Meta Data Management

Philip A. Bernstein
Sergey Melnik
{philbe, melnik}@microsoft.com

Microsoft Research

Modified version of the seminar presented at ICDE, Boston, April 1, 2004 – for presentation at CSEP 544

Chapter 1: Introduction

Meta Data Problems

- Many data management applications primarily involve transformations of structured data
  - Data translation
  - Schema evolution
  - XML message translation
  - Application integration
  - Data warehouse loading
  - ER/UML design tools
  - Wrapper generation for SQL
  - UI / 4GL generation
  - Dependency tracking
  - Lineage tracing
  - Info resource management
  - Binding, renaming
  - Software build (make)
  - Configuration management

Why Meta Data is Important

- Many DB problems are easier to solve by manipulating metadata instead of writing code
  - Instead of writing code
  - Instead of manipulating data directly
- Meta-data-based solutions all involve metadata models (schema and mappings)
  - Mappings - data transformations, queries, dependencies, ...
  - Model, manipulate, and generate them
  - Usually, generate code from them

Chapter 2: Meta Data Management

Outline

- Introduction
  - Meta data problems
  - Design patterns
  - Solution templates
  - Wrap up

Example: Object-Oriented Wrapper for SQL Tables

- Manually program a wrapper for each table
- This is very repetitive work
- So you write a program to generate a wrapper for each table
**OO wrapper for SQL (cont'd)**

**SQL model**

- PurchaseOrder
  - OrdID
  - OrderDate

**Class model**

- PurchaseOrder
  - OrdID
  - OrderDate

---

**Example - Data Translation**

1. Translate data from one data model to another
2. Either write a program or generate it

**Hierarchical Schema**

- PurchaseOrder
  - OrdID
  - OrderDate
  - Items
    - ProdID
    - ProdName

**Relational Schema**

- Items
  - ItemID
  - ItemName

---

**Outline**

1. **Introduction**
   - Meta data problems
   - Design patterns
   - Solution templates
   - Wrap up

---

**Meta Data Solution Template**

1. Get a data manager for models and mappings
   - Usually, it's an object manager
     - OOP programming language
     - OODB
   - Hence, meta-meta model is the object manager's built-in types
     - Classes, attributes, methods, objects
     - Plus operators to manipulate them, such as NewClass, NewAttribute, NewObject, WriteAttribute

2. Design meta models (e.g., for SQL schema)

---

**Meta Data Solution Template**

1. Get a data manager for models and mappings
   - If the meta-meta model is OOP, then the meta model consists of class definitions

---

© 2004 Microsoft Corporation. All rights reserved.
Meta Data Solution Template

1. Get a data manager for models and mappings
2. Design meta models (e.g., for SQL schemas)
3. Build a model importer for each meta model
4. Invoke model importer(s)

Example - Data Translation

Meta Models

Record

Field

Repeating Group

Table

Column
Meta Data Problems

1. Data translation
2. OO or XML wrapper generation for SQL DB
3. User interface / 4GL program generation
4. Design tool support (DB, UML, ...)
   - Model generation, reverse engineering
   - Round-trip engineering
5. Schema evolution (applies to all scenarios)
6. XML message translation for e-commerce
7. Integrate custom apps with commercial apps

Meta Data Problems (cont’d)

1. Data warehouse loading (clean & transform)
2. Lineage tracing (provenance)
3. Information resource management
4. Dependency tracking
5. Impact analysis
6. Navigation between tools
7. Binding, renaming
8. Software build & make
9. Version and configuration management
   - Release management
   - Product data management

Meta Data Solutions

1. They strongly resemble one another
2. We characterize that resemblance
   - Prototypical problems, design patterns
   - Solution specifications, solution templates
   - Primitive solution steps, operators
3. Goals
   - A methodology to solve meta data problems
   - Ultimately, operator implementations to turn solution templates into solution programs
Outline

