IBM Research

# Model Management and Schema Mappings: Theory and Practice (Part II)

Howard Ho

IBM Almaden Research Center

For VLDB07 Tutorial with Phil Bernstein, Microsoft Research

# The Integration Challenge

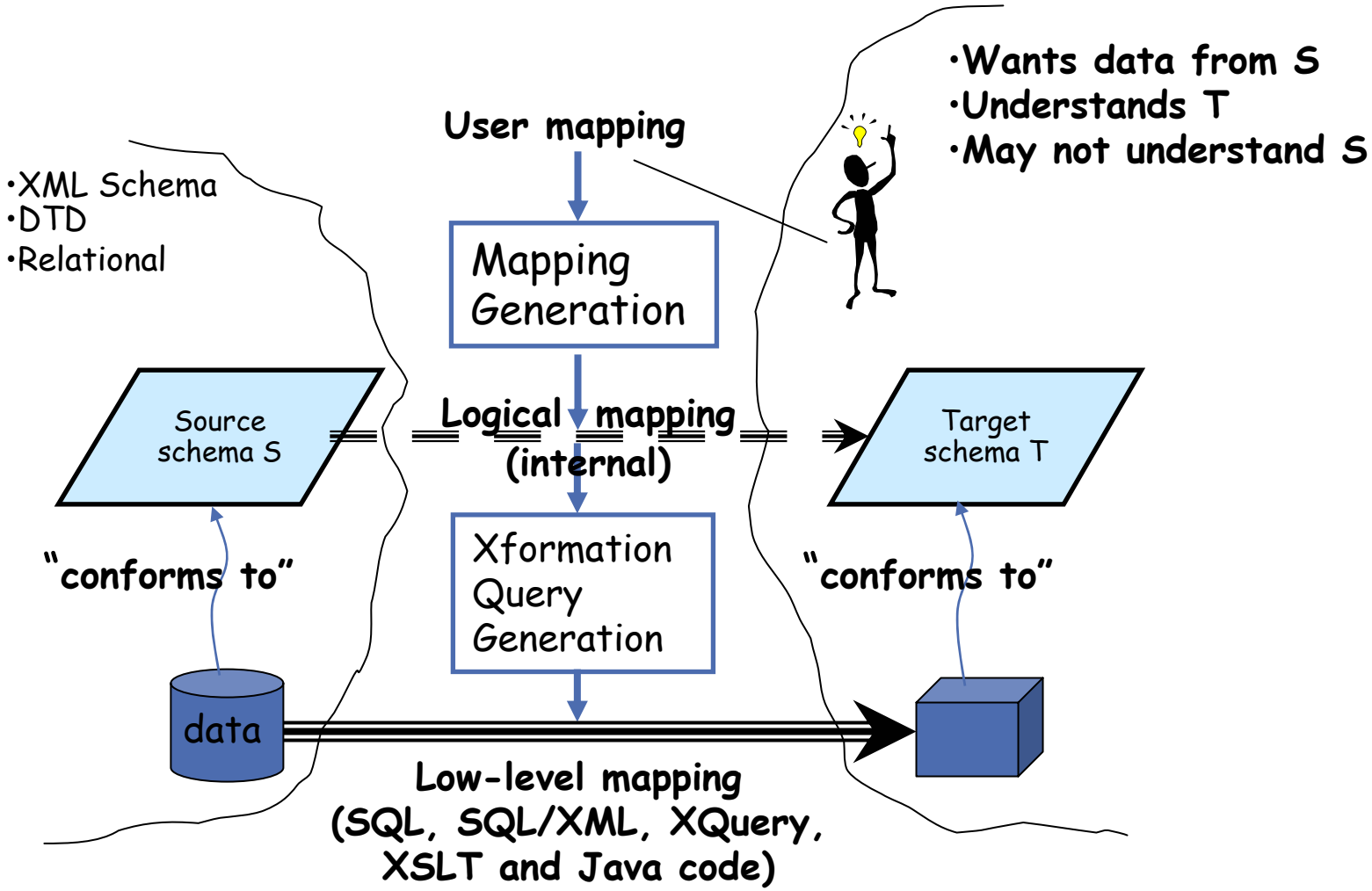
- Complex and heterogeneous environments
  - –Many different types of systems
  - –Many inter-related applications
- Escalating needs
  - –Variety, velocity, volume
- People are expensive

# Outline

- Clio: Basic Features of a Schema Mapping System
  - Schema Matching
  - Schema Mapping
  - Query Generation
  - IBM Rational Data Architect Product
- MAUI: Advanced Features of a Schema Mapping System
  - Nested Mapping Model
  - Mapping-Based XML Transformation Engine
  - Schema Integration
  - Schema Evolution
- Clio2010: Mapping-Based Authoring of Data Flows
  - ETL: "Mapping ↔ ETL" Conversion
  - Web-Service Composition: Mapping for web-service data sources
  - Mashups: "Mapping → Mashup" Generation
  - Reuse: Mapping Polymorphism
- Conclusions and Future Directions



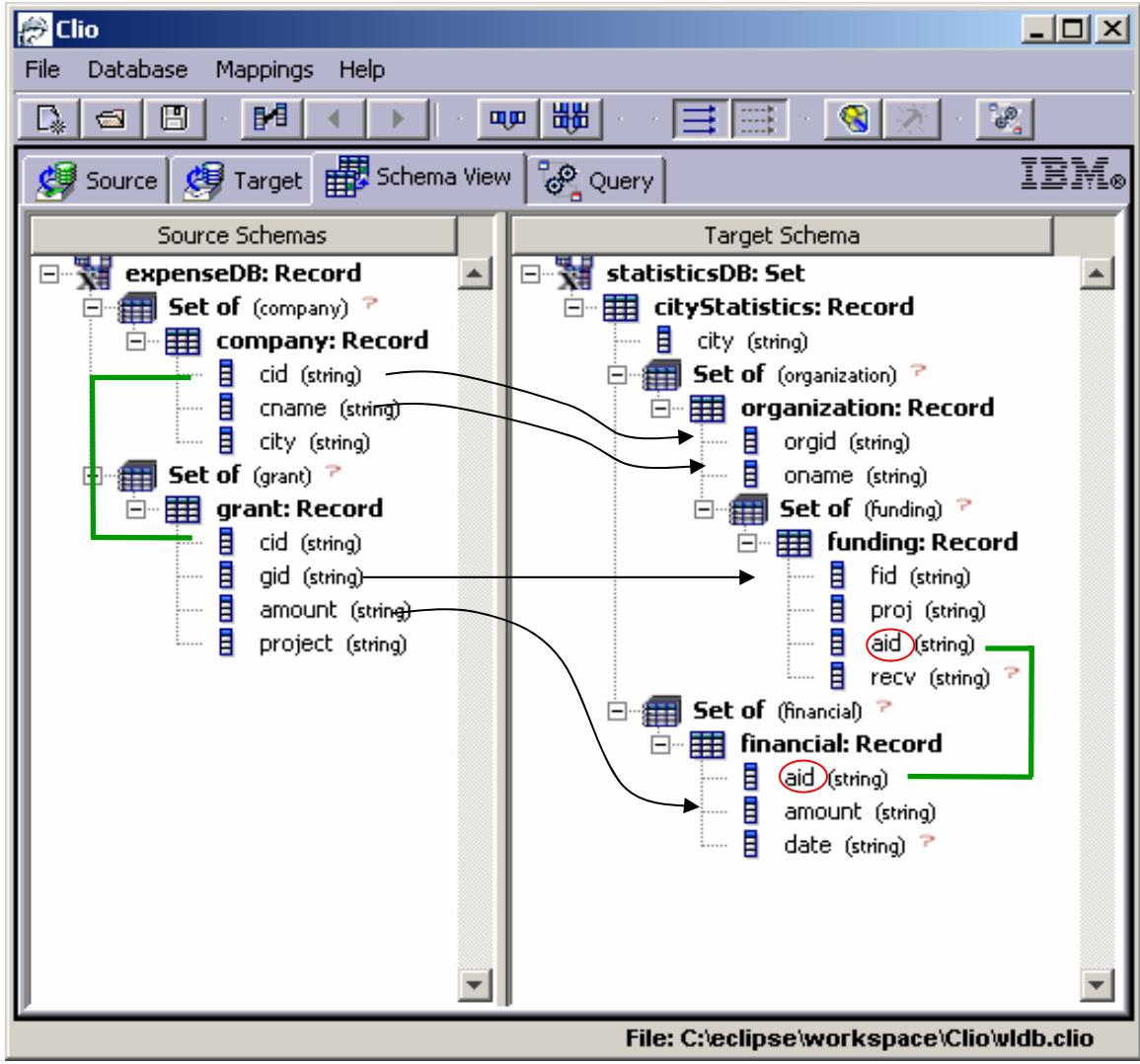
# Clio: A Schema Mapping System




Logical mappings can be used for both **target materialization** or **query rewriting**

# Major Features (and Challenges)

- Schemas can be arbitrarily different
- Element correspondences
  - Human friendly
  - Automatic discovery
- Support Nested Structures
  - Nested Relational Model
  - Nested Constraints
- Produce Correct Grouping
- Preserve data meaning
  - Discover associations
  - Use constraints & schema
- Create New Target Values
- and …



# Generate Transformation Queries (XQuery)

```
<?xml version="1.0" encoding="UTF-8"?>
<statisticsDB>
   <cityStatistics>
      <city/>,
      distinct (
      FOR
       $x0 IN $doc/expenseDB/grant,
       $x1 IN $doc/expenseDB/company
      WHERE
         $x1/cid/text() = $x0/cid/text()
      RETURN
         <organization>
            <orgid>   $x0/cid/text()  </orgid>,
            <oname>   $x1/cname/text()  </oname>,
            distinct (
            FOR
             $x0L1 IN $doc/expenseDB/grant,
             $x1L1 IN $doc/expenseDB/company
            WHERE
               $x1L1/cid/text() = $x0L1/cid/text() AND
               $x1/cname/text() = $x1L1/cname/text() AND
               $x0/cid/text() = $x0L1/cid/text()
            RETURN
               <funding>
                  <fid>   "Sk35(", $x0L1/amount/text(), ", ", $x1L1/cname/text(), ", ", $x0L1/cid/text(), ")"  </fid>,
                  <proj>   "Sk36(", $x0L1/amount/text(), ", ", $x1L1/cname/text(), ", ", $x0L1/cid/text(), ")"  </proj>,
                  <aid>   "Sk32(", $x0L1/amount/text(), ", ", $x1L1/cname/text(), ", ", $x0L1/cid/text(), ")"  </aid>
               </funding> )
         </organization> ),
      distinct (
      FOR
       $x0 IN $doc/expenseDB/grant,
       $x1 IN $doc/expenseDB/company
      WHERE
         $x1/cid/text() = $x0/cid/text()
      RETURN
         <financial>
            <aid>   "Sk32(", $x0/amount/text(), ", ", $x1/cname/text(), ", ", $x0/cid/text(), ")"  </aid>,
            <amount>   $x0/amount/text()  </amount>
         </financial> )
   </cityStatistics>
</statisticsDB>
```

