Meta Data Management

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

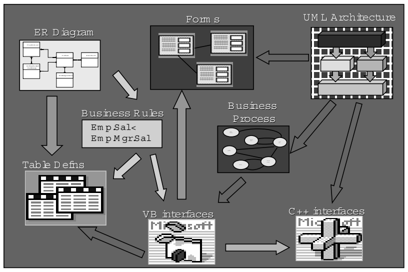
Microsoft Research

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

© 2004 Microsoft Corporation. All rights reserved.

Meta Data Management

- Meta data = structural information
  - DB schema, interface defn, web site map, form defns, …



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 mgmt
- Binding, renaming
- Software build (make)
- Configuration mgmt

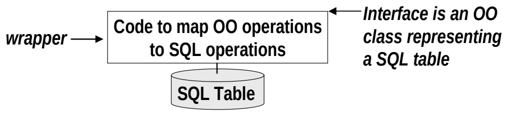
Outline

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

Why Meta Data is Important

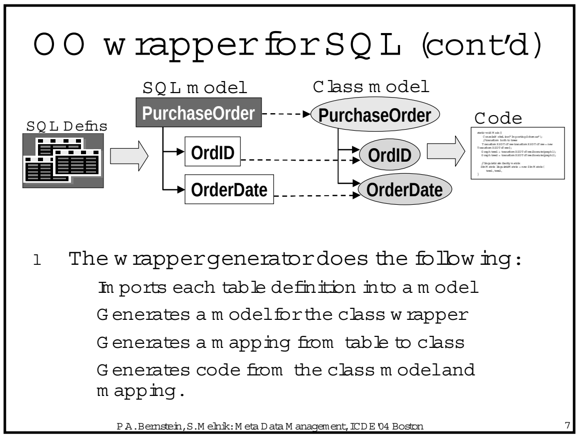
- Many DB problems are easier to solve by manipulating meta data
  - Instead of writing code
  - Instead of manipulating data directly
- Meta-data-based solutions all involve models (schemas) and mappings
  - Mappings - data transformations, queries, dependencies, …
  - Model, manipulate, and generate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

Example: Object-Oriented Wrapper for SQL Tables


wrapper
Code to map OO operations to SQL operations
Interface is an OO class representing a SQL table
SQL Table

- Manually program a wrapper for each table
- This is very repetitive work
- So you write a program to <u>generate</u> a wrapper for each table

© 2004 Microsoft Corporation. All rights reserved.



OO wrapper for SQL (cont'd)

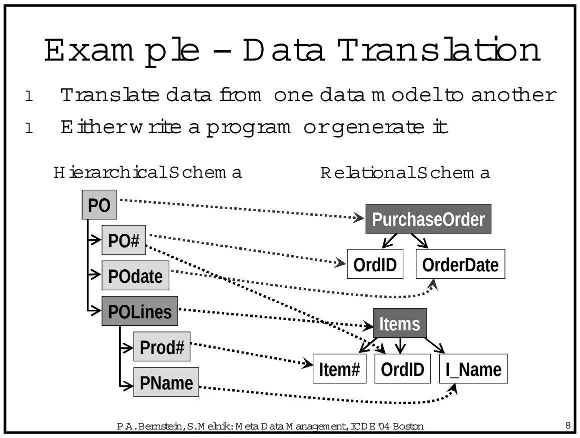
SQL Defns

SQL model: PurchaseOrder → OrdID, OrderDate

Class model: PurchaseOrder → OrdID, OrderDate

Code

- The wrapper generator does the following:
  - Imports each table definition into a model
  - Generates a model for the class wrapper
  - Generates a mapping from table to class
  - Generates code from the class model and mapping.

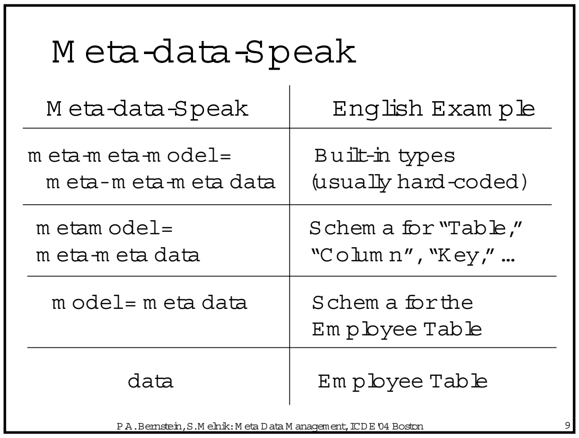


Example – Data Translation

- Translate data from one data model to another
- Either write a program or generate it

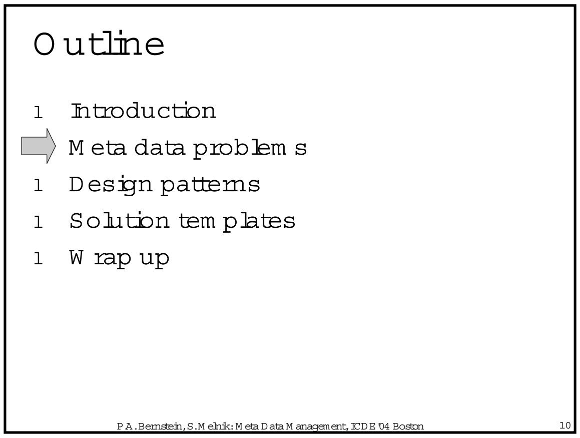
Hierarchical Schema: PO → PO#, POdate, POLines → Prod#, PName

Relational Schema: PurchaseOrder → OrdID, OrderDate; Items → Item#, OrdID, I_Name



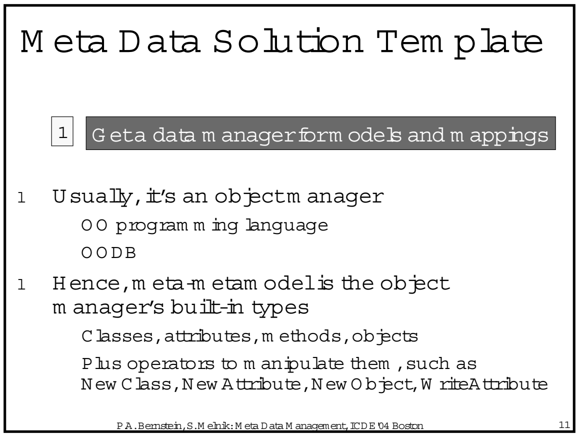
Meta-data-Speak

Meta-data-Speak	English Example
meta-meta-model = meta-meta-meta data	Built-in types (usually hard-coded)
metamodel = meta-meta data	Schema for "Table," "Column", "Key," …
model = meta data	Schema for the Employee Table
data	Employee Table



Outline

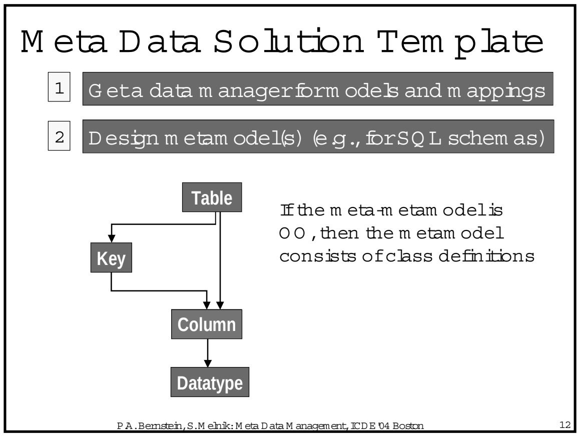
- Introduction
- ⇨ Meta data problems
- Design patterns
- Solution templates
- Wrap up



Meta Data Solution Template

1 Get a data manager for models and mappings

- Usually, it's an object manager
  - OO programming language
  - OODB
- Hence, meta-metamodel is the object manager's built-in types
  - Classes, attributes, methods, objects
  - Plus operators to manipulate them, such as NewClass, NewAttribute, NewObject, WriteAttribute



Meta Data Solution Template

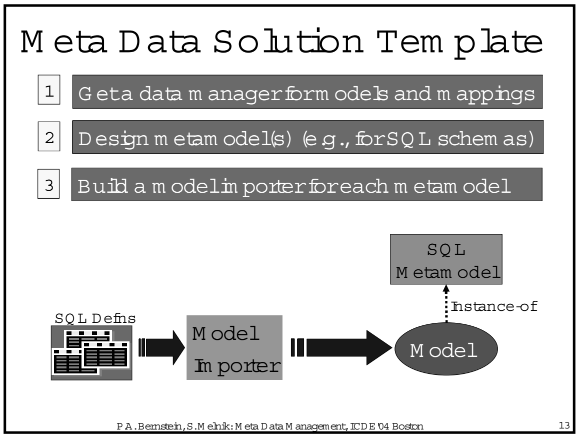
1 Get a data manager for models and mappings

2 Design metamodel(s) (e.g., for SQL schemas)

Table → Key, Column; Key → Column; Column → Datatype

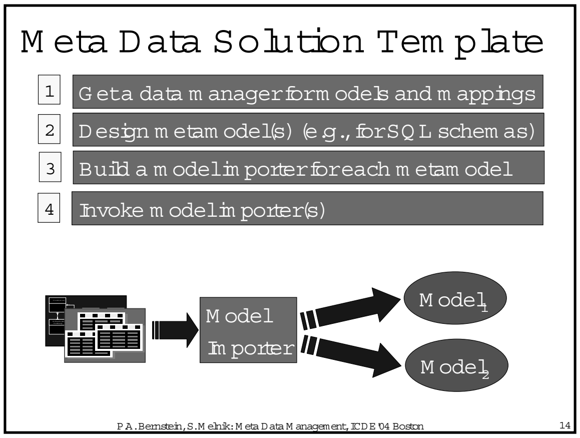
If the meta-metamodel is OO, then the metamodel consists of class definitions

© 2004 Microsoft Corporation. All rights reserved.