1. Introduction
2. Meta data problems
3. Design patterns
4. Solution templates
5. Wrap up

Meta Data Design Patterns

1. Design pattern - a problem description consisting of
   Input models and mappings
   Output models and mappings
   Criteria for the output to be correct
   An application specializes to meta models and mapping languages

2. Solution template - a sequence of operators producing the desired output

3. Operators - a single step that computes a model and/or mappings

Operators

1. \( m_{ap} = \text{Match}(M_1, M_2) \)
   Return a mapping between the two models

2. \( \langle M_2, m_{ap} \rangle = \text{Model}(M_1, m_{etamodel}) \)
   Return a model that is expressed in meta model
   and is equivalent to model \( M_1 \)

3. \( \langle M_3, m_{ap}, m_{ap} \rangle = \text{Merge}(M_1, M_2, m_{ap}) \)
   Return the union of models \( M_1 \) and \( M_2 \)

4. \( m_{ap} = \text{Compose}(m_{ap}, m_{ap}) = m_{ap} \circ m_{ap} \)
   Return the composition of \( m_{ap} \) and \( m_{ap} \), which is a mapping from \( m_{ap} \)'s domain to \( m_{ap} \)'s range.

Operators (cont'd)

1. \( m_{ap} = \text{Confluence}(m_{ap}, m_{ap}) = m_{ap} \cdot m_{ap} \)
   Return the "merge" of mappings \( m_{ap} \) and \( m_{ap} \)

2. \( \langle M_2, m_{ap} \rangle = \text{Extract}(M_1, m_{ap}) \)
   Return the sub-model of \( M_1 \) that participates in the mapping \( m_{ap} \)

3. \( \langle M_2, m_{ap} \rangle = \text{Diff}(M_1, m_{ap}) \)
   Return the sub-model of \( M_1 \) that does not participate in the mapping \( m_{ap} \)

Meta Modeling

1. Design pattern - develop a representation (i.e. meta model) form models and mappings
2. Applications - they all depend on this
3. Solution template
   Design a meta model
   Write Import & Export functions
   \( \text{ImportSQL, ImportXSD, ImportERD, ...} \)
   Today, it is manual engineering design
   Design once and reuse often

Design Patterns

1. Meta Modeling
2. Model Mapping
3. Model Generation
4. Model Integration
5. Mapping Composition
6. Mapping Alignment
7. Change Propagation
8. Model Reintegration

Single Operator Solutions
Meta Modeling (cont’d)

1. The Import function for models
   
   Parse text
   
   Copy elements of the parsed form into a model that conforms to its metamodel

2. The Import function for mappings
   
   Same as models but may require some semantic analysis
   
   E.g., program dependencies, data lineage
   
   For some languages and mappings, Export is hard (e.g., XSLT)

Model Mapping

1. Design pattern - Design a mapping between two models and generate code from it

   \[ M_1 \xrightarrow{\text{map}} M_2 \]

   1. Applications
      
      - Data translation
      - XML message translation for e-commerce
      - Integrate custom and commercial apps
      - Data warehouse extract, transform & load

   1. Solution templates
      
      \( m_{\text{ap}} = \text{Match}(M_1, M_2); \text{Export}(m_{\text{ap}}) \)
      
      Mapping reuse: Compose, Confide

An XML Mapping Tool

A Data Warehouse Loading Tool

Model Generation

1. Design pattern - Given a model, generate an equivalent model in another metamodel

   \[ M_1 \xrightarrow{\text{map}} M_2 \]

   1. Applications
      
      - Mapper generation SQL(like or XML)
      - Design tools ER, SQL, SQL(like ER)
      - UI/4GL generation

   1. Solution template
      
      \( (M_1, m_{\text{ap}}) \rightarrow \text{ModelGen}(M_1, m_{\text{ap}}); \text{Export}(M_2) \)
      
      ModelGen often needs human guidance

Model Integration

1. Design pattern - Given two models, develop a model that subsumes both of them

   \[ M_1 \xleftarrow{\text{map}_{12}} M_2 \]