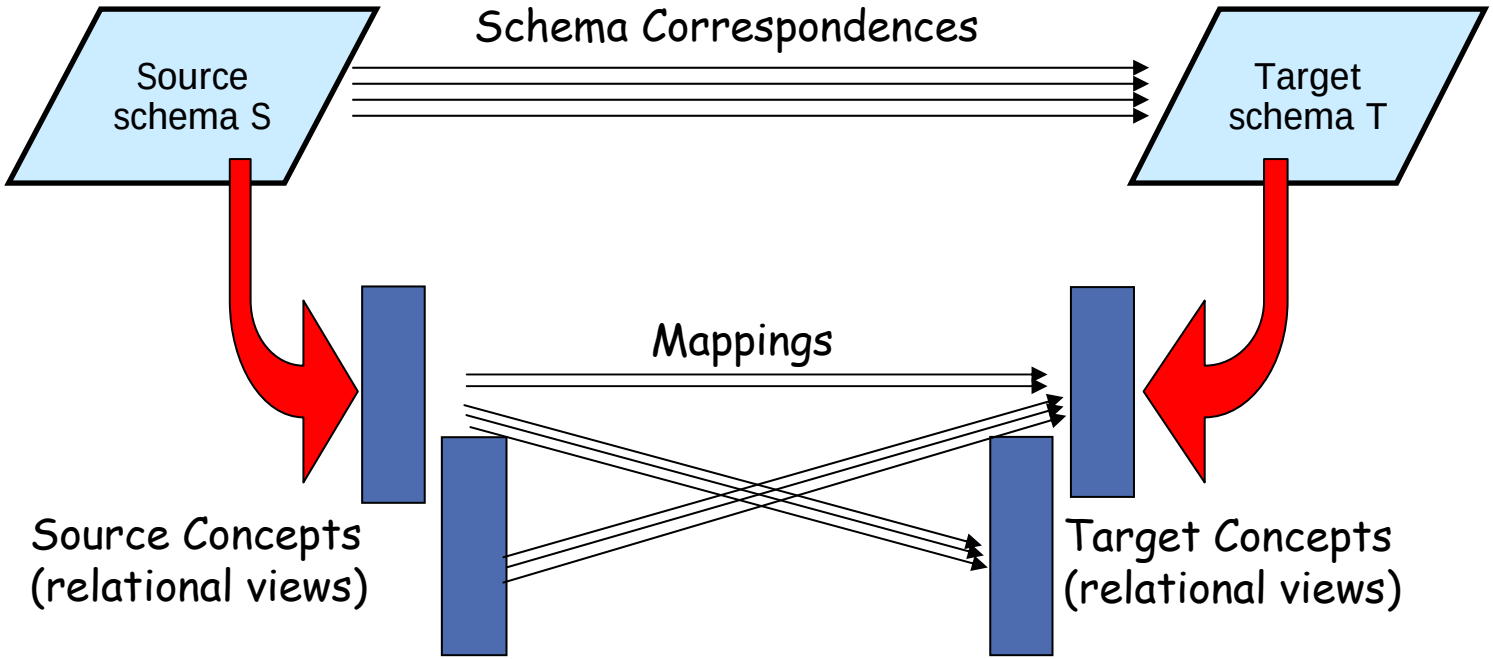


# Generate Transformation Scripts (XSLT)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/">
        <result>
            <xsl:call-template name="q0"/>
        </result>
    </xsl:template>
    <xsl:template name="q0">
        <xsl:element name="statisticsDB">
            <xsl:attribute name="isRoot">true</xsl:attribute>
            <xsl:element name="ClioSet">
                <xsl:attribute name="id">Sk_statisticsDB()</xsl:attribute>
            </xsl:element>
        </xsl:element>
        <xsl:for-each select="/expenseDB/grant"> <xsl:variable name="x0" select="."/>
        <xsl:for-each select="/expenseDB/company"> <xsl:variable name="x1" select="."/>
        <xsl:if test="$x1/cid=$x0/cid">
            <xsl:element name="cityStatistics">
                <xsl:attribute name="inSet">Sk_statisticsDB()</xsl:attribute>
                <xsl:element name="city"/>
                <xsl:element name="ClioSet">
                    <xsl:attribute name="id">Sk_statisticsDB_0_1(Sk_statisticsDB())</xsl:attribute>
                </xsl:element>
                <xsl:element name="ClioSet">
                    <xsl:attribute name="id">Sk_statisticsDB_0_2(Sk_statisticsDB())</xsl:attribute>
                </xsl:element>
            </xsl:element>
            <xsl:element name="organization">
                <xsl:attribute name="inSet">Sk_statisticsDB_0_1(Sk_statisticsDB())</xsl:attribute>
                <xsl:element name="orgid"><xsl:value-of select="$x0/cid"/></xsl:element>
                <xsl:element name="oname"><xsl:value-of select="$x1/cname"/></xsl:element>
                <xsl:element name="ClioSet">
                    <xsl:attribute name="id">Sk_statisticsDB_0_1_0_2(<xsl:value-of select="$x0/cid"/>,
                                                                     <xsl:value-of select="$x1/cname"/>,
                                                                    Sk_statisticsDB_0_1(Sk_statisticsDB()))
                    </xsl:attribute>
                </xsl:element>
            </xsl:element>
            <xsl:element name="funding">
                <xsl:attribute name="inSet">Sk_statisticsDB_0_1_0_2(<xsl:value-of select="$x0/cid"/>,
                                                                    <xsl:value-of select="$x1/cname"/>,
                                                                    Sk_statisticsDB_0_1(Sk_statisticsDB()))
                 </xsl:attribute>
                <xsl:element name="fid">
                        Sk35(<xsl:value-of select="$x0/amount"/>, <xsl:value-of select="$x1/cname"/>,
                             <xsl:value-of select="$x0/cid"/>)
                </xsl:element>
```



# (Flat) Mapping Generation




- **Step 1**. Extraction of "concepts" (in each schema).
  - Concept = one category of data that can exist in the schema
- **Step 2**. Mapping generation
  - Enumerate all non-redundant maps between pairs of concepts
- [Popa, Velegrakis, Miller, Hernandez, Fagin. VLDB02]
- [Fagin, Kolatios, Miller, Popa. ICDT 03]
- [Haas, Hernandez, Ho, Popa, Roth. SIGMOD 05]

# (Flat) Mapping Example

- $m_1$ maps **proj** to **dept-projects**

$m_1$:

$\forall(p_0 \text{ in proj})$

$\exists(d \text{ in dept}) \exists(p \text{ in } d.\text{projects})$

$p_0.\text{dname} = d.\text{dname}$

$\wedge\ p_0.\text{pname} = p.\text{pname}$

The concept of "project of a department"

$m_2$:

$\forall(p_0 \text{ in proj}) \forall(e_0 \text{ in } p_0.\text{emps})$

$\exists(d \text{ in dept}) \exists(p \text{ in } d.\text{projects})$

$\exists(e \text{ in } d.\text{emps}) \exists(w \text{ in } e.\text{worksOn})$

$w.\text{pid} = p.\text{pid}$

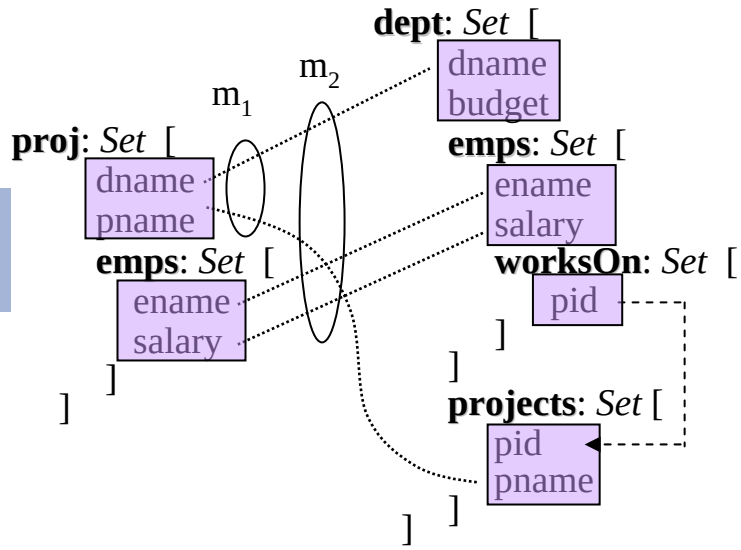
$\wedge\ p_0.\text{dname} = d.\text{dname}$

$\wedge\ p_0.\text{pname} = p.\text{pname}$

$\wedge\ e_0.\text{ename} = e.\text{ename}$

$\wedge\ e_0.\text{salary} = e.\text{salary}$

expression for **dept-emps-worksOn-projects**

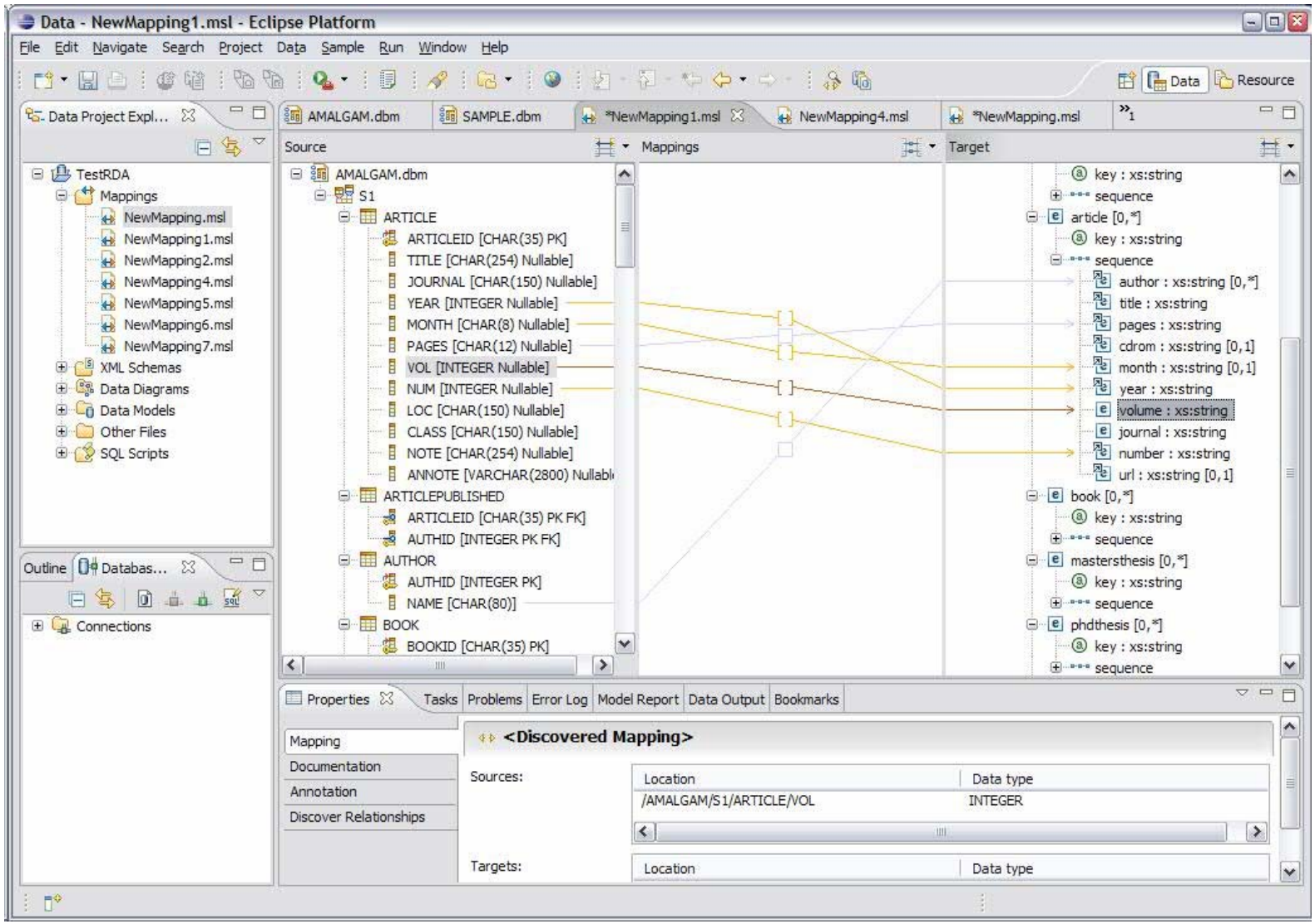



- $m_2$ maps **proj-emps** to **dept-emps-worksOn-projects**

The concept of "project of an employee of a department"

- Two 'basic' mappings (or *source-to-target tgds or GLAV formulas*)