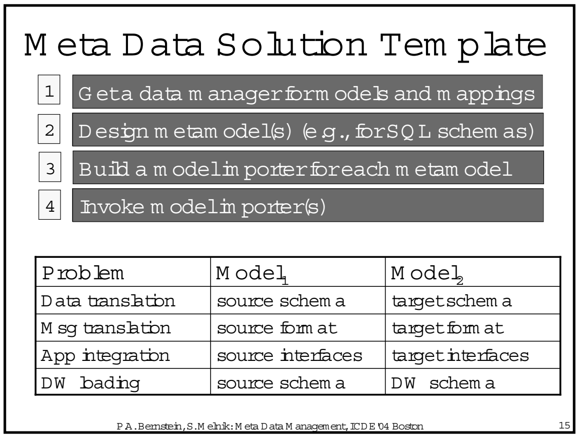
Meta Data Solution Template

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



Meta Data Solution Template

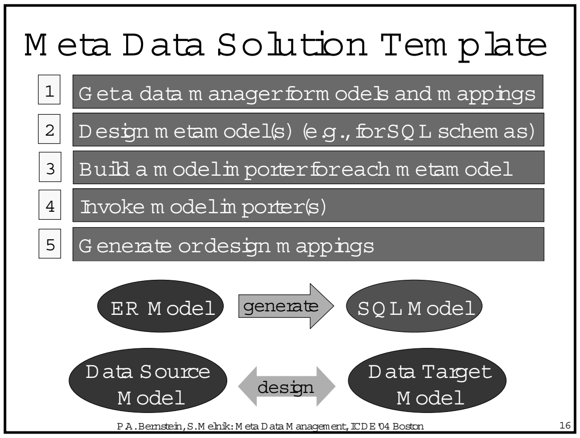
1. Get a data manager for models and mappings
2. Design metamodel(s) (e.g., for SQL schemas)
3. Build a model importer for each metamodel
4. Invoke model importer(s)



Meta Data Solution Template

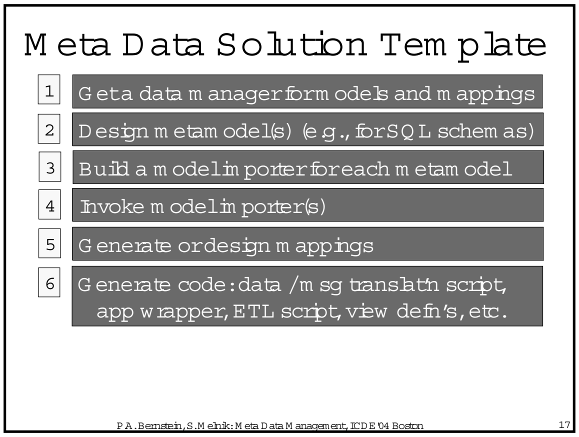
1. Get a data manager for models and mappings
2. Design metamodel(s) (e.g., for SQL schemas)
3. Build a model importer for each metamodel
4. Invoke model importer(s)

Problem	Model$_1$	Model$_2$
Data translation	source schema	target schema
Msg translation	source format	target format
App integration	source interfaces	target interfaces
DW loading	source schema	DW schema



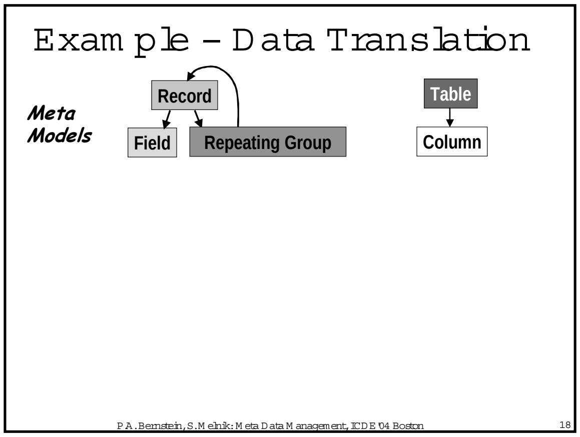
Meta Data Solution Template

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



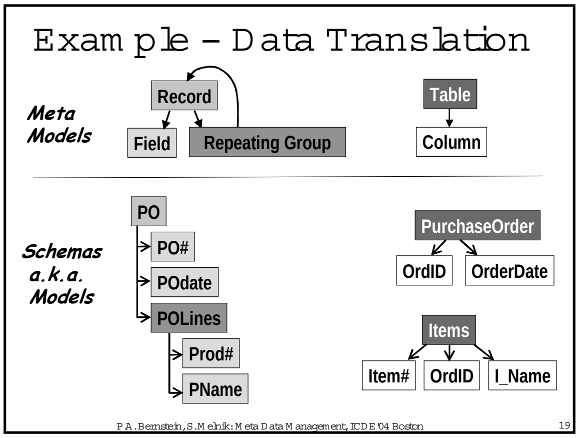
Meta Data Solution Template

1. Get a data manager for models and mappings
2. Design metamodel(s) (e.g., for SQL schemas)
3. Build a model importer for each metamodel
4. Invoke model importer(s)
5. Generate or design mappings
6. Generate code: data / msg translat'n script, app wrapper, ETL script, view defn's, etc.

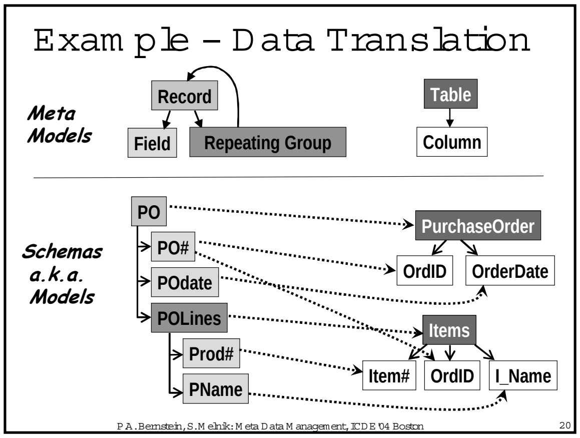


Example – Data Translation

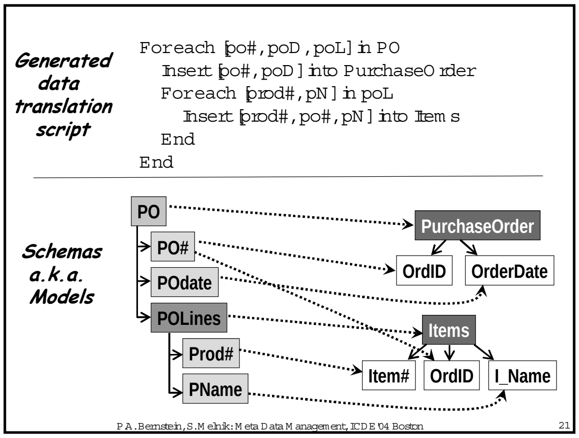
© 2004 Microsoft Corporation. All rights reserved.



Example – Data Translation

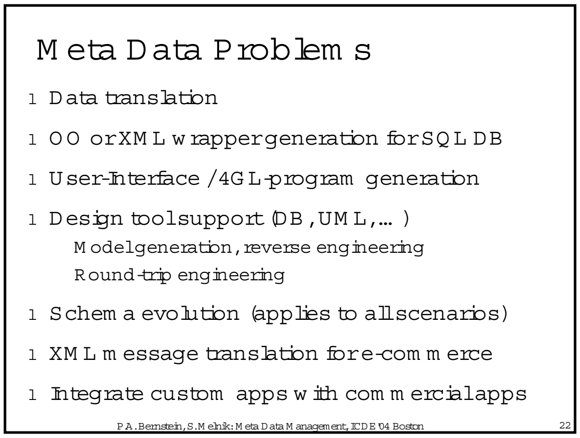


Example – Data Translation



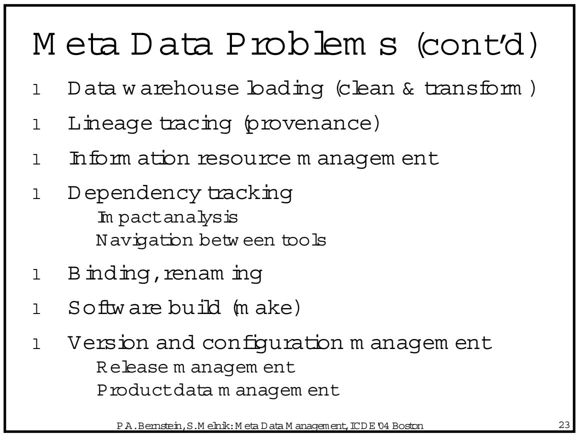
Generated data translation script

```
Foreach [po#,poD,poL] in PO
  Insert [po#,poD] into PurchaseOrder
  Foreach [prod#,pN] in poL
    Insert [prod#,po#,pN] into Items
  End
End
```