   1. Applications
      
      - View integration
      - Data integration

   1. Solution template
      
      \( (M_1, m_{\text{ap}_{12}}, m_{\text{ap}_{23}}) \rightarrow \text{Merge}(M_1, M_2, m_{\text{ap}_{12}}); \text{Export}(M_3, m_{\text{ap}_{23}}) \)
Mapping Composition
1. Design pattern - Compose two given mappings
   \[ M_1 \xrightarrow{\text{map}_{12}} M_2 \xrightarrow{\text{map}_{23}} M_3 \]

   Applications
   - Processing queries on views

   Solution templates
   - map_{13} = Compose(map_{12}, map_{23})
   - Answering queries using views: \( \text{LaV}, \text{LoLaV} \)

Mapping Alignment
1. Design pattern - Align two mappings between the same pair of models
   \[ M_1 \xleftrightarrow{\text{map}_1} M_2 \]

   Applications
   - P2P query processing, mapping design

   Solution template
   - map_{1} = Confidence(map_{1}, map_{2})

Model Reintegration
1. Design pattern - Given a model and mappings to two modified versions of the model, produce a merged model
   \[ M_1 \xrightarrow{\text{map}_{12}} M_2 \xrightarrow{\text{map}_{13}} M_4 \]

   Applications
   - Parallel development

   Solution template
   - Multistep application of any operators

Change Propagation
1. Design pattern - Given two models and a mapping. One model changes. Fix the mapping and other model.
   \[ M_1 \xrightarrow{\text{map}_{12}} M_2 \xrightarrow{\text{map}_{23}} M_3 \]

   Applications
   - Schema evolution, interface evolution, ...
   - Required maintenance for all meta data problems

   Solution template
   - Requires all of the operators

Outline
1. Introduction
2. Meta data problems
   1. Design patterns
      1. Solution templates
      - Change propagation
      - Model reintegration
      - Change propagation revisited
3. Research background
4. Wrap up
Change Propagation

1. Given
   - Map between xsd1 and SQL schema rdb1
   - xsd2, a modified version of xsd1

2. Produce
   - rdb2 to store instances of xsd2
     - Mapping between xsd2 and rdb2

3. Map1 = Match(xsd1, xsd2)

4. Map2 = Map1

Now we need to merge Diff(xsd2, Map4) into rdb3

5. Map5 = Map2

6. Map6 = ModelGen(xsd2', SQL)

7. Map7 = Map5

8. Map8 = Map6

9. Map9 = Map7

10. Map10 = Merge(rdb3, rdb4, Map8)
Complete Script in Rondo

Operator Definition: PropagateChanges(s1, d1, s1_d1, s2, c, s2_c)
1. s1_s2 = Match(s1, s2);  
2. (d1¢, d1¢_d1) = Delete(d1, Traverse(All(s1) - Domain(s1_s2), s1_d1));  
3. (c¢, c¢_c) = Extract(c, Traverse(All(s2) - Range(s1_s2), s2_c));  
4. c¢_d1¢ = c¢_c * Invert(s2_c) * Invert(s1_s2) * s1_d1 * Invert(d1¢_d1);  
5. (d2, c¢_d2, d1¢_d2) = Merge(c¢, d1¢, c¢_d1¢);  
6. s2_d2 = s2_c * Invert(c¢_c) * c¢_d2 + Invert(s1_s2) * s1_d1 * Invert(d1¢_d1) * d1¢_d2;  
7. return (d2, s2_d2);

Model reintegration

1 Design pattern
Reconcile independent changes
All changes of each model
No “duplicate additions”

1 Simplified example
“Additions” = add model element
(also: add constraints, reorganize model)
“Deletions” = delete model element
(also: drop constraints, reorganize model)
Mappings shown as lines between elements

Design pattern

<table>
<thead>
<tr>
<th>OrderID</th>
<th>Product</th>
<th>Brand</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_A</td>
<td>m_m_A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td>Rebate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Naive solution

<table>
<thead>
<tr>
<th>OrderID</th>
<th>Product</th>
<th>Rebate</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_A</td>
<td>m_m_A</td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td>Rebate</td>
<td></td>
</tr>
</tbody>
</table>