# IBM Rational Data Architect Product



# IBM Rational Data Architect Product

- Schema Matching, Schema Mapping and Query Generation Technologies from Clio
- Value Correspondences in the GUI
  - Blue Lines: Confirmed by the users
  - Gray Lines: Suggested by the schema matching algorithms
- Schema Matching
  - Five different algorithms: two name-based (including thesaurus lookup) and three instance-based
  - Users can choose
    - Any weighted combination of the 5 schema matching algorithms
    - Source or target
    - One element (element/attribute or column) or a group of elements (subtree or table)
    - The value k (for the top-k matches)
  - The system returns the top-k matches for each element
- Current Release
  - Source is relational (other IBM products support XML sources)
  - Target can be relational (generates SQL) or XML (generates SQL/XML)
  - Mapping is standardized within IBM, as an EMF in-memory object and as a serialized XML document

# Outline

- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
  - Nested Mapping Model
    - [Fuxman, Hernandez, Ho, Miller, Papotti, Popa. VLDB 06]
  - Mapping-Based XML Transformation Engine
  - Schema Integration
  - Schema Evolution
- Clio2010: Mapping-Based Authoring of Data Flows
- Conclusions and Future Directions



# New Nested-Mapping Engine for Clio

- Existing Clio engine is based on a flat mapping model
  - Pros: easier to implement
  - Cons:
    - Fragmentation into many overlapping mappings
    - Inefficiency in execution
    - Redundancy in the output data
    - No user-defined grouping semantics
- New Clio engine is based on a nested-mapping model
  - Cons: more challenging to design and implement
  - Pros: overcomes the above problems in the flat mapping model



# (Flat) Mapping Example

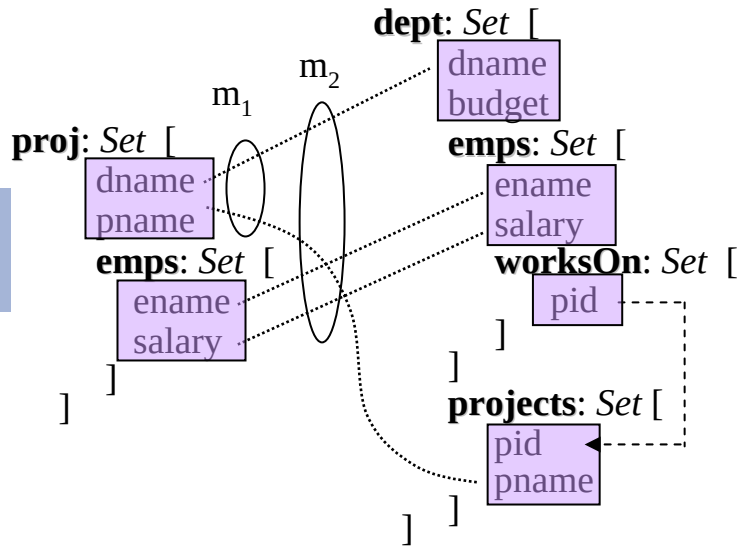
- $m_1$ maps **proj** to **dept-projects**

$m_1$:
$\forall(p_0 \text{ in proj})$
$\exists(d \text{ in dept}) \exists(p \text{ in } d.\text{projects})$
$p_0.\text{dname} = d.\text{dname}$
$\wedge\ p_0.\text{pname} = p.\text{pname}$

$m_2$:
$\forall(p_0 \text{ in proj}) \forall(e_0 \text{ in } p_0.\text{emps})$
$\exists(d \text{ in dept}) \exists(p \text{ in } d.\text{projects})$
$\exists(e \text{ in } d.\text{emps}) \exists(w \text{ in } e.\text{worksOn})$
$w.\text{pid} = p.\text{pid}$
$\wedge\ p_0.\text{dname} = d.\text{dname}$
$\wedge\ p_0.\text{pname} = p.\text{pname}$
$\wedge\ e_0.\text{ename} = e.\text{ename}$
$\wedge\ e_0.\text{salary} = e.\text{salary}$

The concept of "project of a department"

expression for **dept-emps-worksOn-projects**




- $m_2$ maps **proj-emps** to **dept-emps-worksOn-projects**

The concept of "project of an employee of a department"

- Two 'basic' mappings (or *source-to-target tgds or GLAV formulas*)

# Correlating Mapping Formulas

$m_1$: $\forall(p_0 \text{ in proj})$
$\exists(d \text{ in dept}) \exists(p \text{ in } d.\text{projects})$
$p_0.\text{dname} = d.\text{dname} \wedge p_0.\text{pname} = p.\text{pname}$

$m_2$: $\forall(p_0 \text{ in proj}) \forall(e_0 \text{ in } p_0.\text{emps})$
$\exists(d \text{ in dept}) \exists(p \text{ in } d.\text{projects}) \exists(e \text{ in } d.\text{emps}) \exists(w \text{ in } e.\text{worksOn})$
$w.\text{pid}=p.\text{pid}$
$\wedge\ p_0.\text{dname} = d.\text{dname} \wedge p_0.\text{pname} = p.\text{pname}$
$\wedge\ e_0.\text{ename} = e.\text{ename} \wedge e_0.\text{salary} = e.\text{salary}$

proj tuples mapped only once

Submapping, correlated to the parent mapping

For every proj tuple, we map all employees, as a group.

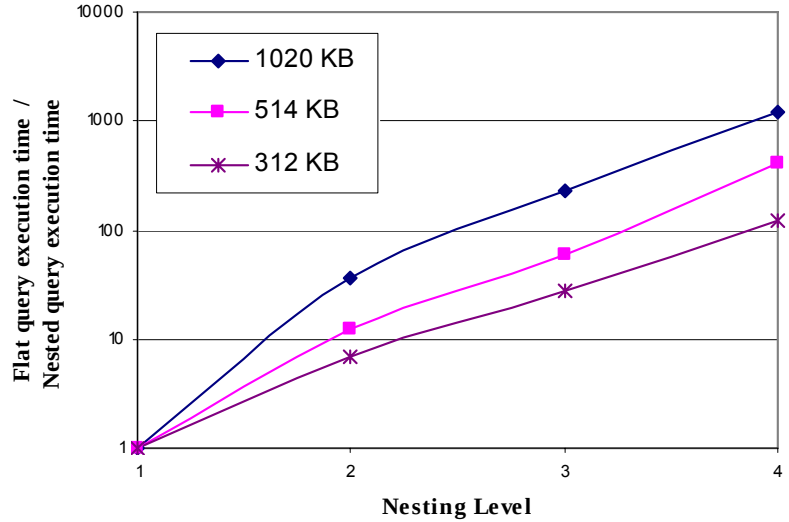
(Source grouping is preserved)

Replace with

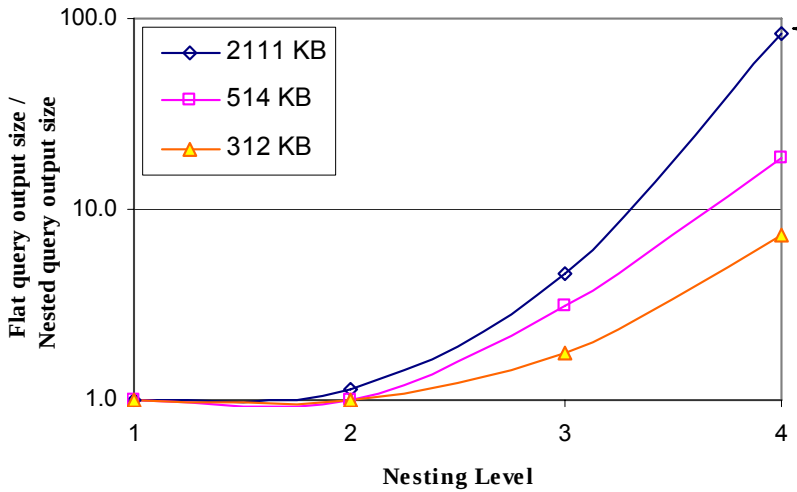
n: $\forall(p_0 \text{ in proj})$
$\exists(d \text{ in dept}) \exists(p \text{ in } d.\text{projects})$
$p_0.\text{dname} = d.\text{dname} \wedge p_0.\text{pname} = p.\text{pname}$
$\wedge\ [\ \forall(e_0 \text{ in } \mathbf{p_0}.\text{emps})$
$\exists(e \text{ in } \mathbf{d}.\text{emps}) \exists(w \text{ in } e.\text{worksOn})$
$w.\text{pid}=\mathbf{p}.\text{pid}$
$\wedge\ e_0.\text{ename} = e.\text{ename} \wedge e_0.\text{salary} = e.\text{salary}$
$]$