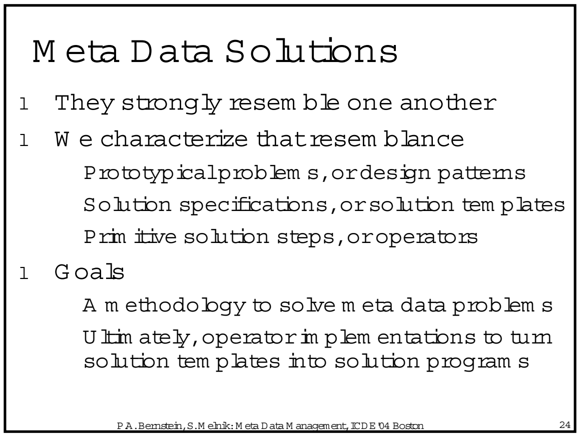
Meta Data Problems

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



Meta Data Problems (cont'd)

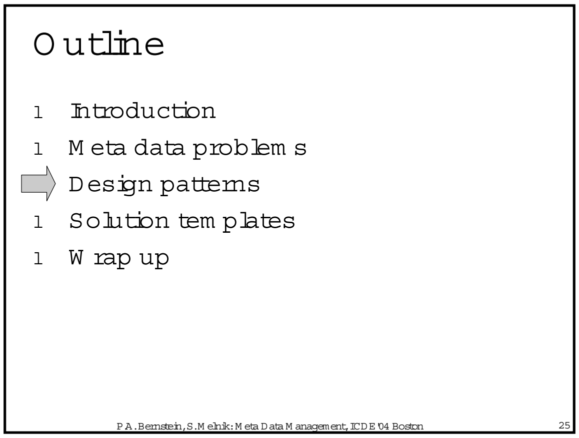
- Data warehouse loading (clean & transform)
- Lineage tracing (provenance)
- Information resource management
- Dependency tracking
  - Impact analysis
  - Navigation between tools
- Binding, renaming
- Software build (make)
- Version and configuration management
  - Release management
  - Product data management



Meta Data Solutions

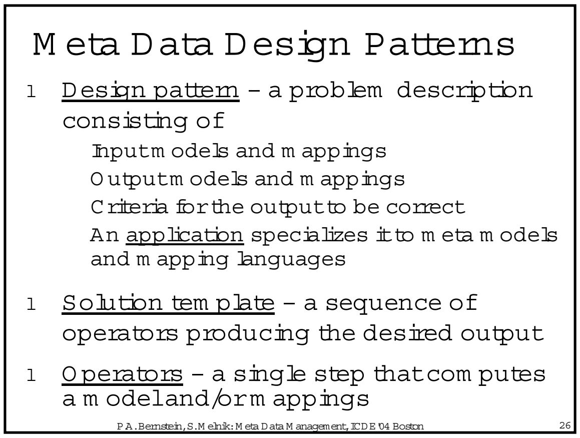
- They strongly resemble one another
- We characterize that resemblance
  - Prototypical problems, or design patterns
  - Solution specifications, or solution templates
  - Primitive solution steps, or operators
- Goals
  - A methodology to solve meta data problems
  - Ultimately, operator implementations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

© 2004 Microsoft Corporation. All rights reserved.



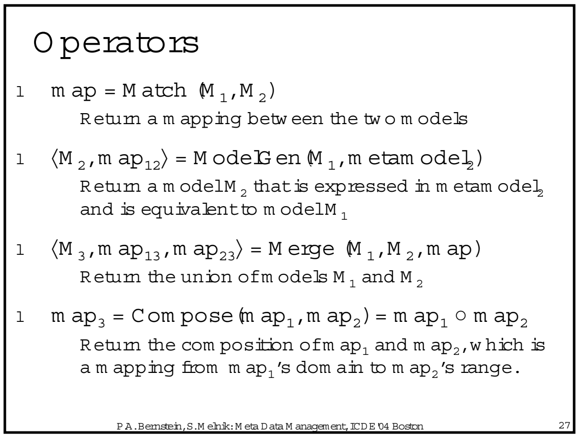
Outline

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



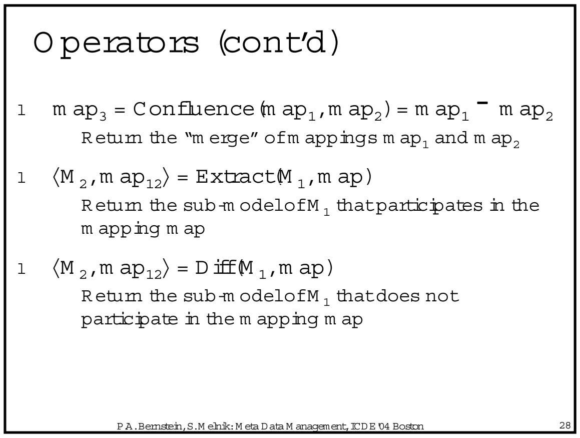
Meta Data Design Patterns

- 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 it to meta models and mapping languages
- Solution template – a sequence of operators producing the desired output
- Operators – a single step that computes a model and/or mappings



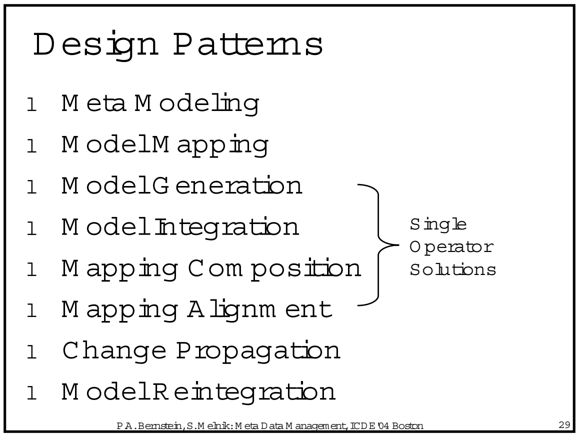
Operators

- $map = Match(M_1, M_2)$
  - Return a mapping between the two models
- $\langle M_2, map_{12} \rangle = ModelGen(M_1, metamodel_2)$
  - Return a model M_2 that is expressed in $metamodel_2$ and is equivalent to model M_1
- $\langle M_3, map_{13}, map_{23} \rangle = Merge(M_1, M_2, map)$
  - Return the union of models M_1 and M_2
- $map_3 = Compose(map_1, map_2) = map_1 \circ map_2$
  - Return the composition of map_1 and map_2, which is a mapping from map_1's domain to map_2's range.



Operators (cont'd)

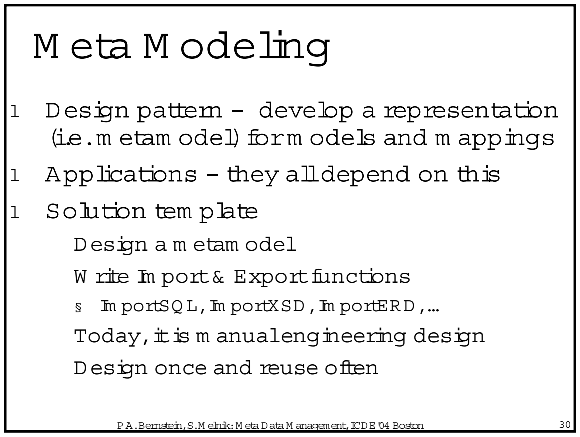
- $map_3 = Confluence(map_1, map_2) = map_1 \;^{-}\; map_2$
  - Return the "merge" of mappings map_1 and map_2
- $\langle M_2, map_{12} \rangle = Extract(M_1, map)$
  - Return the sub-model of M_1 that participates in the mapping map
- $\langle M_2, map_{12} \rangle = Diff(M_1, map)$
  - Return the sub-model of M_1 that does not participate in the mapping map



Design Patterns

- Meta Modeling
- Model Mapping
- Model Generation
- Model Integration
- Mapping Composition
- Mapping Alignment
- Change Propagation
- Model Reintegration

} Single Operator Solutions



Meta Modeling

- Design pattern – develop a representation (i.e. metamodel) for models and mappings
- Applications – they all depend on this
- Solution template
  - Design a metamodel
  - Write Import & Export functions
    - § ImportSQL, ImportXSD, ImportERD, …
  - Today, it is manual engineering design
  - Design once and reuse often

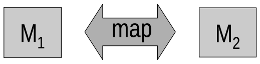
© 2004 Microsoft Corporation. All rights reserved.