Solution template

<table>
<thead>
<tr>
<th>OrderID</th>
<th>Product</th>
<th>Brand</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_A</td>
<td>m_m_A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td>Rebate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Composition produces a partial mapping

<table>
<thead>
<tr>
<th>OrderID</th>
<th>Product</th>
<th>Rebate</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_A</td>
<td>m_m_A</td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td>Rebate</td>
<td>Discount</td>
</tr>
</tbody>
</table>
Solution script

1. \( (m_A, m_A_m_A) = \text{PropagateDeletions}(m, m_A, m_A, m_B, m_B, m_B) \);
2. \( (m_B, m_B_m_B) = \text{PropagateDeletions}(m, m_B, m_B, m_A, m_A, m_A) \);
3. \( (m_A, m_B) = \text{Diff}(m_A, \text{Invert}(m_A_m_A)) \);
4. \( (m_B, m_B) = \text{Diff}(m_B, \text{Invert}(m_B_m_B)) \);
5. \( m_A = m_A \cup m_A_m_A \cup \text{Match}(m_A, m_B) \);
6. \( m_B = m_B \cup m_B_m_B \cup \text{Match}(m_B, m_A) \);
7. \( m_R = m_A \cup m_A_m_A \cup \text{Match}(m_B, m_B_m_B) \);
8. \( m_m_R = m_B \cup m_B_m_B \cup \text{Match}(m_A, m_B) \);
9. return \( (m_R, m_m_R) \);
Sub-pattern: propagate deletions

Solution script

Outline

Change propagation

Change propagation revisited

First-cut taxonomy of patterns
Outline

1. Introduction
2. Meta data problems
3. Design patterns
4. Solution templates
5. Research background

Wrap up

The Commercial World

1. Books for IT professionals
   - A. Tanenbaum: Metadata Solutions, Addison-Wesley, 2001
   - D. Marco: Building and Managing the Meta Data Repository, Wiley, 2000
2. Standards
   - UML, MOF, CWM (OMG)
   - XML, RDF, XML Schema, OWL (W3C)
3. Products and tools
   - Modeling: IBM Rational Rose, Visio, CA AllFusion, Borland Together
   - General metadata managers: CA Advantage, Microsoft Meta Data Services, Meta Integration
   - Meta data services in data warehousing: ETL tools: Informatica, Ascential, ETL Data Advantage...

The Research World

1. Model Management
   - A computational meta data framework based on models, mappings, and the operators described here (Match, Merge, Compose, ...)
2. Meta Data is a very active research area
   - Papers coming from many DB research groups
   - Some are problem-focused (e.g., data integration)
   - Some are operator-focused (e.g., Match, Merge)

Summary

1. Many DB problems are easier to solve by manipulating meta data
2. Meta data problems and solutions strongly resemble one another
3. Methodology: Use design patterns, solution templates, and operators to simplify development of meta data applications
4. There is much research to be done

References

1. http://research.microsoft.com/db/ModelMgt
2. Overview
   - Bernstein, CIDR 2003
   - Bernstein, Halevy, Pottinger, SIGMOD Record, Dec. 2000
3. Imp for entation
   - Mekik, Rahm, & Bernstein, SEMOD 2003
   - J.W eb Semantics 1, 2003
4. Data Warehouse Examples
   - Bernstein & Rahm, ER 2000
5. Match Operation
6. Merge Operation
   - Pottinger & Bernstein, VLDB 2003
The Match “Operator”

1. Schema matching (mapping discovery)
   Given two schem as, return correspondences that specify pairs of related elements.

2. Semantic mapping (query discovery)
   Given correspondences between two schem as, return an expression that translates instances of one schem a into instances of the other.

Schema Matching Problem

1. Input
   - Schem as S₁ and S₂
   - Possibly data instances for S₁ and S₂
   - Background knowledge – thesauri, validated matches, standard schem as, constraints (keys, data types), ontologies, NL glossaries, etc.