This is a nested mapping

# Performance




Execution time for flat: **22mins**
Execution time for nested: **1.1s**




- Size of generated data (flat) – including duplicates: **45MB**
- Size of generated data (nested): **552KB**

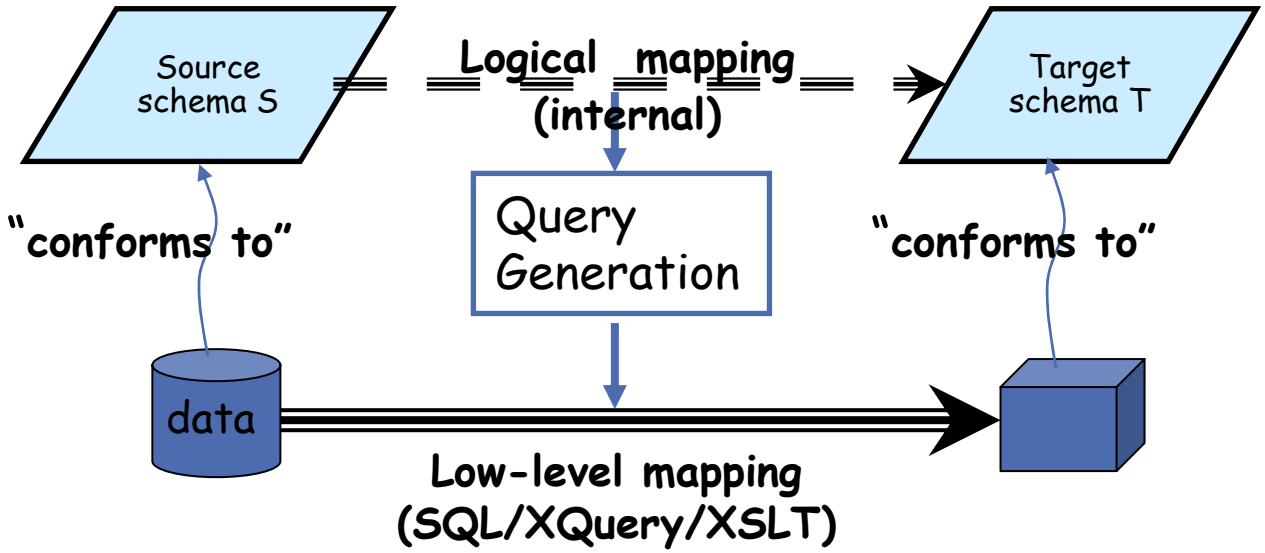
The nested mapping generates much more efficient execution and less redundant data

# Outline

- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
  - Nested Mapping Model
  - Mapping-Based XML Transformation Engine
    - [Jiang, Ho, Popa, Han. WWW 07]
  - Schema Integration
  - Schema Evolution
- Clio2010: Mapping-Based Authoring of Data Flows
- Conclusions and Future Directions

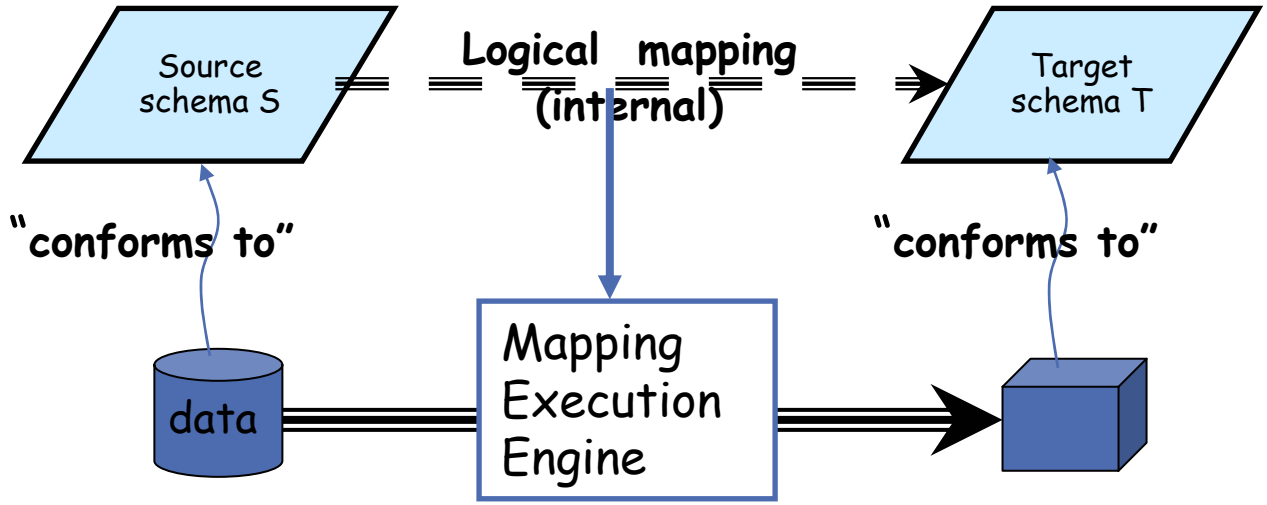


# Performance in Executing Mappings




- Mappings are translated into general purpose query languages
  - E.g., SQL, XQuery, XSLT
- Query optimization issues are left to the runtime engine of each of these languages to decide
  - i.e., Q.O. decisions are not encoded in the queries.
- Idea: Execute mappings directly in our own runtime.

# Mapping Execution Engine in Java



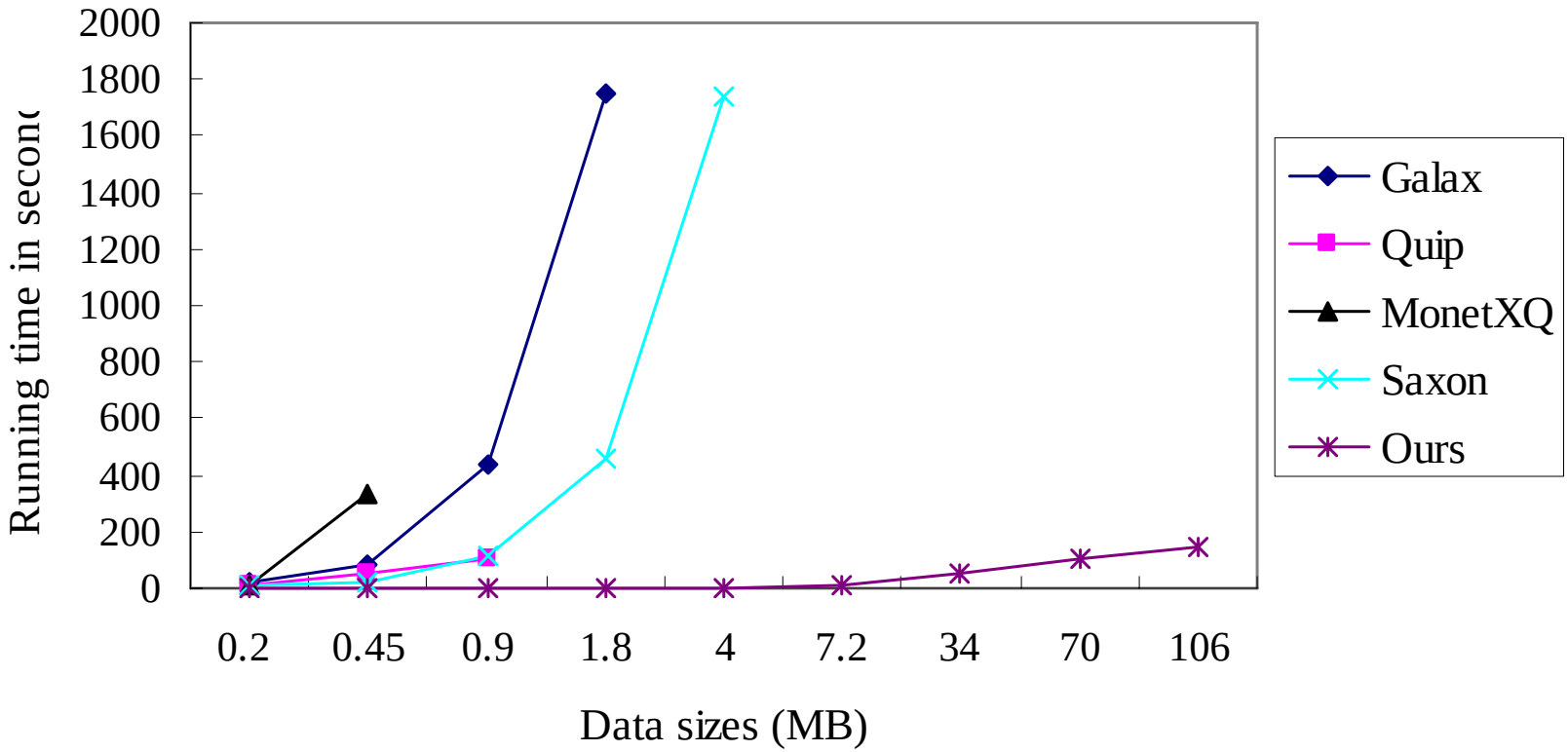

- Motivation
  - Grouping (on multiple levels) of transformed values.
  - XQuery & XSLT have no specific constructs (hence algorithms) for such grouping tasks
- Scalability and efficiency
  - Controllable memory resource usage
  - Speed of transformation almost linear to input sizes
- Mapping-based
  - Model high-level mapping semantics using IBM Mapping Specification Language (MSL) standard

# Three-Phase Algorithm for Executing Mappings

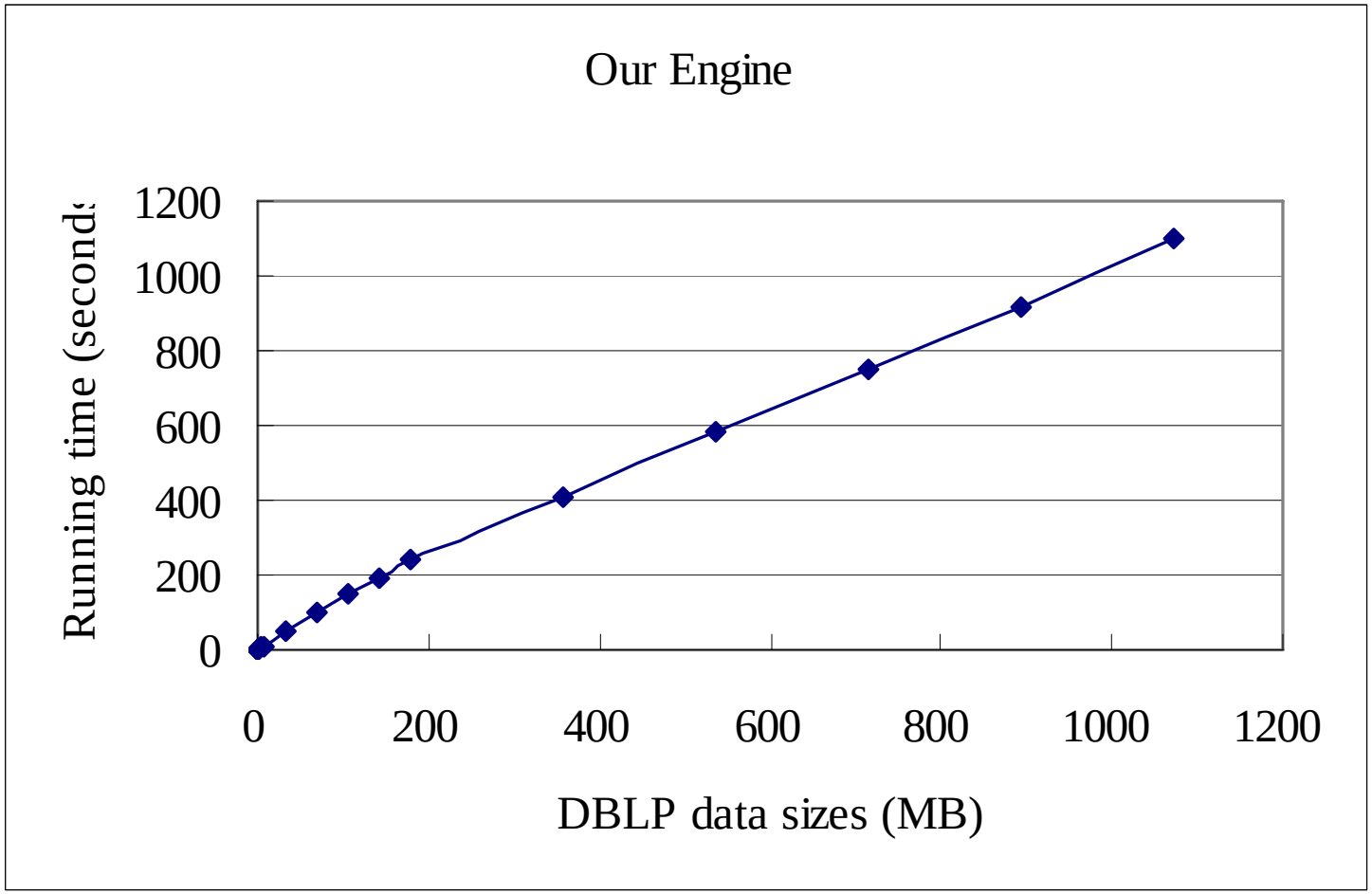
- Phase 1 (Extract): tuple extraction
  - Streaming holistic twig join
    - Streaming and stored/indexed XML
  - SQL queries through ODBC
    - Relational source
- Phase 2 (Transform): generate an XML tree from each tuple
- Phase 3 (Merge): data merging
  - A dynamic, scalable merge algorithm
    - Hash-based vs. sort-based algorithms



# XML-to-XML Comparative Results



# XML-to-XML Scalability Results



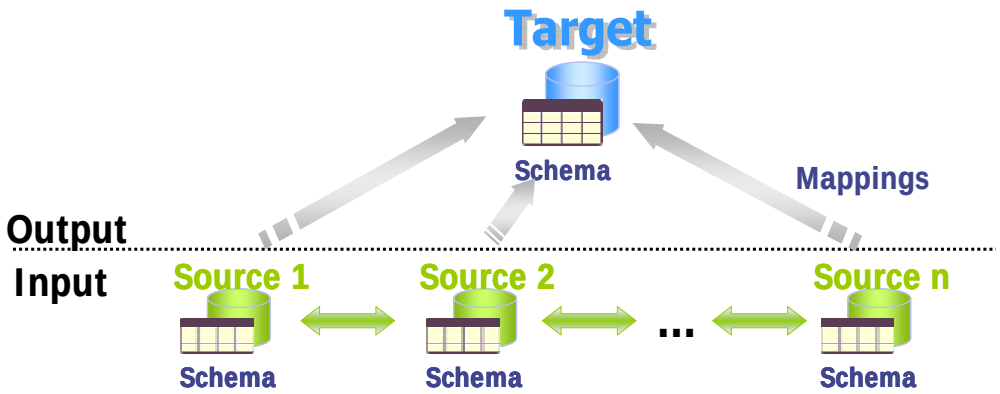
# Outline

- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
  - Nested Mapping Model
  - Mapping-Based XML Transformation Engine
  - Schema Integration
    - [Chiticariu, Hernandez, Popa, Kolaitis. VLDB 07 demo]
  - Schema Evolution
- Clio2010: Mapping-Based Authoring of Data Flows
- Conclusions and Future Directions



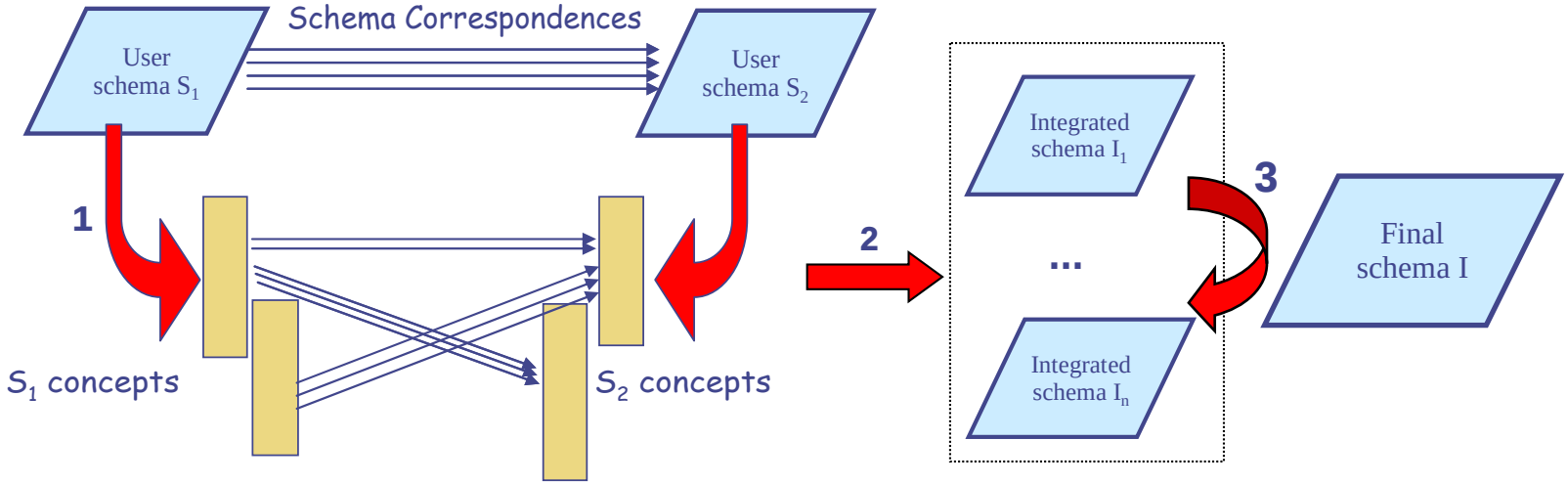
# Schema Integration

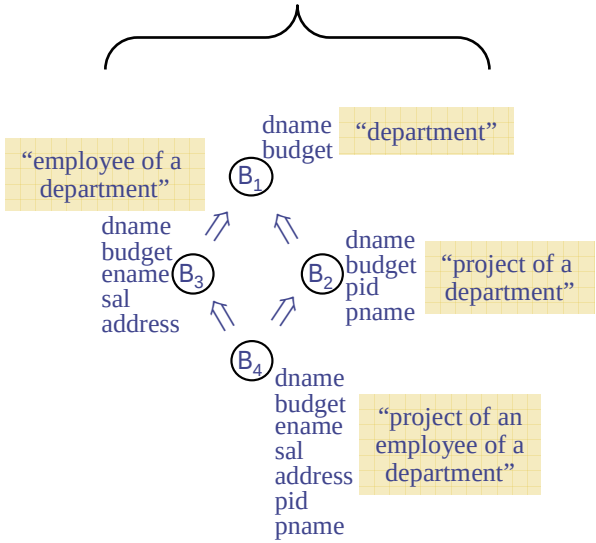
- Problem
  - Given multiple overlapping schemas in the same domain, consolidate them into one.



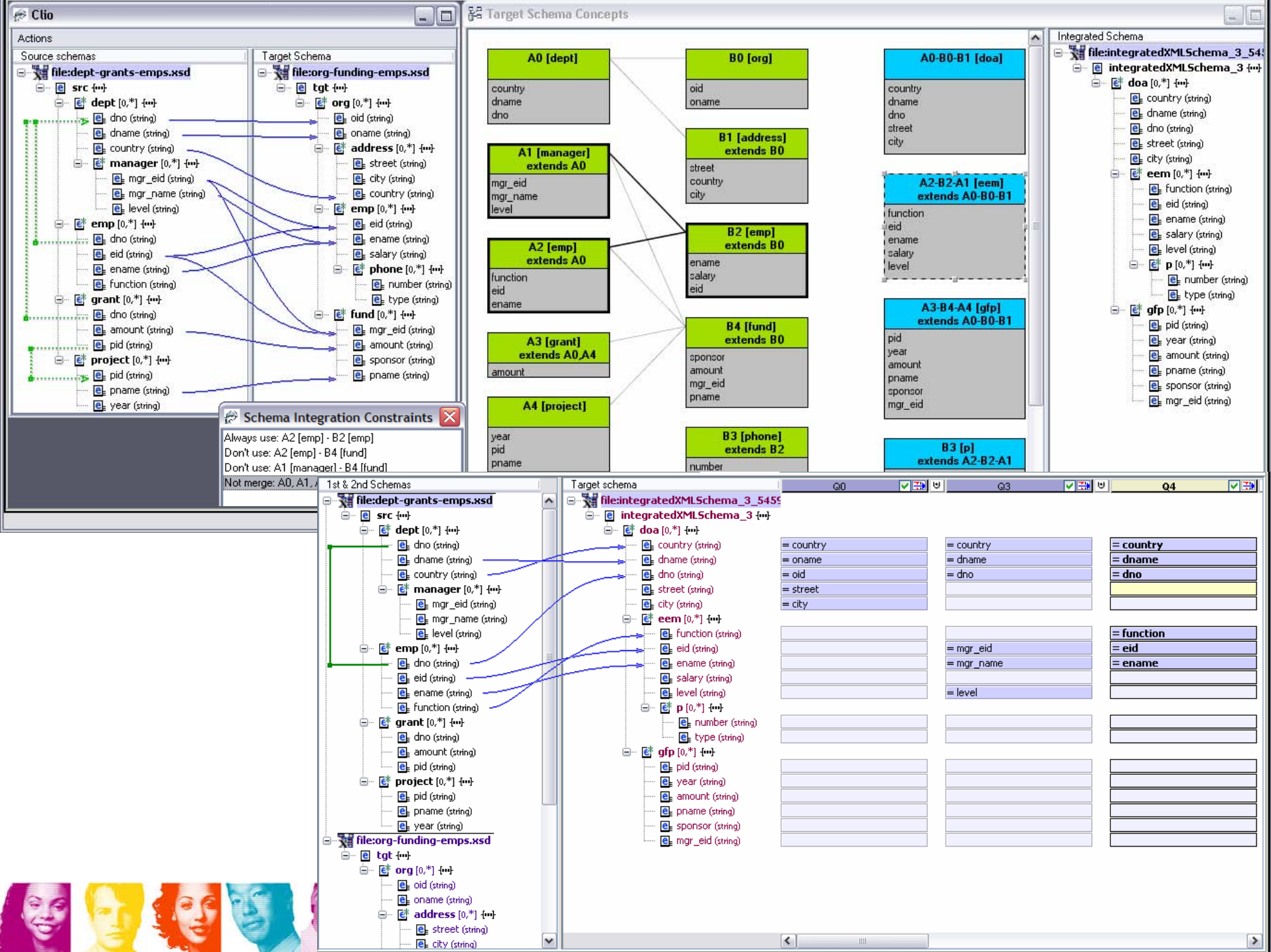

- Applications
  - Provide a standard representation of the data (for unified querying, data warehousing, reference point, etc.)
  - "Metadata chaos reduction"
- Schema integration is a hard problem
  - Requires a lot of human interaction/feedback
  - A series of "recipes" that arrive at a single integrated schema
- Related work
  - [Pottinger, Bernstein. VLDB 03]
  - [Chiticariu, Hernandez, Popa, Kolaitis. VLDB 07 demo]

# Schema Integration [Chiticariu et al.]




- **Step 1**. Extraction of "concepts" from a hierarchy
- **Step 2**. Consider all possible ways of performing a hierarchical merge of the related concepts
  - Generate initial set of candidate "good" integrated schemas
  - ***Duplication-free enumeration algorithm***
    - Avoids duplicates by using constraints (Horn clauses)
    - *Polynomial-delay algorithm* for enumerating satisfying assignments
- **Step 3.** Browse/Search/Refine set of candidate schemas
  - ***Combination of partial enumeration with user constraints***
  - Based on the schemas seen so far, users express constraints on the "future" schemas to be generated

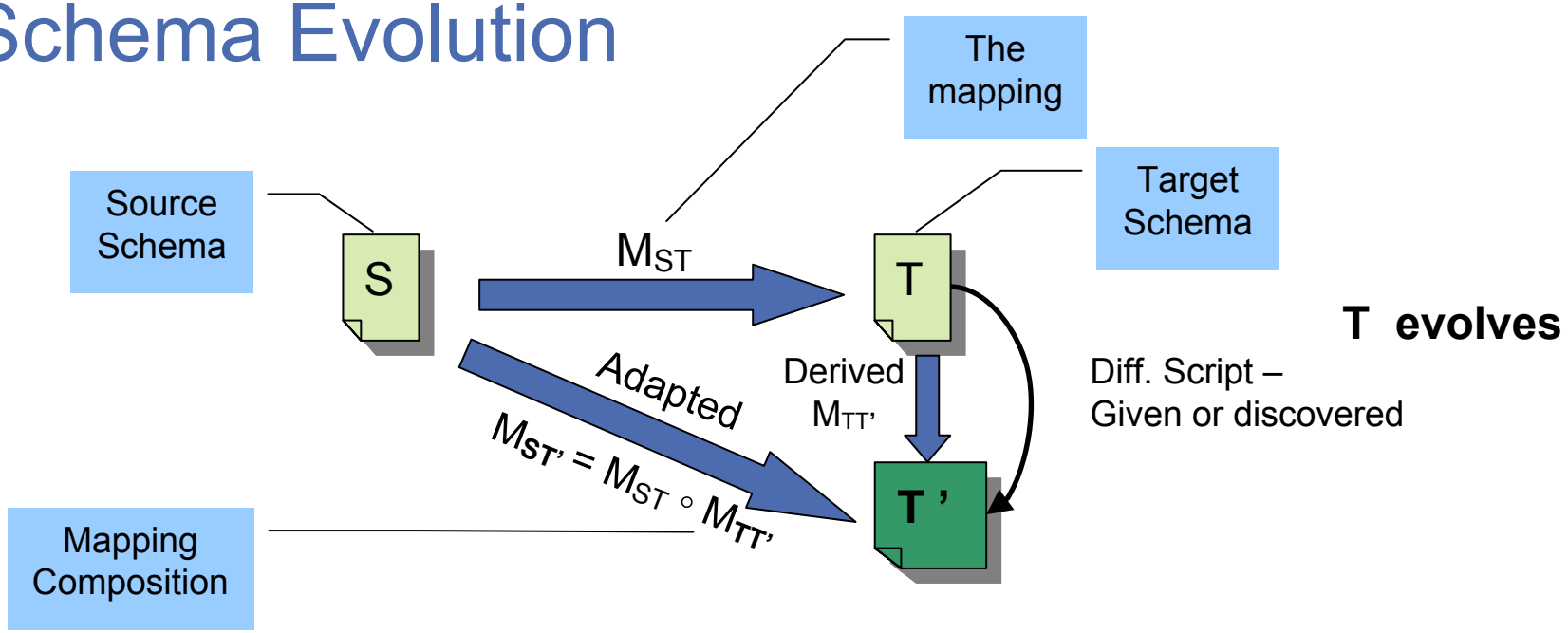

Clio
Actions
Source schemas
file:dept-grants-emps.xsd
src
dept [0,*]
dno (string)
dname (string)
country (string)
manager [0,*]
mgr_eid (string)
mgr_name (string)
level (string)
emp [0,*]
dno (string)
eid (string)
ename (string)
function (string)
grant [0,*]
dno (string)
amount (string)
pid (string)
project [0,*]
pid (string)
pname (string)
year (string)
Target Schema
file:org-funding-emps.xsd
