Zip)}}(\text{Empl} \bowtie \text{Addr}) = \\
\pi_{\text{SID,Address}}(\text{Staff})
\]
Scenarios

- **Create mappings** (*Given two schemas, generate a mapping*)
  - Correspondences (schema matching)
  - Mapping constraints (ConstraintGen)
  - Transformations (TransGen)

- **Evolve mappings**
Create Mappings

- **Three steps:**
  - **Element correspondences**
    - Exploit lexical analysis of element names, data types, previous matching, etc.
  - **Mapping constraints** relate instances of schemas
    - E.g., equality of relational expressions
  - **Transformation** is an executable mapping constraint
    - Constructs target instances from source instances
    - E.g., SQL query
Code generation Scenarios
Correspondences ➔ Constraints

- Directly interpret correspondences as mapping constraints
- If it’s a tree schema and keys correspond

Diagram:

- Empl
  - EID
  - Name
  - Tel
  - AID
  - Addr
  - AID
  - City
  - Zip

- Staff
  - SID
  - Name
  - BirthDate
  - City

1. \( \pi_{\text{EID}}(\text{Empl}) = \pi_{\text{SID}}(\text{Staff}) \)
2. \( \pi_{\text{EID,Name}}(\text{Empl}) = \pi_{\text{SID,Name}}(\text{Staff}) \)
3. \( \pi_{\text{EID,City}}(\text{Empl} \bowtie \text{Addr}) = \pi_{\text{SID,City}}(\text{Staff}) \)
Another example

Target: EER

Person
  Id
  Name

Employee
  Dept

Customer
  CreditScore
  BillingAddr

Source: SQL

HR
  Id
  Name

Empl
  Id
  Dept

Client
  Id
  Name
  Score
  ...
Mapping Constraints

SELECT p.Id, p.Name
FROM Persons AS p
WHERE p IS OF (ONLY Person)
   OR p IS OF (ONLY Employee)

SELECT Id, Name
FROM dbo.HR

SELECT e.Id, e.Dept
FROM Persons AS e
WHERE e IS OF Employee

SELECT Id, Dept
FROM dbo.Empl

SELECT c.Id, c.Name,
    c.CreditScore, c.BillingAddr
FROM Persons AS c
WHERE c IS OF Customer

SELECT Id, Name,
    Score, Addr
FROM dbo.Client
**Constraints** → **Transformations**

```sql
SELECT p.Id, p.Name
FROM Persons AS p
WHERE p IS OF (ONLY Person) OR p IS OF (ONLY Employee)

SELECT e.Id, e.Dept
FROM Persons AS e
WHERE e IS OF Employee

SELECT c.Id, c.Name, c.CreditScore, c.BillingAddr
FROM Persons AS c
WHERE c IS OF Customer

SELECT Id, Name
FROM dbo.HR

SELECT Id, Dept
FROM dbo.Empl

SELECT Id, Name, Score, Addr
FROM dbo.Client

SELECT CASE
    WHEN (T5._from2 AND NOT(T5._from1)) THEN Person(T5.Person_Id, T5.Person_Name)
    WHEN (T5._from1 AND T5._from2)
    THEN Employee(T5.Person_Id, T5.Person_Name, T5.Employee_Dept)
END
FROM (SELECT T1.Person_Id, T1.Person_Name, T2.Employee_Dept,
    CAST(NULL AS SqlServer.Int) AS Customer_CreditScore,
    CAST(NULL AS SqlServer.NVarChar) AS Customer_BillingAddr, False AS _from0,
    (T2._from1 AND T2._from1 IS NOT NULL) AS _from1, T1._from2
    FROM (SELECT T.Id AS Person_Id, T.Name AS Person_Name, True AS _from2
        FROM HR AS T)
LEFT OUTER JOIN (SELECT T.Id AS Person_Id, T.Dept AS Employee_Dept, True AS _from1
    FROM dbo.Empl AS T)
UNION ALL (SELECT T.Id AS Person_Id, T.Name AS Person_Name,
    CAST(NULL AS SqlServer.NVarChar) AS Employee_Dept,
    T.Score AS Customer_CreditScore, T.Addr AS Customer_BillingAddr,
    True AS _from0, False AS _from1, False AS _from2
    FROM Client AS T)

) AS T5
```
Schema Evolution

- Schema evolves
- What about database & view?

\[ \text{Map}_{SV} \]

\[ \text{view} \]
Create mapping: \( \text{schema} \leftrightarrow \text{evolved schema} \)

Generate a transformation
Compose $Map_{SV}$ and $Map_{ES}$ to connect view to evolved schema
Discussion

- This paper gives a revised vision of MMS compared to the original one. Now which one do you prefer, the first vision or the second one?? Do you think that we need such a complex system or do you still think the original MMS can be useful and applicable??
Thanks