2. Output
   - Correspondences between elements of S₁ and S₂

Schema Matching Approaches

1. Many good ideas
   - Rahm & Bernstein, VLDB J, Dec '01
   - But none are robust!

The Cupid Algorithm

1. Computes linguistic similarity of element pairs
2. Computes structural similarity of element pairs
3. Generates a mapping

Matching Anatom y Ontologies

1. Match two human anatomy ontologies
   - FMA – Univ. of Washinton
   - Galen CRM – Univ. of Manchester (UK)
   - By Peter Mor k (Univ. of Washinton)

2. Both models are big
3. Ultimate goal was finding differences
4. Like most match algorithms, ours calculates a similarity score for the m·n pairs of elements
Aligning Representations

FMA:

Cardiac Valve: generic part of Heart

CRM:

Heart: has Structural Component: Valve

Anatomy Matching Algorithm

1. Lexical Match
   - Normalize string, UMLS dictionary lookup, convert to concept ID from thesaurus

   - String comparison: 306 matches
   - Adding spaces, ignoring case: 1834 matches
   - Lexical tools: 3503 matches

2. Structure Match
   - Similarity (reified nodes) / Average (neighbors)
   - Back-propagate to neighbors

   S: similarity score

Anatomy Matching Example

S = 2/15

S = 1

S = 1

S = 2/3

S = 1

S = 5/6
Anatomy Matching Algorithm

1. Lexical Match
   - Normalize string, UMLS dictionary lookup, convert to concept-ID from thesaurus
2. Structure Match
   - Similarity(reified nodes) = Average(neighbors)
   - Back-propagate to neighbors
3. Align Superclasses
   - Super-class similarity = average similarity of children, grandchildren, great-grandchildren

Some Lessons

1. A common encoding of models is hard and involves compromises
2. Different styles of reifying relationships
3. CRM stores transitive relationships
4. Match needs to invent generalizations
   - In FMA, arterial supply, venous drainage, nerve supply, lymphatic drainage
   - In CRM, these all map to isServedBy
5. On big models, Match is expensive
   - Some steps required days to execute
   - Cross-product filled 80 GB (< 1GB input).

Outline

Introduction to Model Management

Using MM to solve meta data problems

1. Matching anatomy ontologies
2. Merging
3. Wrap-up

Merge(M_1, M_2, map)

1. Return the union of models M_1 and M_2
2. Use map to guide the Merge
3. Elements x = y in map, then collapse them into one element

Merge(M_1, M_2, map)

1. Buneman, Davidson, Kosky, EDBT 92
2. Meta model has aggregation & generalization only
3. Union, and collapse objects having the same name
4. Fix-up step for inconsistencies created by merging
5. Successive fixups lead to different results
6. Batch them at the end, to get a unique minimal result
7. Now enrich the meta model (containment, complex mappings, ... & merge sem antics (conflicts, deletes))
**Resolving Merge Conflicts**

<table>
<thead>
<tr>
<th>Meta Model Conflict</th>
<th>Model Conflict</th>
<th>Meta Model Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp map</td>
<td>Employee</td>
<td>Name Emp</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

**Contributions to Merge**

[Pottinger & Bernstein, VLDB'03]

1. Generic correctness criteria for Merge
2. Use of first-class inputs mapping (not just correspondences)
3. Taxonomy of conflicts & resolution strategies
4. Characterize when Merge can be automatic
5. An algorithm for an EER representation
6. Experiments evaluation

---

**An Approach to ModelGen**

[Atzeni & Torlone, EDBT'96]

1. Meta models are made of patterns
   - Object has sub-object
   - Aggregation has attributes
   - Aggregation has key
   (a) (b) (c)
2. Define pattern transformations as rules
   - For SQL, \[ A \rightarrow \begin{array} \end{array} \]
3. To translate Model into meta-model, apply rules that replace patterns in Model by patterns that are in Meta

**ModelGen Research**

1. More complete repertoire of patterns
2. Make patterns more generic
3. Integrate with rules engine (avoid cycles, control search)
4. Implement