tgt
org [0,*]
oid (string)
oname (string)
address [0,*]
street (string)
city (string)
country (string)
emp [0,*]
eid (string)
ename (string)
salary (string)
phone [0,*]
number (string)
type (string)
fund [0,*]
mgr_eid (string)
amount (string)
sponsor (string)
pname (string)
Schema Integration Constraints
Always use: A2 [emp] - B2 [emp]
Don't use: A2 [emp] - B4 [fund]
Don't use: A1 [manager] - B4 [fund]
Not merge: A0, A1,
Target Schema Concepts
A0 [dept]
country
dname
dno
A1 [manager] extends A0
mgr_eid
mgr_name
level
A2 [emp] extends A0
function
eid
ename
A3 [grant] extends A0,A4
amount
A4 [project]
year
pid
pname
B0 [org]
oid
oname
B1 [address] extends B0
street
country
city
B2 [emp] extends B0
ename
salary
eid
B4 [fund] extends B0
sponsor
amount
mgr_eid
pname
B3 [phone] extends B2
number
A0-B0-B1 [doa]
country
dname
dno
street
city
A2-B2-A1 [eem] extends A0-B0-B1
function
eid
ename
salary
level
A3-B4-A4 [gfp] extends A0-B0-B1
pid
year
amount
pname
sponsor
mgr_eid
B3 [p] extends A2-B2-A1
Integrated Schema
file:integratedXMLSchema_3_545
integratedXMLSchema_3
doa [0,*]
country (string)
dname (string)
dno (string)
street (string)
city (string)
eem [0,*]
function (string)
eid (string)
ename (string)
salary (string)
level (string)
p [0,*]
number (string)
type (string)
gfp [0,*]
pid (string)
year (string)
amount (string)
pname (string)
sponsor (string)
mgr_eid (string)
1st & 2nd Schemas
file:dept-grants-emps.xsd
src
dept [0,*]
dno (string)
dname (string)
country (string)
manager [0,*]
mgr_eid (string)
mgr_name (string)
level (string)
emp [0,*]
dno (string)
eid (string)
ename (string)
function (string)
grant [0,*]
dno (string)
amount (string)
pid (string)
project [0,*]
pid (string)
pname (string)
year (string)
file:org-funding-emps.xsd
tgt
org [0,*]
oid (string)
oname (string)
address [0,*]
street (string)
city (string)
Target schema
Q0
Q3
Q4
file:integratedXMLSchema_3_5459
integratedXMLSchema_3
doa [0,*]
country (string) = country | = country | = country
dname (string) = oname | = dname | = dname
dno (string) = oid | = dno | = dno
street (string) = street
city (string) = city
eem [0,*]
function (string) = function
eid (string) = mgr_eid = eid
ename (string) = mgr_name = ename
salary (string)
level (string) = level
p [0,*]
number (string)
type (string)
gfp [0,*]
pid (string)
year (string)
amount (string)
pname (string)
sponsor (string)
mgr_eid (string)

# Outline

- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
  - Nested Mapping Model
  - Mapping-Based XML Transformation Engine
  - Schema Integration
  - Schema Evolution
- Clio2010: Mapping-Based Authoring of Data Flows
- Conclusions and Future Directions

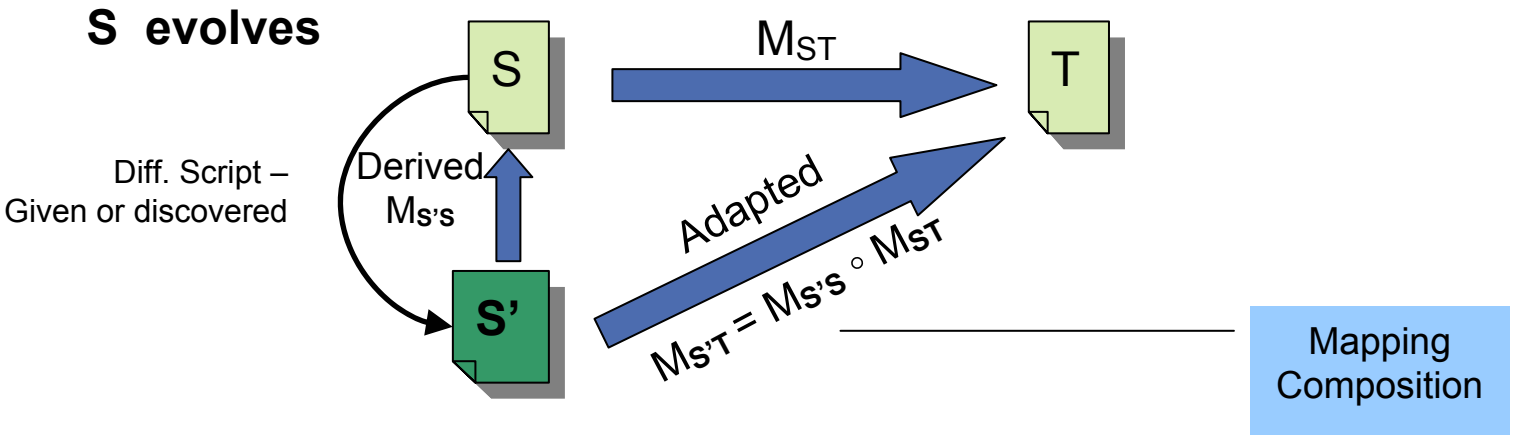


# Schema Evolution




- When the target schema evolves, the system can generate an initial default mapping for T → T' automatically
- The user adds "new mapping lines" for new schema elements in T' to complete the mapping $M_{TT'}$ (for T → T')
- The new mapping $M_{ST'}$ (S → T') is a composition of $M_{ST}$ (S → T) and $M_{TT'}$ (T → T')

# Schema Evolution




- When the source schema evolves, the system can generate an initial default mapping for S' → S automatically
- The user adds "new mapping lines" for new schema elements in S' to complete the mapping $M_{S'S}$ (for S' → S)
- The new mapping $M_{S'T}$ (S' → T) is a composition of $M_{S'S}$ (S' → S) and $M_{ST}$ (S → T)

# Related Theory and Algorithms

- Mapping Composition
  - [Madhavan, Halevy. VLDB 03]
  - [Fagin, Kolaitis, Popa, Tan. PODS 04]
  - [Nash, Bernstein, Melnik. PODS 05]
  - [Bernstein, Green, Melnik, Nash. VLDB 06]
- Mapping Inversion
  - [Fagin. PODS 06]
  - [Fagin, Kolaitis, Popa, Tan. PODS 07]
- Schema Evolution
  - [Rahm, Bernstein. SIGMOD Rec. Dec 06]
- Mapping Adaptation under Evolving Schemas
  - [Velegrakis, Miller, Popa. VLDB 03]
  - [Yu, Popa. VLDB 05]
- Query rewrite
  - [Yu, Popa. SIGMOD 04]



# Outline

- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
- Clio2010: Mapping-Based Authoring of Data Flows
  - ETL: "Mapping ↔ ETL" Conversion
  - Web-Service Composition: Mapping for web-service data sources
  - Mashups: "Mapping → Mashup" Generation