Meta Modeling (cont'd)

- The Import function for models
  - Parse text
  - Copy elements of the parsed form into a model that conforms to its metamodel
- The Import function for mappings
  - Same as models but may require more semantic analysis
  - E.g., program dependencies, data lineage
  - For some languages and mapping metamodels, Export is hard (e.g., XSLT)

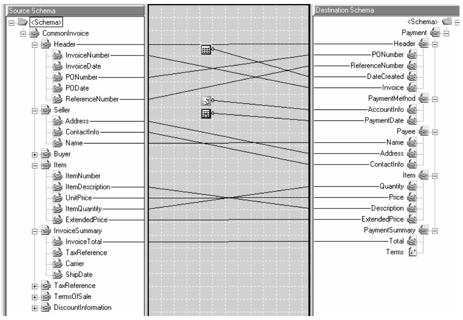
Model Mapping

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

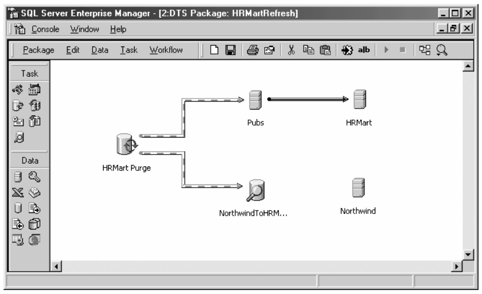


- Applications
  - Data translation
  - XML message translation for e-commerce
  - Integrate custom and commercial apps
  - Data warehouse extract, transform & load
- Solution templates
  - map = Match (M_1, M_2); Export(map)
  - Mapping reuse: Compose, Confluence

An XML Mapping Tool

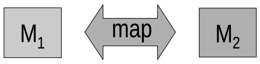


A Data Warehouse Loading Tool



Model Generation

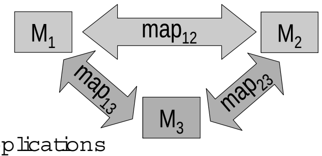
- Design pattern – Given a model, generate an equivalent model in another metamodel



- Applications
  - Wrapper generation (SQL → OO or XML)
  - Design tools (ER → SQL, SQL → ER)
  - UI/4GL generation
- Solution template
  - $\langle M_2$, map$\rangle$ = ModelGen (M_1, metamodel$_2$); Export(M_2)
  - ModelGen often needs human guidance

Model Integration

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

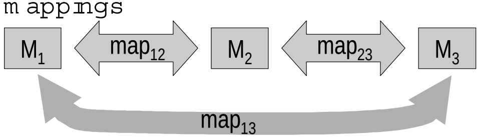


- Applications
  - View integration
  - Data integration
- Solution template
  - $\langle M_3$, map$_{13}$, map$_{23}\rangle$ = Merge (M_1, M_2, map); Export(M_3, map$_{13}$, map$_{23}$)

© 2004 Microsoft Corporation. All rights reserved.

Mapping Composition

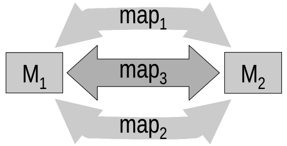
- Design pattern – Compose two given mappings



- Applications
  - Processing queries on views
- Solution templates
  - map_{13} = Compose(map_{12}, map_{23})
  - Answering queries using views (LaV),
  - Query modification (GaV), GLaV

Mapping Alignment

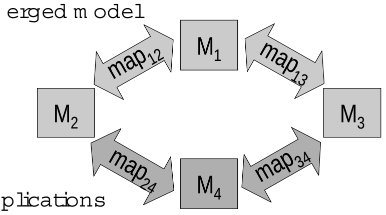
- Design pattern – Align two mappings between the same pair of models



- Applications
  - P2P query processing, mapping design
- Solution template
  - map_3 = Confluence(map_1, map_2)

Model Reintegration

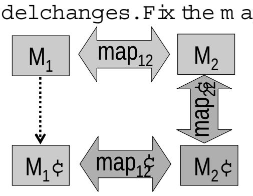
- Design pattern – Given a model and mappings to two modified versions of the model, produce a merged model



- Applications
  - Parallel development
- Solution template
  - Multistep application of many operators

Change Propagation

- Design pattern – Given two models and a mapping. One model changes. Fix the mapping and other model.



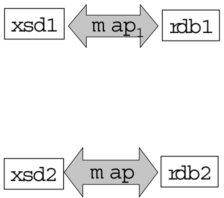
- Applications
  - Schema evolution, interface evolution, …
  - Required maintenance for all meta data problems
- Solution template
  - Requires all of the operators

Outline

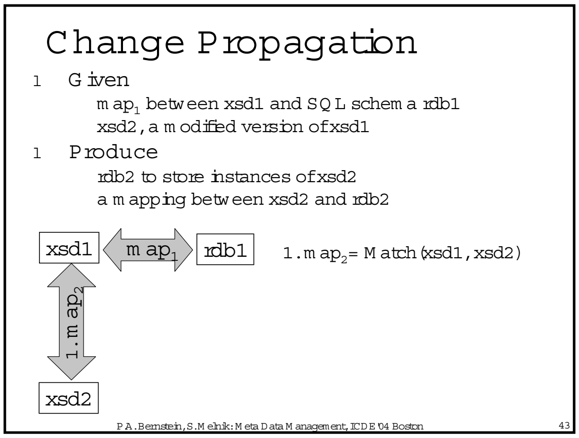
- Introduction
- Meta data problems
- Design patterns
- ➪ Solution templates
  - Change propagation
  - Model reintegration
  - Change propagation revisited
- Research background
- Wrap up

Change Propagation

- Given
  - map_1 between xsd1 and SQL schema rdb1
  - xsd2, a modified version of xsd1
- Produce
  - rdb2 to store instances of xsd2
  - a mapping between xsd2 and rdb2



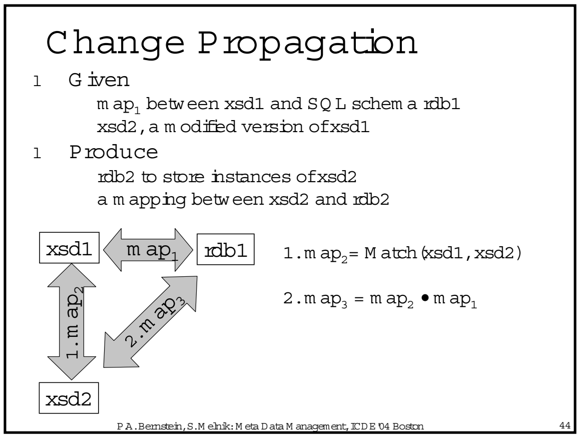
© 2004 Microsoft Corporation. All rights reserved.



Change Propagation

- Given
  - map_1 between xsd1 and SQL schema rdb1
  - xsd2, a modified version of xsd1
- Produce
  - rdb2 to store instances of xsd2
  - a mapping between xsd2 and rdb2

1. map_2 = Match(xsd1, xsd2)

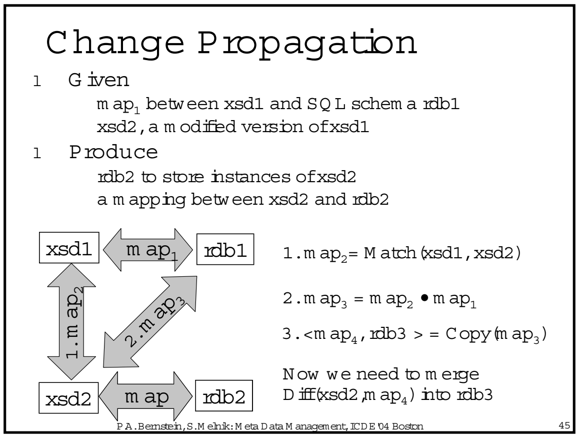


Change Propagation

- Given
  - map_1 between xsd1 and SQL schema rdb1
  - xsd2, a modified version of xsd1
- Produce
  - rdb2 to store instances of xsd2
  - a mapping between xsd2 and rdb2

1. map_2 = Match(xsd1, xsd2)

2. $map_3 = map_2 \bullet map_1$



Change Propagation

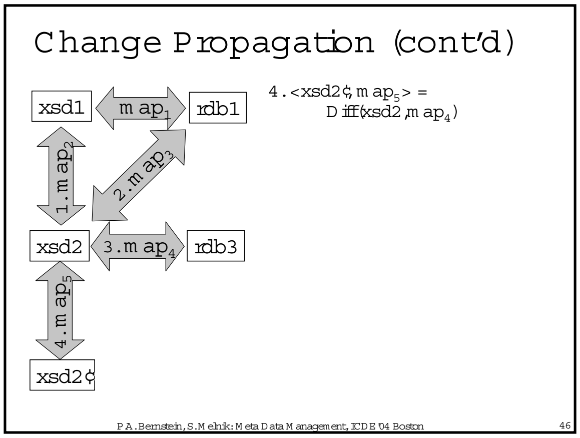
- Given
  - map_1 between xsd1 and SQL schema rdb1
  - xsd2, a modified version of xsd1
- Produce
  - rdb2 to store instances of xsd2
  - a mapping between xsd2 and rdb2

1. map_2 = Match(xsd1, xsd2)

2. $map_3 = map_2 \bullet map_1$

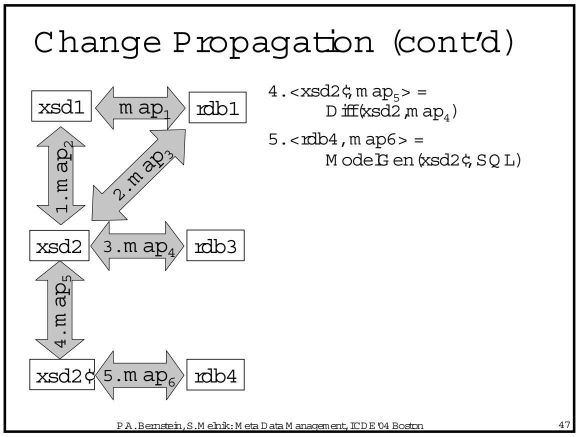
3. <map_4, rdb3 > = Copy(map_3)

Now we need to merge Diff(xsd2, map_4) into rdb3



Change Propagation (cont'd)

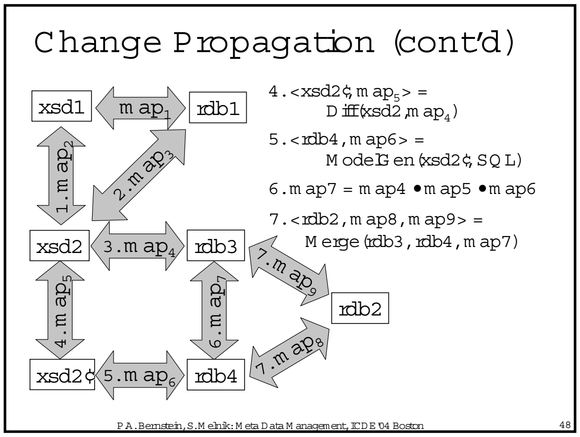
4. <xsd2¢, map_5> = Diff(xsd2, map_4)



Change Propagation (cont'd)

4. <xsd2¢, map_5> = Diff(xsd2, map_4)

5. <rdb4, map6> = ModelGen(xsd2¢, SQL)



Change Propagation (cont'd)

4. <xsd2¢, map_5> = Diff(xsd2, map_4)

5. <rdb4, map6> = ModelGen(xsd2¢, SQL)

6. map7 = map4 • map5 • map6

7. <rdb2, map8, map9> = Merge(rdb3, rdb4, map7)

© 2004 Microsoft Corporation. All rights reserved.

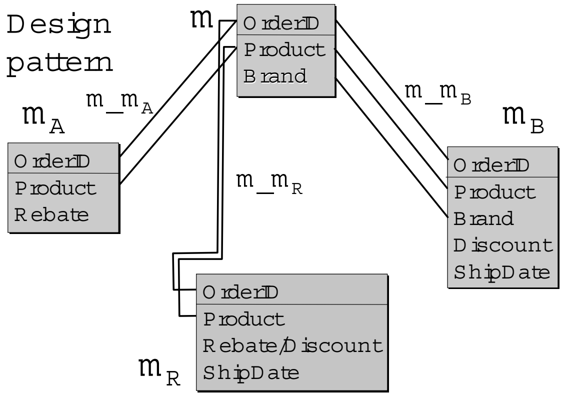
Complete Script in Rondo

```
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〉;
Operator Use:
SQL  XSD: PropagateChanges(s1, d1, s1_d1, s2, ModelGen(s2, XSD));
```

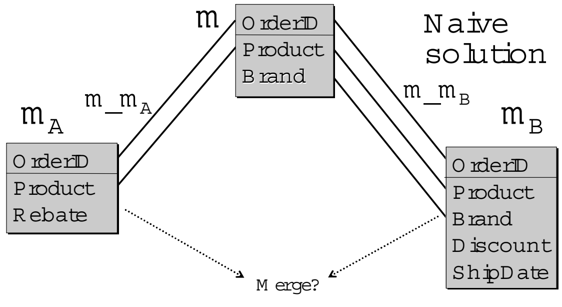
Model reintegration

- Design pattern
  - Reconcile independent changes
  - All changes of each model
  - No "duplicate additions"
- Simplified example
  - "Additions" = add model element
    (also: drop constraints, reorg. model)
  - "Deletions" = delete model element
    (also: add constraints, reorg. model)
  - Mappings shown as lines betw. elements

Design pattern

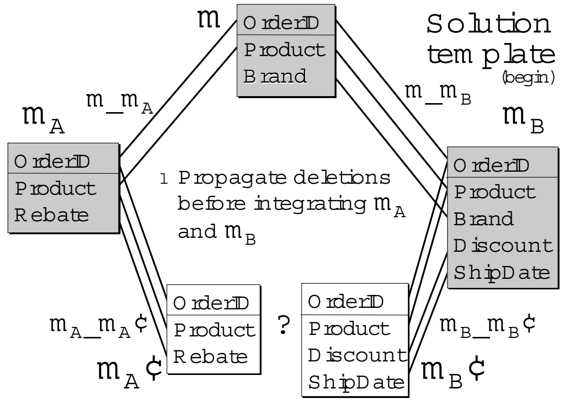


Naive solution

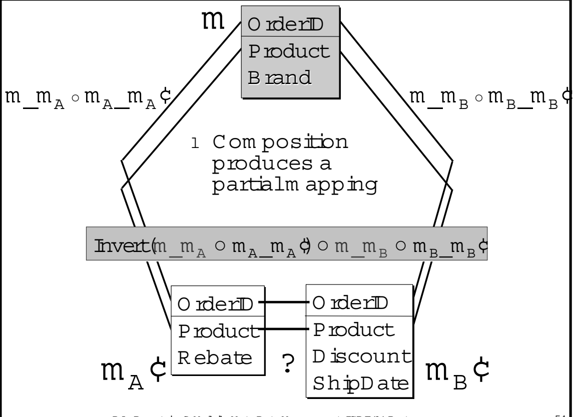


- Direct model integration "loses" either deletions or additions
- Need m, m_m_A, m_m_B

Solution template (begin)

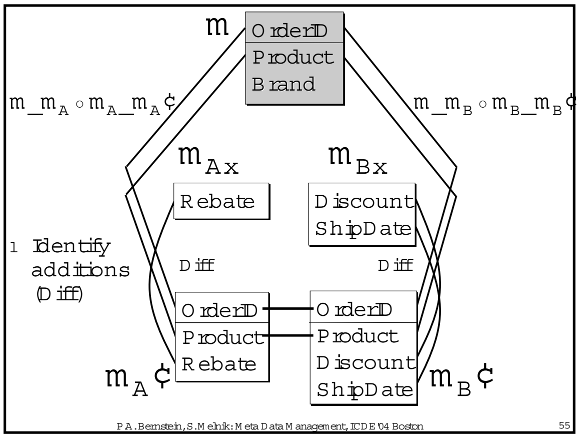


- Propagate deletions before integrating m_A and m_B



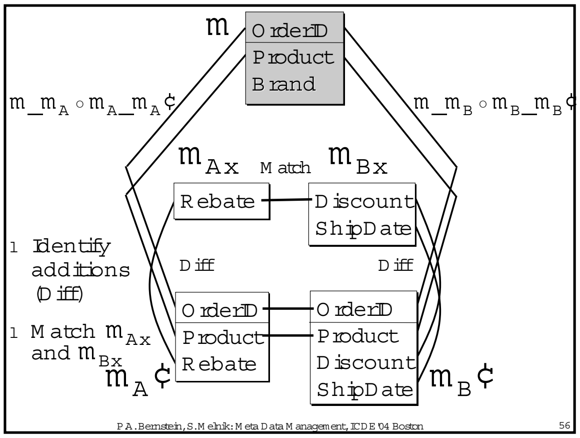
- Composition produces a partial mapping

© 2004 Microsoft Corporation. All rights reserved.



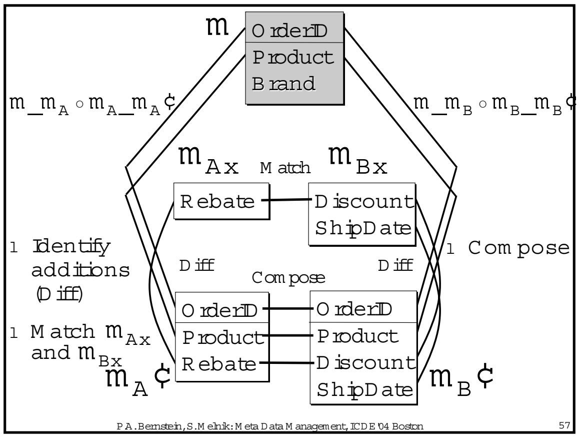
- Identify additions (Diff)

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE'04 Boston 55



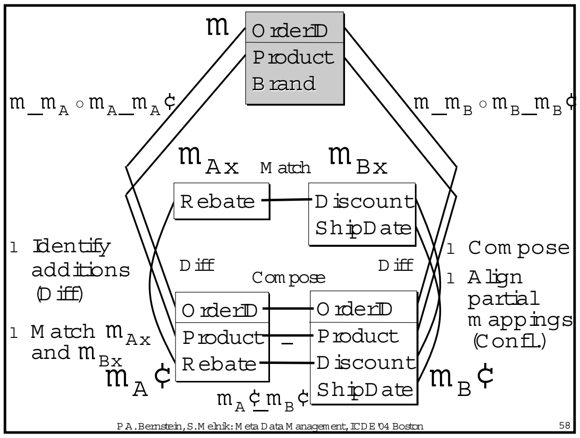
- Identify additions (Diff)
- Match m_{Ax} and m_{Bx}