  - Reuse: Mapping Polymorphism
- Conclusions and Future Directions



# Mapping-Based Authoring of Data Flows

- Motivation
  - Schema mappings are the building blocks for larger data transformation and integration applications
  - The graph of mappings is a declarative specification of the flow of data
    - Similar to ETL & mashups
    - Need to take functional (web-service) data sources
  - Need to extend Clio where multiple mappings can be defined, loaded (sharing a context), reused and managed



# Clio2010

- Clio2010
  - Automatically assemble a graph of initial, uncorrelated mappings into larger, richer mappings
  - Support more complicated
    - mapping composition and
    - mapping merge
  - Support functional data sources (e.g., web services)
  - Compile the larger mappings into a global execution plan in
    - Queries (XQuery)
    - transformation scripts (XSLT)
    - ETL flows (e.g., IBM WebSphere DataStage)
    - mashups (e.g., IBM DAMIA)
  - Support mapping reuse through mapping polymorphism framework



# Outline

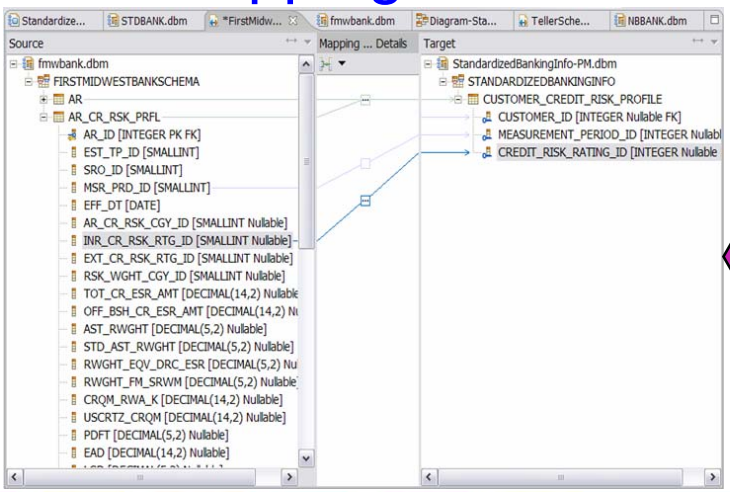
- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
- Clio2010: Mapping-Based Authoring of Data Flows
  - ETL: "Mapping ↔ ETL" Conversion
    - [Hernandez et al. 07]
  - Web-Service Composition: Mapping for web-service data sources
  - Mashups: "Mapping → Mashup" Generation
  - Reuse: Mapping Polymorphism
- Conclusions and Future Directions



# Orchid: “Mapping ↔ ETL” Conversion

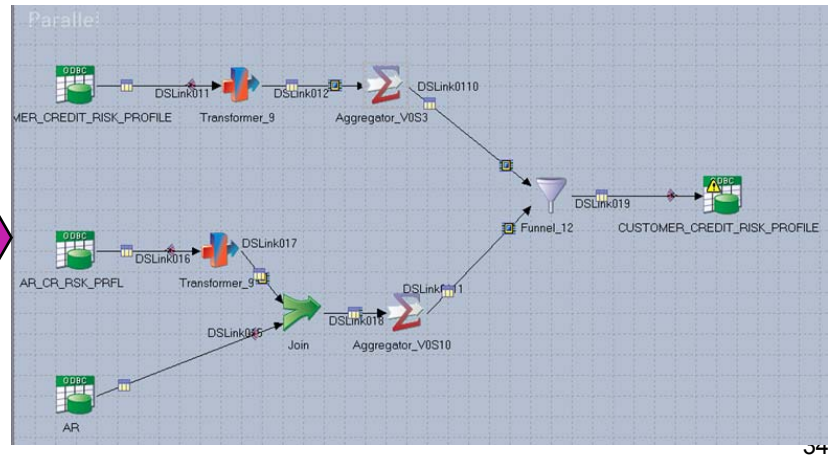
- Round-tripping between ETL (Extract, Transform, Load) scripts and declarative mappings
  - Generate ETL script from mappings
  - Extract mapping information from ETL scripts
  - Track and propagate mapping-related modifications
- Optimization of ETL scripts
  - Remove redundant operations
  - Push computation to other runtimes
  - “Rewrite” ETL scripts

## EII Mapping in RDA

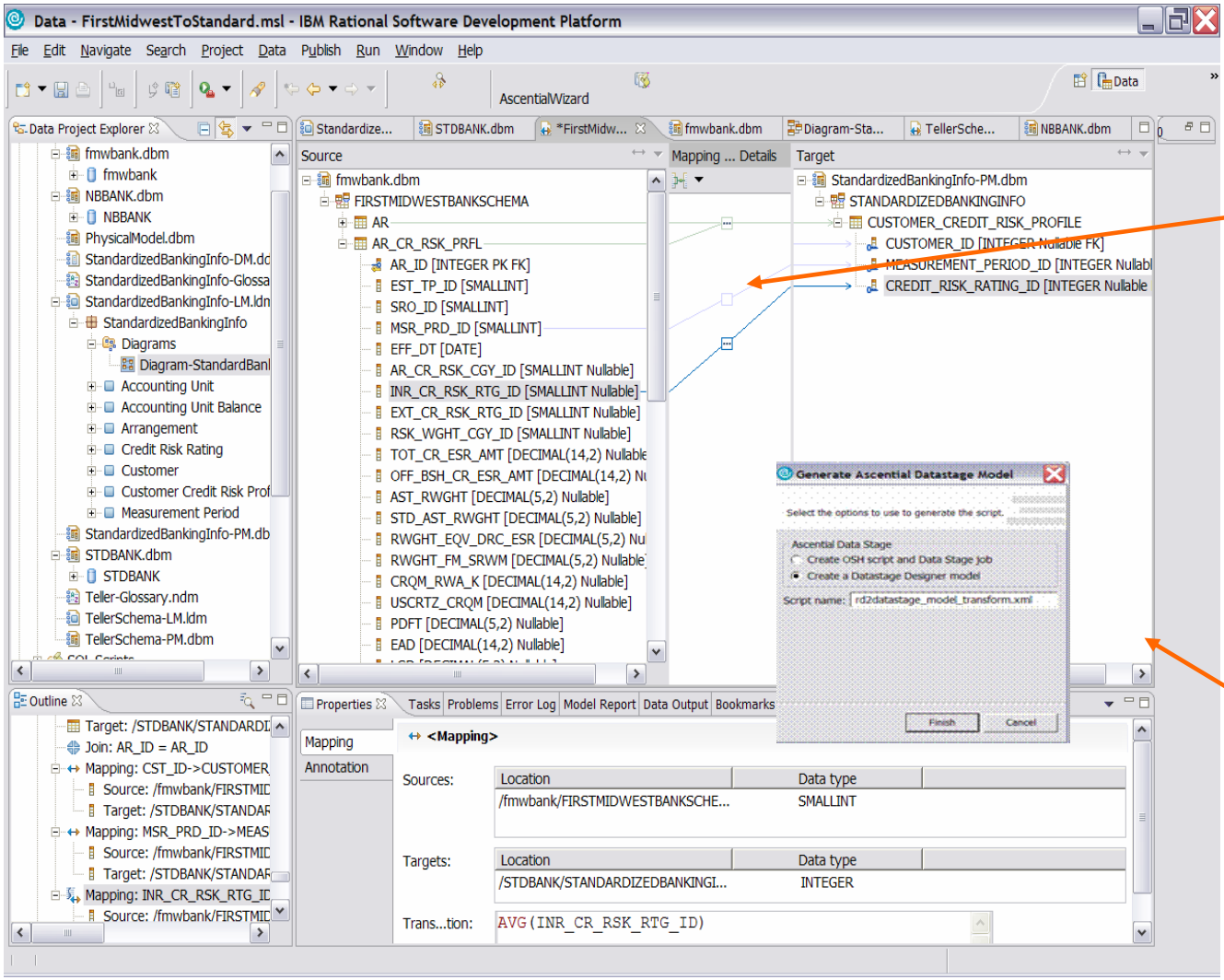



Orchid

## ETL Flow



# Mapping → ETL Generation




The developer inspects the connections between the sources, rules, and data warehouse model…

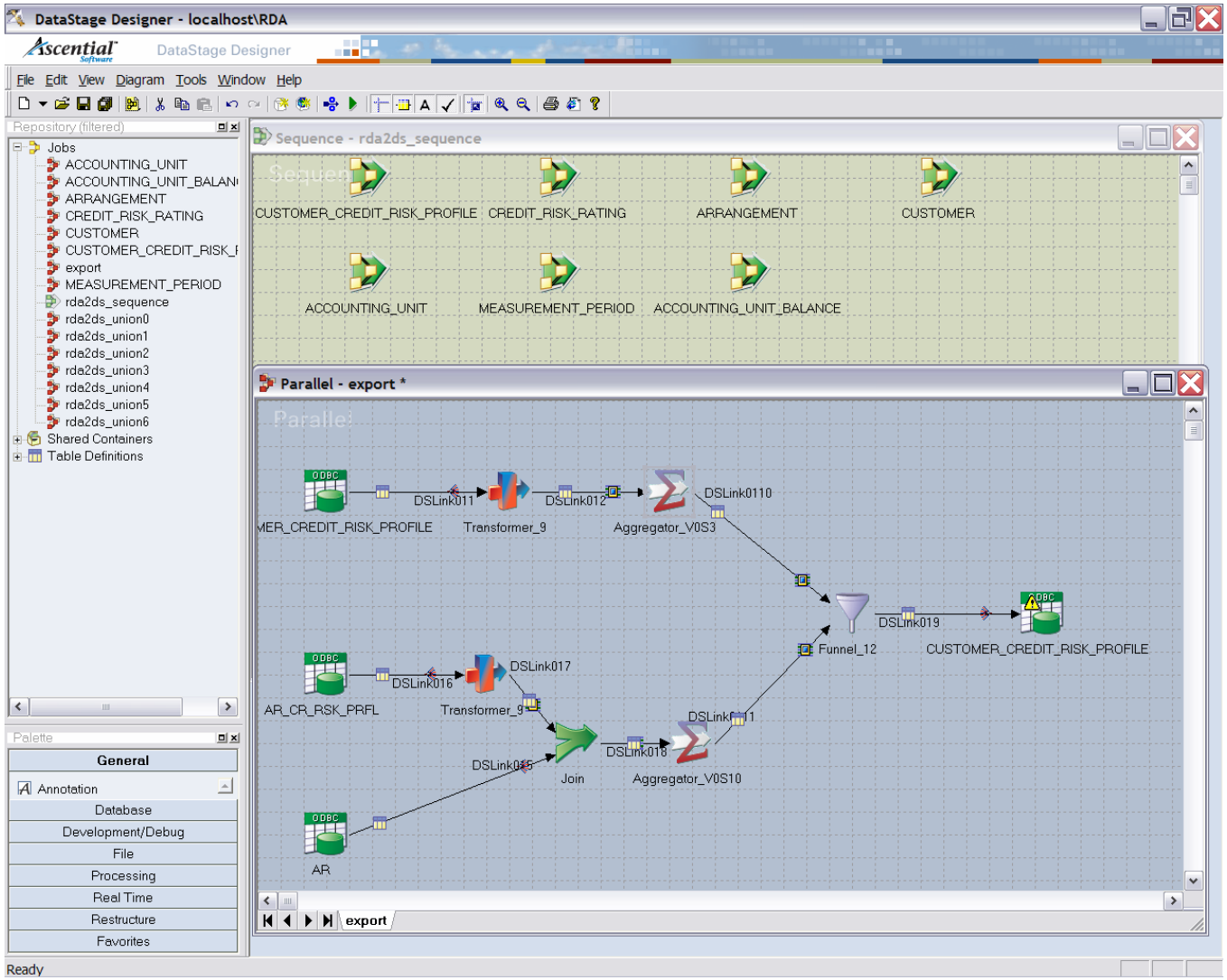
*…and generates a DataStage job to move data*

# ETL → Mapping Abstraction

*The developer refines, tests and deploys the DataStage job to production.*

*Need ETL → Mapping:*

- *To reflect changes made in the ETL script back in the mapping.*
- *To extract mappings out of ETL scripts.*

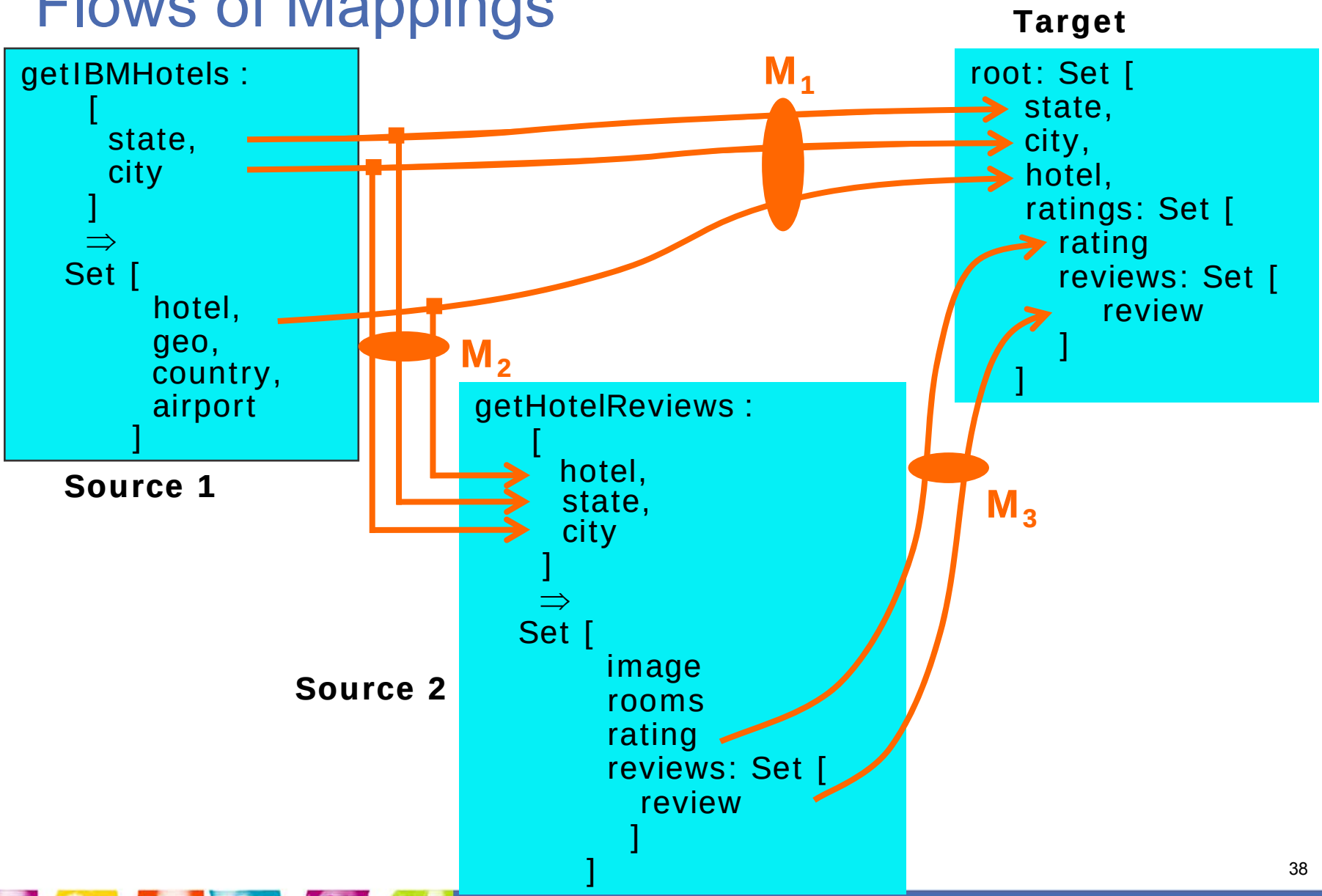


# Outline

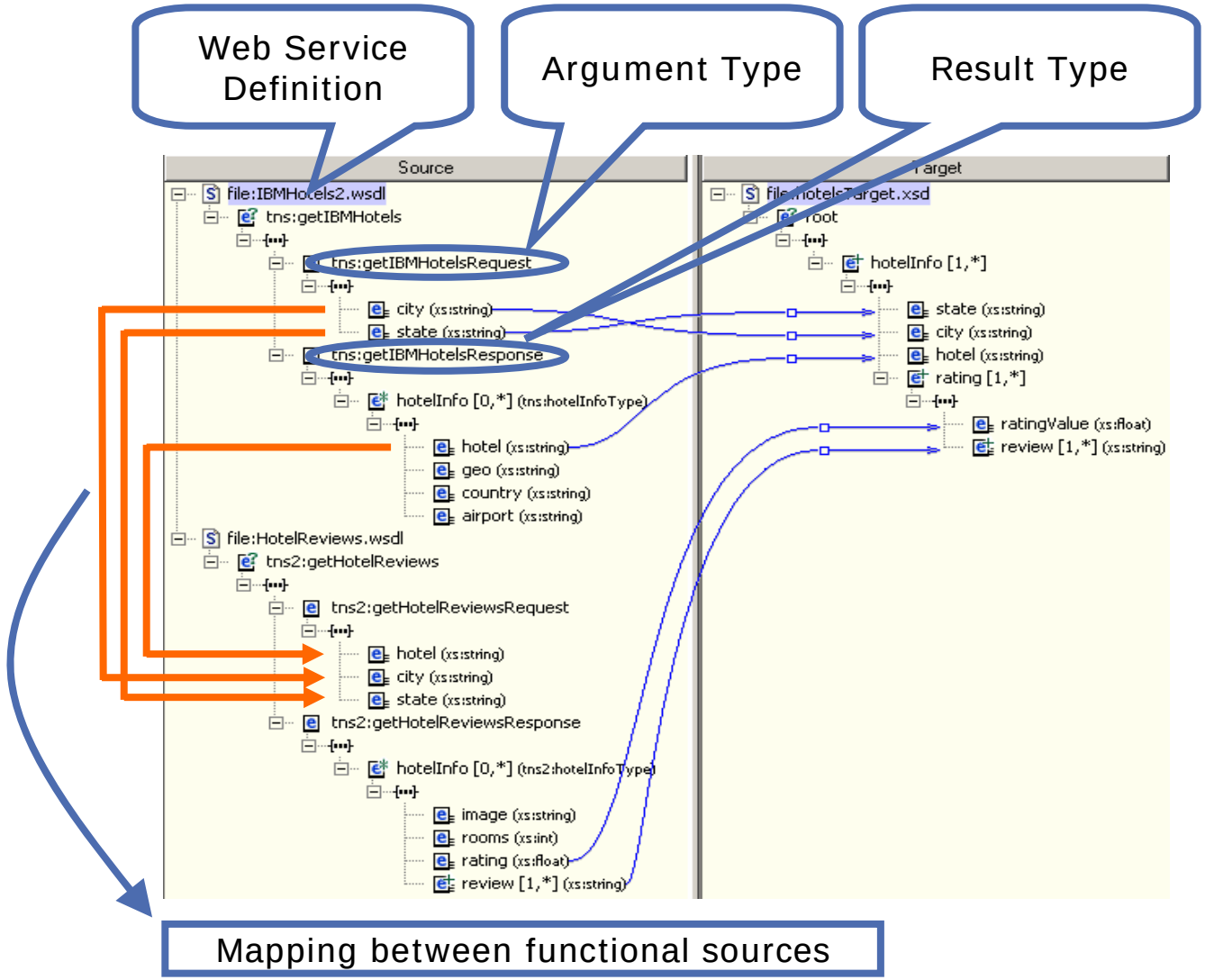
- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
- Clio2010: Mapping-Based Authoring of Data Flows
  - ETL: "Mapping ↔ ETL" Conversion
  - Web-Service Composition: Mappings for web-service data sources
    - [Alexe et al. 07]
  - Mashups: "Mapping → Mashup" Generation
  - Reuse: Mapping Polymorphism
- Conclusions and Future Directions



# Flows of Mappings



# Functional Data Sources



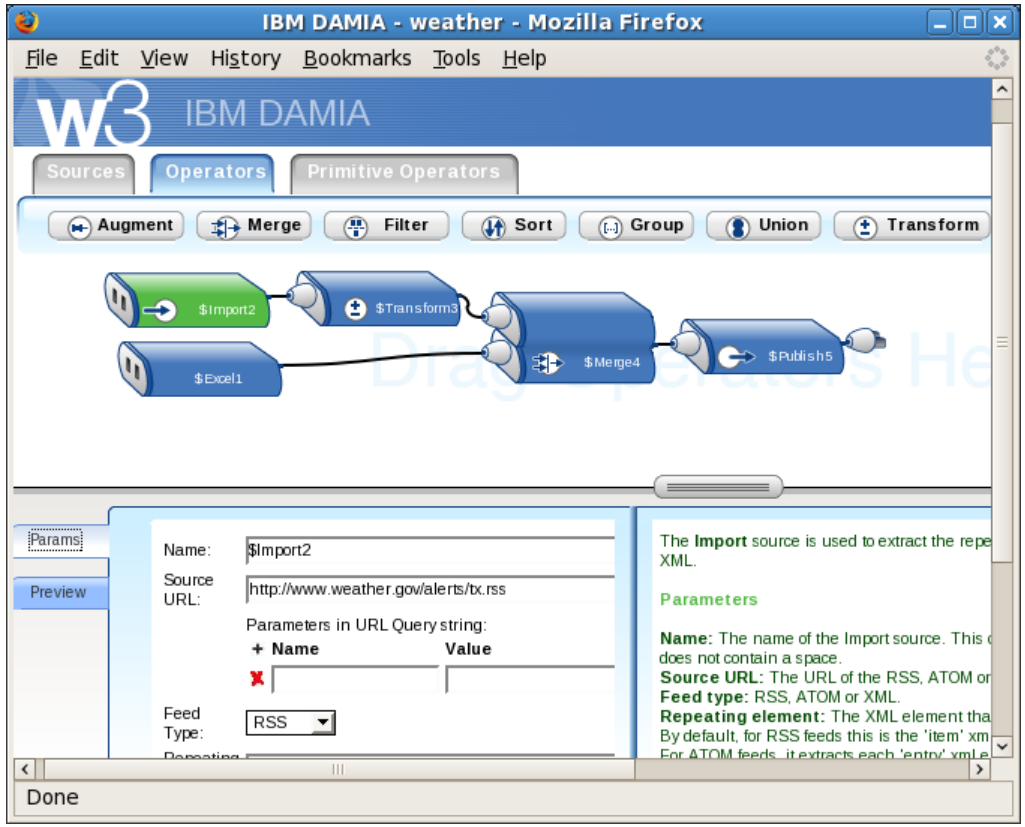
# Outline

- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
- Clio2010: Mapping-Based Authoring of Data Flows
  - ETL: "Mapping ↔ ETL" Conversion
  - Web-Service Composition: Mapping for web-service data sources
  - Mashups: "Mapping → Mashup" Generation
    - [Camacho et al. 07]
  - Reuse: Mapping Polymorphism
- Conclusions and Future Directions

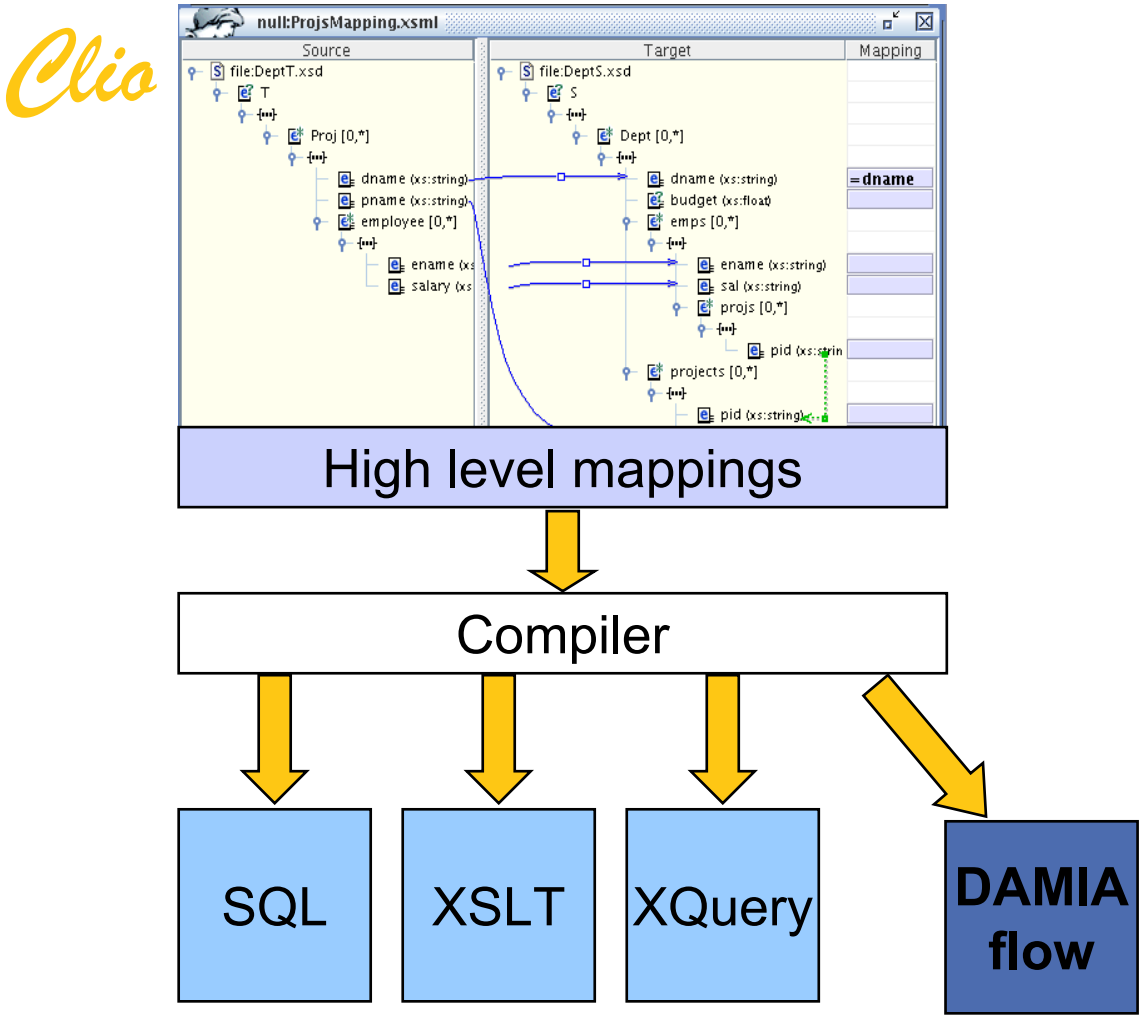


# IBM DAMIA: A Mashup Tool

- IBM DAMIA
  - Produces easily customizable flows
  - Mashup fabric for data aggregation and transformation
  - Web-based tool with user-friendly interface
  - No scripting/programming skills required
  - XQuery-like model



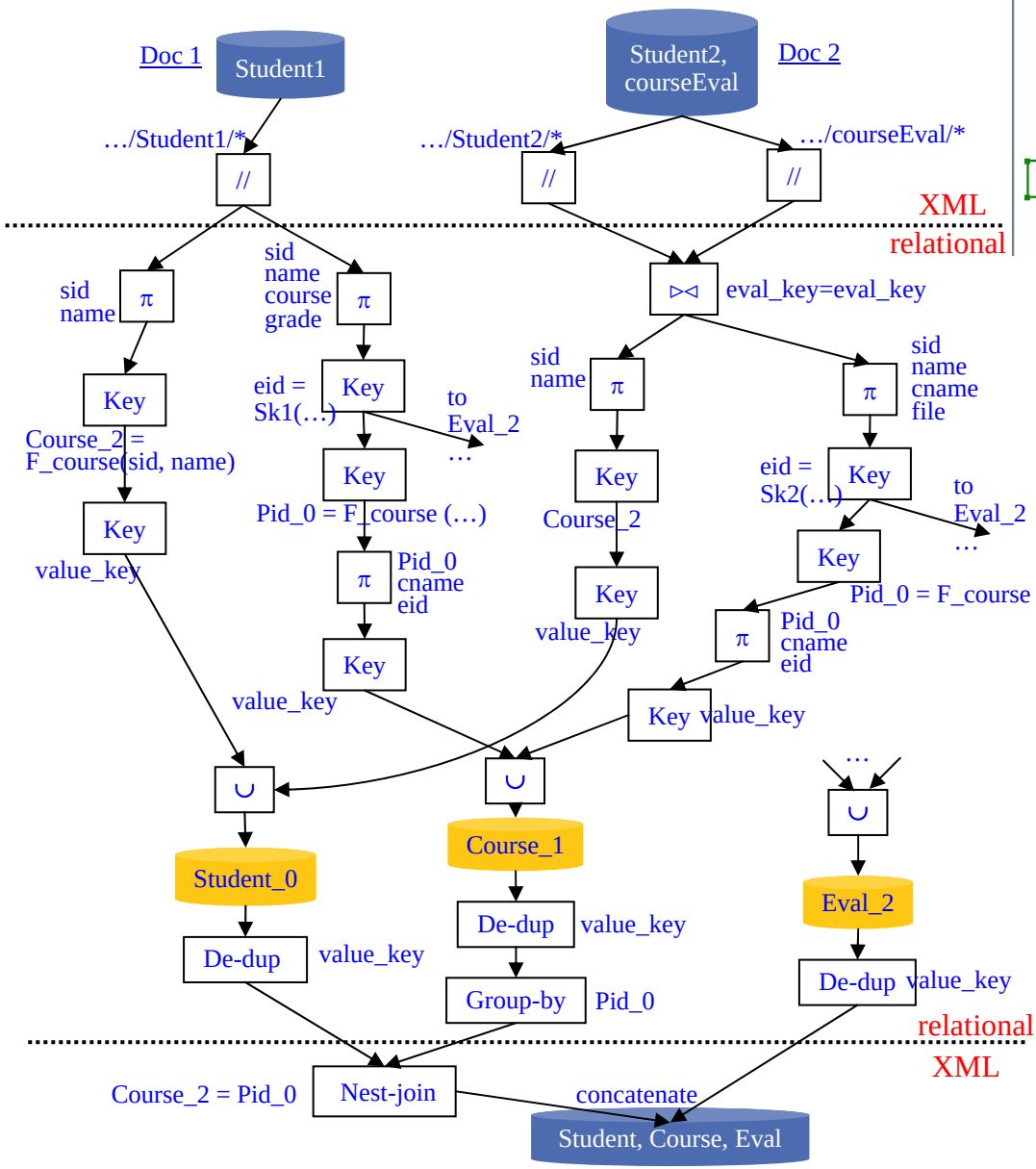
# Generate DAMIA Flows from Clio

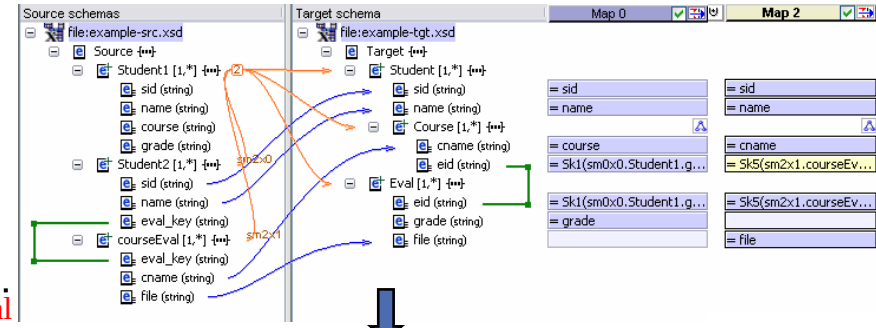


# From XQuery to ETL

- The XQuery has a specific flow pattern, with several phases:
  - Source extraction queries (projection, navigation, join)
    - Result: sets of flat tuples
  - Key generation
  - Union
  - Duplicate elimination
  - Computation of target groups
  - Generate target output (hierarchy)
    - Back to hierarchical data
- We can visualize all this as a graph of operators
  - we get the equivalent of an ETL job for XML






```
(: ======== Phase-1: shredding into relational views ======== :)
declare function local:shred($doc0 as element(root)) as element(root) {
<root>
 {
  for $sm0x0 in $doc0/Source/Student1
  return
    <Student_0>
      <sid_0>{ $sm0x0/sid/text() }</sid_0>
      <name_1>{ $sm0x0/name/text() }</name_1>
      <Course_2>{ fn:concat("F_course(", $sm0x0/sid/text(), ",", $sm0x0/name/text(), ")") }</Course_2>
      <value-key>{ fn:string-join(($sm0x0/sid/text(),$sm0x0/name/text(),fn:concat("F_course(", $sm0x0/sid/text(), ",",
$sm0x0/name/text(), ")")), "|") }</value-key>
    </Student_0>
 }
 {
  for $sm2x0 in $doc0/Source/Student2,
      $sm2x1 in $doc0/Source/courseEval
  where $sm2x0/eval_key = $sm2x1/eval_key
  return
    <Student_0>
      <sid_0>{ $sm2x0/sid/text() }</sid_0>
      <name_1>{ $sm2x0/name/text() }</name_1>
      <Course_2>{ fn:concat("F_course(", $sm2x0/sid/text(), ",", $sm2x0/name/text(), ")") }</Course_2>
      <value-key>{ fn:string-join(($sm2x0/sid/text(),$sm2x0/name/text(),fn:concat("F_course(", $sm2x0/sid/text(), ",",
$sm2x0/name/text(), ")")), "|") }</value-key>
    </Student_0>
 }
 {
  for $sm0x0 in $doc0/Source/Student1
  return
    <Course_1>
      <Pid_0>{ fn:concat("F_course(", $sm0x0/sid/text(), ",", $sm0x0/name/text(), ")") }</Pid_0>
      <cname_1>{ $sm0x0/course/text() }</cname_1>
      <eid_2>{ fn:concat("Sk0(", $sm0x0/grade/text(), ",", $sm0x0/course/text(), ",", $sm0x0/name/text(), ",", $sm0x0/sid/
text(), ")") }</eid_2>
      <value-key>{ fn:string-join((fn:concat("F_course(", $sm0x0/sid/text(), ",", $sm0x0/name/text(), ")"),$sm0x0/course/
text(),fn:concat("Sk0(", $sm0x0/grade/text(), ",", $sm0x0/course/text(), ",", $sm0x0/name/text(), ",", $sm0x0/sid/text(),
")")), "|") }</value-key>
    </Course_1>
 }
 ,
 . . .