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE'04 Boston 56



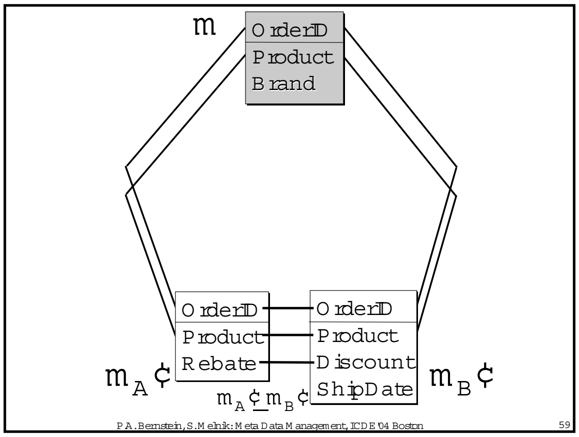
- Identify additions (Diff)
- Match m_{Ax} and m_{Bx}
- Compose

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE'04 Boston 57

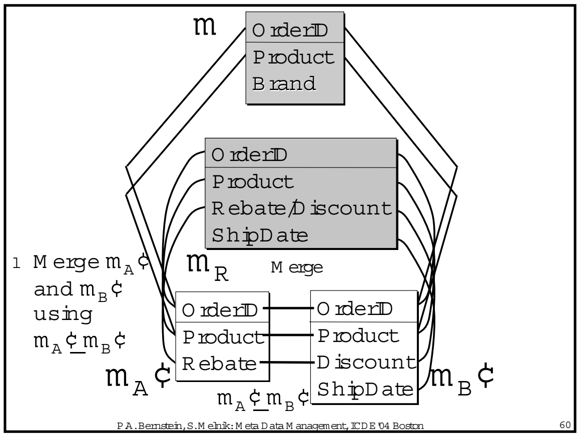


- Identify additions (Diff)
- Match m_{Ax} and m_{Bx}
- Compose
- Align partial mappings (Confl.)

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE'04 Boston 58



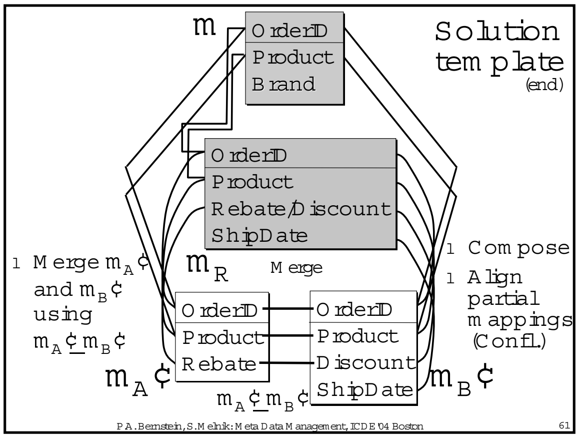
P.A. Bernstein, S. Melnik: Meta Data Management, ICDE'04 Boston 59



- Merge m_A¢ and m_B¢ using m_A¢$\_m_B$¢

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE'04 Boston 60

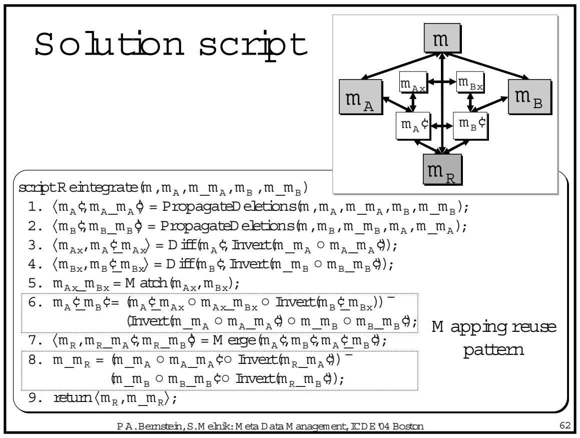
© 2004 Microsoft Corporation. All rights reserved.



Solution template

(end)

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE '04 Boston 61

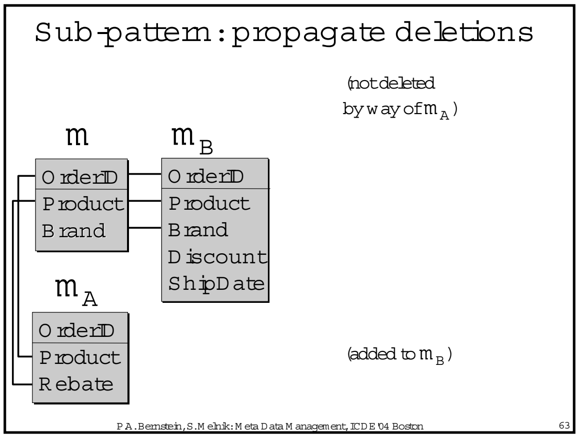


Solution script

```
script Reintegrate($m, m_A, m\_m_A, m_B, m\_m_B$)
 1. $\langle m_A$¢$, m_A\_m_A$¢$\rangle$ = PropagateDeletions($m, m_A, m\_m_A, m_B, m\_m_B$);
 2. $\langle m_B$¢$, m_B\_m_B$¢$\rangle$ = PropagateDeletions($m, m_B, m\_m_B, m_A, m\_m_A$);
 3. $\langle m_{Ax}, m_A$¢$\_m_{Ax}\rangle$ = Diff($m_A$¢, Invert($m\_m_A \circ m_A\_m_A$¢));
 4. $\langle m_{Bx}, m_B$¢$\_m_{Bx}\rangle$ = Diff($m_B$¢, Invert($m\_m_B \circ m_B\_m_B$¢));
 5. $m_{Ax}\_m_{Bx}$ = Match($m_{Ax}, m_{Bx}$);
 6. $m_A$¢$\_m_B$¢= ($m_A$¢$\_m_{Ax} \circ m_{Ax}\_m_{Bx} \circ$ Invert($m_B$¢$\_m_{Bx}$)) $^-$
                (Invert($m\_m_A \circ m_A\_m_A$¢) $\circ\ m\_m_B \circ m_B\_m_B$¢);
 7. $\langle m_R, m_R\_m_A$¢$, m_R\_m_B$¢$\rangle$ = Merge($m_A$¢$, m_B$¢$, m_A$¢$\_m_B$¢);
 8. $m\_m_R$ = ($m\_m_A \circ m_A\_m_A$¢$\circ$ Invert($m_R\_m_A$¢)) $^-$
             ($m\_m_B \circ m_B\_m_B$¢$\circ$ Invert($m_R\_m_B$¢));
 9. return $\langle m_R, m\_m_R\rangle$;
```

Mapping reuse pattern

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE '04 Boston 62

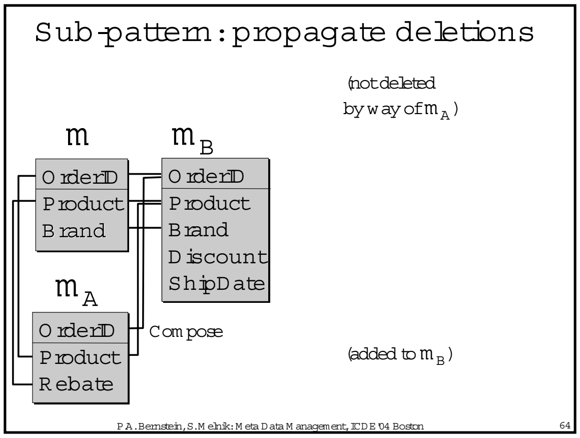


Sub-pattern: propagate deletions

(not deleted by way of m_A)

(added to m_B)

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE '04 Boston 63

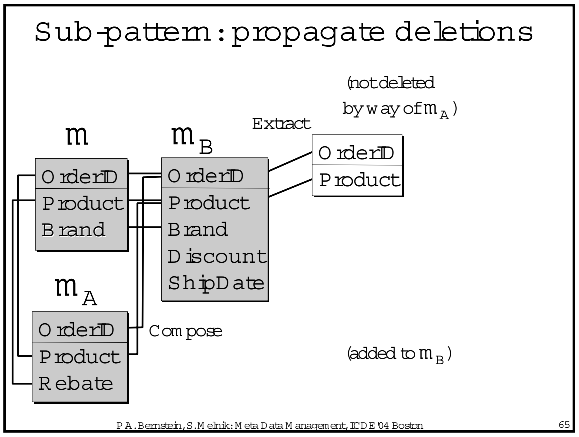


Sub-pattern: propagate deletions

(not deleted by way of m_A)

(added to m_B)

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE '04 Boston 64

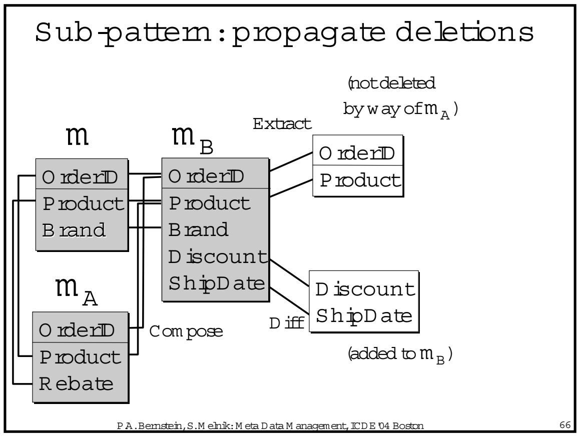


Sub-pattern: propagate deletions

(not deleted by way of m_A)