(: ======== main query ======== :)
let $p0 := local:idgen(),
    $p1 := local:dupelim(local:shred($p0)),
    $p2 := local:nest($p1)
return $p2
```

# Outline

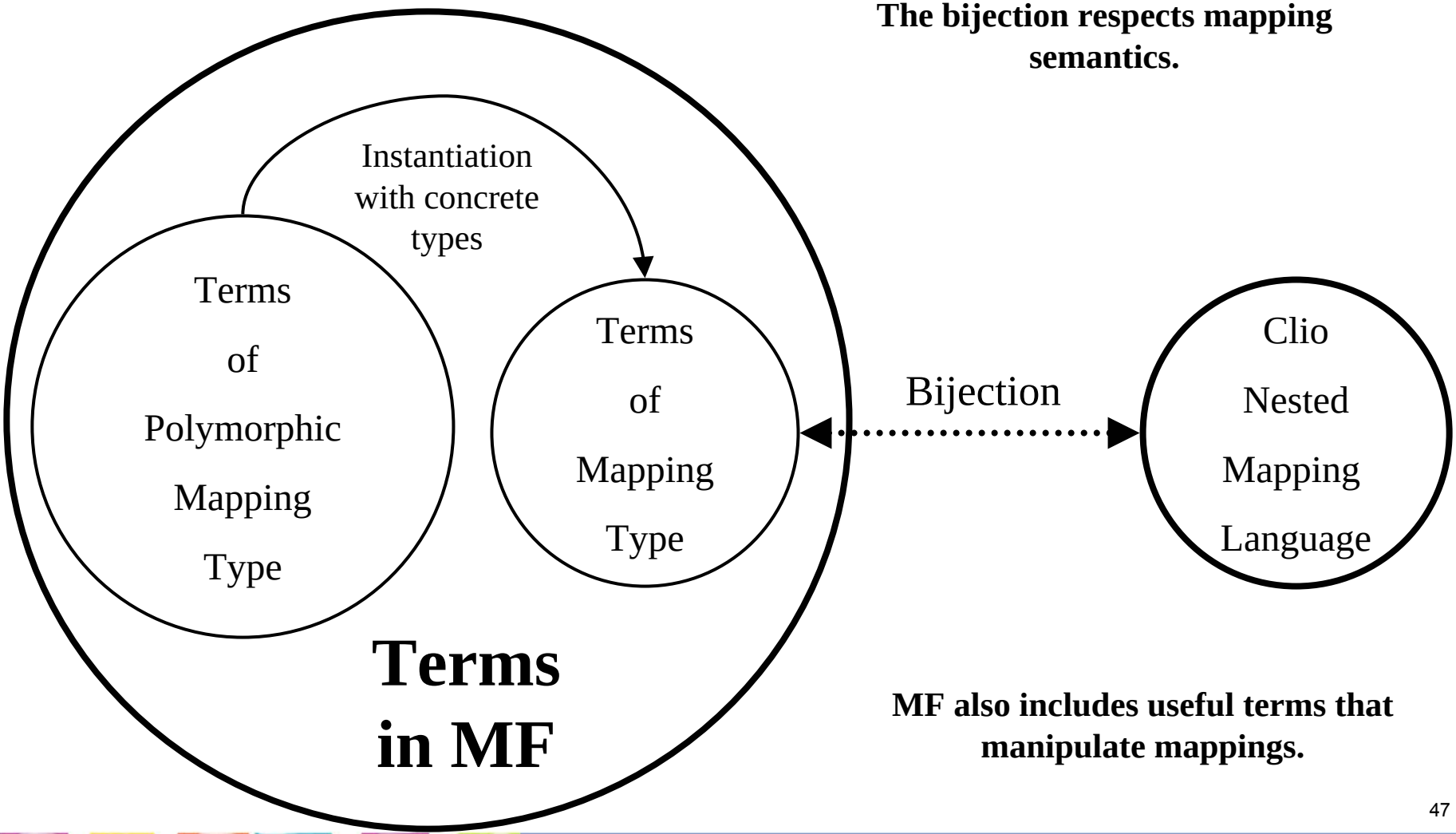
- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
- Clio2010: Mapping-Based Authoring of Data Flows
  - ETL: "Mapping ↔ ETL" Conversion
  - Web-Service Composition: Mapping for web-service data sources
  - Mashups: "Mapping → Mashup" Generation
  - Reuse: Mapping Polymorphism
    - [Wisnesky et al. 07]
- Conclusions and Future Directions



# Mapping Reuse

- Motivaion:
  - Existing mapping formalisms *implicitly* contain information that we need to be *explicit*
- Example:
  - A Clio nested mapping expression implicitly defines a class of schemas for which it has an interpretation
  - But in Clio mapping representation (XSML), every mapping instance requires concrete source and target schemas
    - The flexibility to re-use mappings is lost
- Approach:
  - We need a formal language for talking about mappings
  - Require theory and tools to manipulate mappings as blocks
- Solution:
  - Mapping Polymorphism

# Clio Language Embedding into MF




**The bijection respects mapping semantics.**

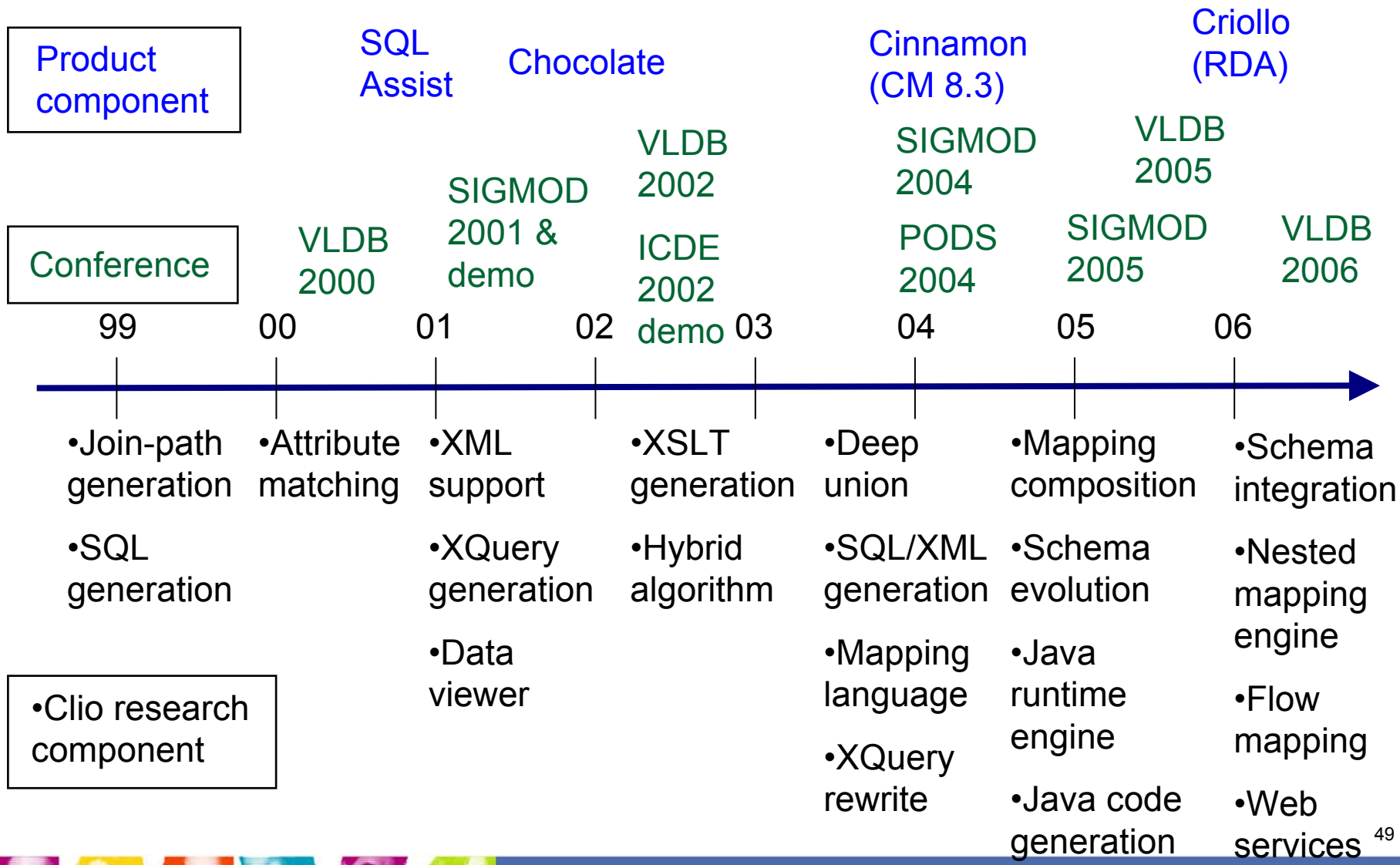
**MF also includes useful terms that manipulate mappings.**

# Outline

- Clio: Basic Features of a Schema Mapping System
- MAUI: Advanced Features of a Schema Mapping System
- Clio2010: Mapping-Based Authoring of Data Flows
- Conclusion and Related Work



# Clio Innovations Over Time



# Conclusion

- Mappings define relationships between schemas of data sources
- Mappings and transformation queries are the “new” metadata
- Customers want “I” (for integration), not EAI, EII, ETL, etc.
  - Consumability for information integration



# Related Work: Very Small Subset

- Information Integration Survey
  - [Haas ICDT 07] “Beauty and the Beast: The Theory and Practice of Information Integration”
  - [Kolatis PODS 05] “Schema mappings, data exchange, and metadata management”
  - [Halevy, Rajaraman, Ordille VLDB 06] “Data Integration: The Teenage Years” as part of their 10-year Best Paper Award on “Information Manifold” paper
- Schema Matching
  - Survey by Erhard Rahm and Philip Bernstein in VLDB J. ‘01
  - Much work by Philip Bernstein, Anhai Doan, Alon Halevy, Jayant Madhavan
  - Information Discovery project at IBM Almaden (Berthold Reinwald et al.)
- Data cleansing, conflict resolution, data quality, duplicate detection, entity resolution, data provenance
- Orchestra project at U Penn
- Schema Mapping Debugger
  - [Alexe, Chiticariu, Tan. VLDB 06]
- A Benchmark for Schema Mapping System
  - [Alexe, Tan, Velegrakis. 07]
- A new schema mapping GUI based on XQBE ideas
  - [Ceri, Hernandez, Raffio et al. 07]
- Deep Web, peer-to-peer, ontology, etc

# Acknowledgements: Clio Research Team


- The Project Founders:
  - Laura Haas and Renee Miller (around 1999)
- Regulars:
  - Clio Group: Mauricio Hernandez, Howard Ho, Lucian Popa, Ioana Stanoi
  - Theory Group: Ron Fagin, Phokion Kolaitis, Alan Nash
- Past Visiting scientists:
  - Stefan Dessloch, Paolo Papotti, Wang-Chiew Tan, Yannis Velegrakis, Cathy Wyss
- Past Interns and Post-docs:
  - Bogdan Alexe, Alfredo Camacho, Laura Chiticariu, Ariel Fuxman, Michael Gubanov, Haifeng Jiang, Felix Naumann, Paolo Papotti, Ahmed Radwan, Alessandro Raffio, Michael Richmond, Shivkumar Shivaji, Yannis Velegrakis, Melanie Weis, Ryan Wisnesky, Lingling Yan, Cong Yu, Jindan Zhou