(added to m_B)

P.A. Bernstein, S. Melnik: Meta Data Management, ICDE '04 Boston 65



Sub-pattern: propagate deletions

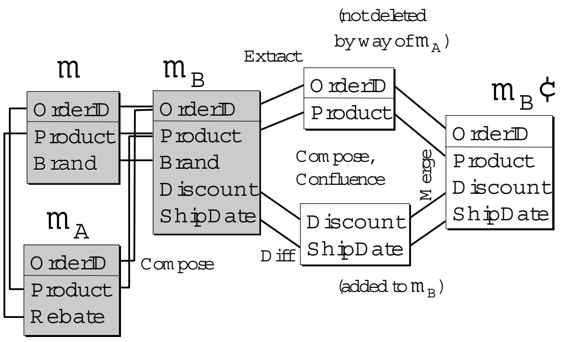
(not deleted by way of m_A)

(added to m_B)

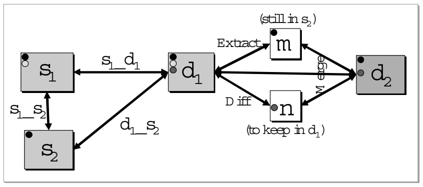
P.A. Bernstein, S. Melnik: Meta Data Management, ICDE '04 Boston 66

© 2004 Microsoft Corporation. All rights reserved.

Sub-pattern: propagate deletions



Solution script



```
script PropagateDeletions($s_1$, $d_1$, $s_1\_d_1$, $s_2$, $s_1\_s_2$)
1. $d_1\_s_2$ = Invert($s_1\_d_1$) $\circ$ $s_1\_s_2$;
2. $\langle m, d_1\_m \rangle$ = Extract($d_1$, $d_1\_s_2$);
3. $\langle n, d_1\_n \rangle$ = Diff($d_1$, Invert($s_1\_d_1$));
4. $\langle d_2, d_2\_m, d_2\_n \rangle$ = Merge(m, n, Invert($d_1\_m$) $\circ$ $d_1\_n$);
5. $d_1\_d_2$ = ($d_1\_m$ $\circ$ Invert($d_2\_m$)) ¯ ($d_1\_n$ $\circ$ Invert($d_2\_n$));
6. return $\langle d_2, d_1\_d_2 \rangle$;
```

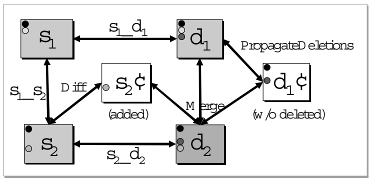
Outline

- Introduction
- Meta data problems
- Design patterns
- Solution templates
  - Change propagation
  - Model reintegration
  - Change propagation revisited
- Research background
- Wrap up

Change propagation

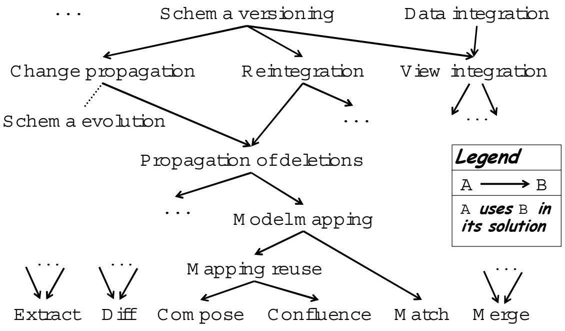
- Solution template
  - Propagate deletions
  - Include additions
  - Merge result

Change propagation



```
script PropagateChanges($s_1$, $d_1$, $s_1\_d_1$, $s_2$, $s_1\_s_2$)
1. $\langle d_1¢, d_1\_d_1¢ \rangle$ = PropagateDeletions($s_1$, $d_1$, $s_1\_d_1$, $s_2$, $s_1\_s_2$);
2. $\langle s_2¢, s_2\_s_2¢ \rangle$ = Diff($s_2$, Invert($s_1\_s_2$));
3. $\langle d_2, d_2\_s_2¢, d_2\_d_1¢ \rangle$ = Merge($s_2¢$, $d_1¢$, Invert($s_1\_s_2$ $\circ$ $s_2\_s_2¢$) $\circ$ $s_1\_d_1$ $\circ$ $d_1\_d_1¢$);
4. $d_1\_d_2$ = (Invert($s_1\_d_1$) $\circ$ $s_1\_s_2$ $\circ$ $s_2\_s_2¢$ $\circ$ Invert($d_2\_s_2¢$)) ¯ ($d_1\_d_1¢$ $\circ$ Invert($d_2\_d_1¢$));
5. $s_2\_d_2$ = (Invert($s_1\_s_2$) $\circ$ $s_1\_d_1$ $\circ$ $d_1\_d_1¢$ $\circ$ Invert($d_2\_d_1¢$)) ¯ ($s_2\_s_2¢$ $\circ$ Invert($d_2\_s_2¢$));
6. return $\langle d_2, d_1\_d_2, s_2\_d_2 \rangle$;
```

First-cut taxonomy of patterns



© 2004 Microsoft Corporation. All rights reserved.

Outline

- Introduction
- Meta data problems
- Design patterns
- Solution templates
- Research background
- ⇨ Wrap up

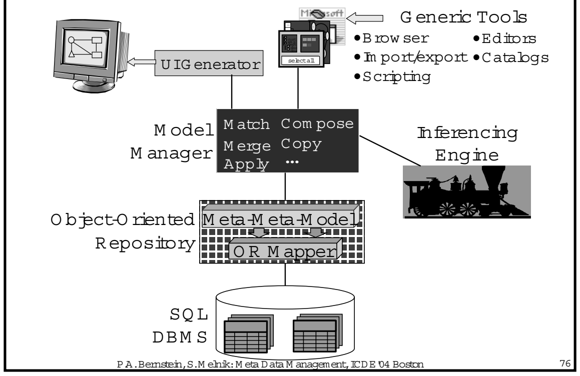
The Commercial World

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

The Research World

- Model Management
  - A computational meta data framework based on models, mappings, and the operators described here (Match, Merge, Compose,…)
- 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)

MM System Architecture



Summary

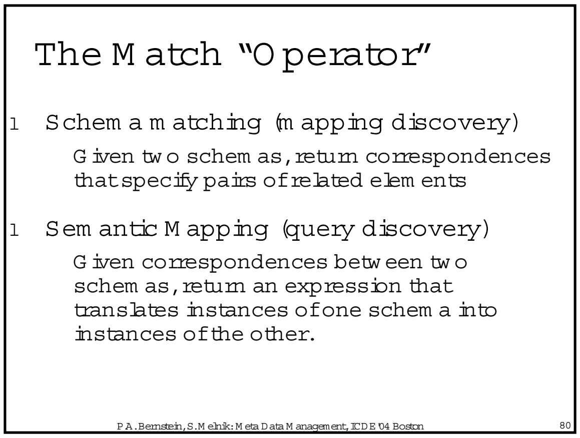
- Many DB problems are easier to solve by manipulating meta data
- Meta data problems and solutions strongly resemble one another
- Methodology: Use design patterns, solution templates, and operators to simplify development of meta data applications
- There is much research to be done

References

- http://research.microsoft.com/db/ModelMgt
- Overview
  - Bernstein, CIDR 2003
  - Bernstein, Halevy, Pottinger, SIGMOD Record, Dec.2000
- Implementation
  - Melnik, Rahm, & Bernstein, SIGMOD 2003 and J.Web Semantics 1, 2003
- Data Warehouse Examples
  - Bernstein & Rahm, ER 2000
- Match Operation
  - Survey: Rahm & Bernstein, VLDB J., Dec.2001
- Merge Operation
  - Pottinger & Bernstein, VLDB 2003

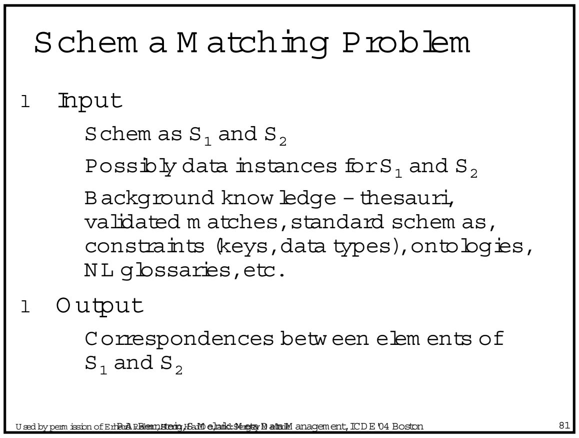
© 2004 Microsoft Corporation. All rights reserved.

© 2004 Microsoft Corporation. All rights reserved.
This presentation is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.



The Match "Operator"

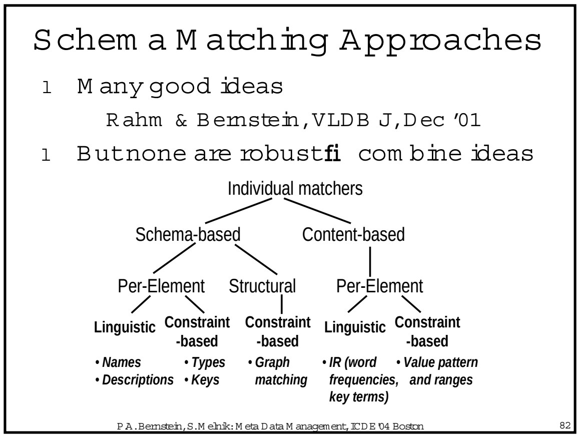
- Schema matching (mapping discovery)
  - Given two schemas, return correspondences that specify pairs of related elements
- Semantic Mapping (query discovery)
  - Given correspondences between two schemas, return an expression that translates instances of one schema into instances of the other.



Schema Matching Problem

- Input
  - Schemas S_1 and S_2
  - Possibly data instances for S_1 and S_2
  - Background knowledge – thesauri, validated matches, standard schemas, constraints (keys, data types), ontologies, NL glossaries, etc.
- Output
  - Correspondences between elements of S_1 and S_2

Used by permission of Erhard Rahm, Hong Hai Do and Sergey Melnik



Schema Matching Approaches

- Many good ideas
  - Rahm & Bernstein, VLDB J, Dec '01
- But none are robust fi combine ideas

Individual matchers

Schema-based

Content-based

Per-Element

Structural

Per-Element

Linguistic

• Names
• Descriptions

Constraint -based

• Types
• Keys

Constraint -based

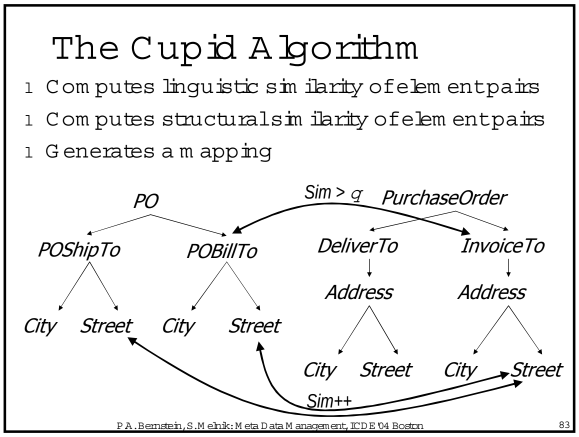
• Graph matching

Linguistic

• IR (word frequencies, key terms)

Constraint -based

• Value pattern and ranges



The Cupid Algorithm

- Computes linguistic similarity of element pairs
- Computes structural similarity of element pairs
- Generates a mapping

PO

POShipTo

City

Street

POBillTo

City

Street

Sim > q

PurchaseOrder

DeliverTo

Address

City

Street

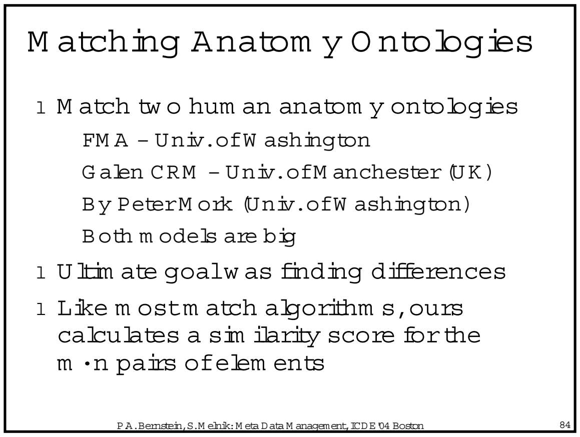
InvoiceTo

Address

City

Street

Sim++



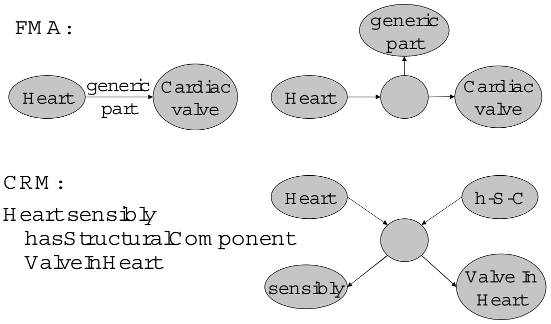
Matching Anatomy Ontologies

- Match two human anatomy ontologies
  - FMA – Univ. of Washington
  - Galen CRM – Univ. of Manchester (UK)
  - By Peter Mork (Univ. of Washington)
  - Both models are big
- Ultimate goal was finding differences
- Like most match algorithms, ours calculates a similarity score for the m ·n pairs of elements

© 2004 Microsoft Corporation. All rights reserved.

Aligning Representations

FMA:



CRM:

Heart sensibly
hasStructuralComponent
ValveInHeart

Anatomy Matching Algorithm

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

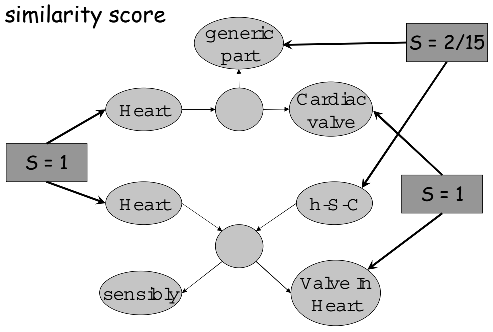
Anatomy Matching Algorithm

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

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

Anatomy Matching Example

S: similarity score

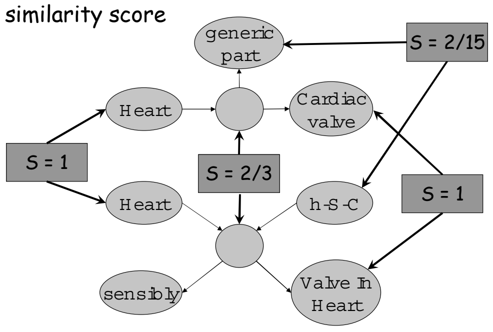


Anatomy Matching Algorithm

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

Anatomy Matching Example

S: similarity score



© 2004 Microsoft Corporation. All rights reserved.

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

- Adds 64 matches (to previous 3503)
  - Implies 875 reified relationship matches

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 Super-classes
   - Super-class similarity = average similarity of children, grandchildren, great-grandchildren

- Adds 213 matches (to 3567)

Some Lessons

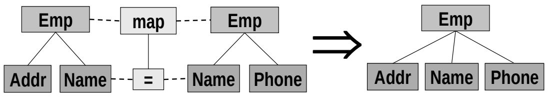
- A common encoding of models is hard and involves compromises
  - Different styles of reifying relationships
  - CRM stores transitive relationships
- Match needs to invent generalizations
  - In FMA, ***arterial supply, venous drainage, nerve supply, lymphatic drainage***
  - In CRM, these all map to ***isServedBy***
- 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
- Matching anatomy ontologies
- Model merging
- Wrap-up

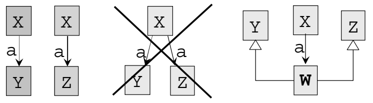
Merge(M_1, M_2, map)

- Return the union of models M_1 and M_2
  - Use map to guide the Merge
  - If elements x = y in map, then collapse them into one element



Merge(M_1, M_2, map)

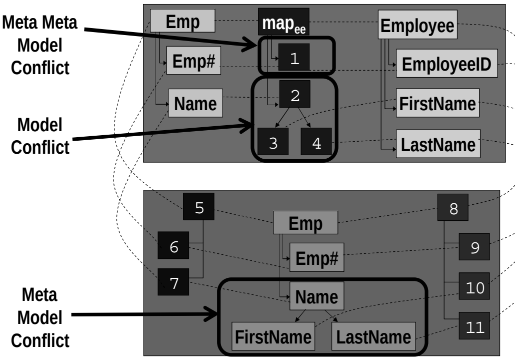
- [Buneman, Davidson, Kosky, EDBT 92]
  - Meta-model has aggregation & generalization only
  - Union, and collapse objects having the same name
  - Fix-up step for inconsistencies created by merging



- Successive fixups lead to different results **L**
- Batch them at the end, to get a unique minimal result
- Now enrich the meta-model (containment, complex mappings, …) & merge semantics (conflicts, deletes)

© 2004 Microsoft Corporation. All rights reserved.

Resolving Merge Conflicts

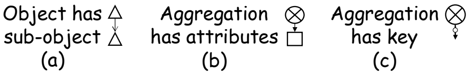


Contributions to Merge
[Pottinger & Bernstein, VLDB 03]

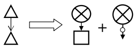
- Generic correctness criteria for Merge
- Use of first-class input mapping (not just correspondences)
- Taxonomy of conflicts & resolution strategies
- Characterize when Merge can be automatic
- A merge algorithm for an EER representation
- Experimental evaluation

An Approach to ModelGen
[Atzeni & Torlone, EDBT '96]

- Meta-models are made of patterns



- Define pattern transformations as rules
  For XSD fi SQL,



- To translate M_s into meta-model$_t$ (MM_t),
  Apply rules that replace patterns in M_S that are not in MM_t by patterns that are in MM_t

ModelGen Research

- More complete repertoire of patterns
- Make patterns more generic
- Integrate with rules engine (avoid cycles, control search)
- Implement it

Microsoft®

© 2004 Microsoft Corporation. All rights reserved.
This presentation is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

© 2004 Microsoft Corporation. All rights reserved.