MUSE: Mapping Understanding and Design by Example

Bogdan Alexe  UC Santa Cruz
Laura Chiticariu  UC Santa Cruz
Renée J. Miller  U. of Toronto
Wang-Chiew Tan  UC Santa Cruz

April 8, 2008
Schema Mappings

- One of the first steps in information integration is to specify the relationships (schema mappings) between schemas. This is known to be a difficult task.
One of the first steps in information integration is to specify the relationships (schema mappings) between schemas. This is known to be a difficult task.

**CompDB:**  
- **Companies:** Set of  
  - **Company:** Rcd  
    - cbranch  
    - cname  
    - location
  - **Projects:** Set of  
    - **Project:** Rcd  
      - pid  
      - pname  
      - cbranch  
      - manager
- **Employees:** Set of  
  - **Employee:** Rcd  
    - eid  
    - ename  
    - contact

**OrgDB:**  
- **Orgs:** Set of  
  - **Org:** Rcd  
    - oname
  - **Projects:** Set of  
    - **Project:** Rcd  
      - pname  
      - manager
- **Employees:** Set of  
  - **Employee:** Rcd  
    - eid  
    - ename
One of the first steps in information integration is to specify the relationships (schema mappings) between schemas. This is known to be a difficult task.

**CompDB:** $Rcd$
- **Companies:** Set of
  - **Company:** $Rcd$
    - cbranch
    - cname
    - location
- **Projects:** Set of
  - **Project:** $Rcd$
    - pid
    - pname
    - cbranch
    - manager
- **Employees:** Set of
  - **Employee:** $Rcd$
    - eid
    - ename
    - contact

**OrgDB:** $Rcd$
- **Orgs:** Set of
  - **Org:** $Rcd$
    - oname
  - **Projects:** Set of
    - **Project:** $Rcd$
      - pnamemanager
  - **Employees:** Set of
    - **Employee:** $Rcd$
      - eid
      - ename

Visual specification through value correspondences
Schema Mappings

- One of the first steps in information integration is to specify the relationships (schema mappings) between schemas. This is known to be a difficult task.

Visual spec.

Source schema S ——— Visual spec. ——— Target schema T
One of the first steps in information integration is to specify the relationships (schema mappings) between schemas. This is known to be a difficult task.

![Diagram]

Visual spec.  

Source schema S  

Target schema T  

Declarative specification  

Executable code (XSLT, XQuery, Java)

Mapping systems  

e.g.  
IBM Clio  
Altova MapForce  
Stylus Studio  
MS Biztalk Mapper
Schema Mappings

- One of the first steps in information integration is to specify the relationships (schema mappings) between schemas. This is known to be a difficult task.

Visual spec.  

Source schema S  

Target schema T  

Declarative specification

Executable code  
(XSLT, XQuery, Java)

Mapping systems  
e.g.  
IBM Clio  
Altova MapForce  
Stylus Studio  
MS Biztalk Mapper

Example:  
Data Exchange
One of the first steps in information integration is to specify the relationships (schema mappings) between schemas. This is known to be a difficult task.

Mapping systems:
- IBM Clio
- Altova MapForce
- Stylus Studio
- MS Biztalk Mapper

Example: Data Exchange
Designing Mappings

- Mapping systems can **automate only part of the mapping process**
  - Typically, **intricate manual work is needed** to perfect the specification
Designing Mappings

- Mapping systems can automate only part of the mapping process
  - Typically, intricate manual work is needed to perfect the specification

- The visual specification may be ambiguous. Mapping systems make default choices to resolve the ambiguities
  - These choices may not correspond to a designer’s intentions
  - The mapping designer might refine the specification manually
Real Life Example

In real life scenarios, mappings are extremely complicated
Real Life Example

In real life scenarios, mappings are extremely complicated

Map 2:
for sm2x0 in S0 dummy_COUNTRY_4
exists tm2x0 in S27 dummy_Country_10, tm2x1 in S27 dummy_organiza_13
where tm2x0.COUNTRY.organization.id = tm2x1.organization.id,
satisfy sm2x0.COUNTRY.AREA = tm2x0.country.area, sm2x0.COUNTRY.CAPITAL = tm2x0.country.capital,
sm2x0.COUNTRY.CODE = tm2x0.country.id, sm2x0.COUNTRY.NAME = tm2x0.country.name,
sm2x0.COUNTRY.POPULATION = tm2x0.country.population,

Map 3:
for sm3x0 in S0 dummy_GEO_RIVE_23, sm3x1 in S0 dummy_RIVER_24,
sm3x2 in S0 dummy_PROVINCE_5
where sm3x0.GEO_RIVER.RIVER = sm3x1.RIVER.NAME, sm3x2.PROVINCE.NAME = sm3x0.GEO_RIVER.PROVINCE,
sm3x2.PROVINCE.COUNTRY = sm2x0.COUNTRY.CODE,
exists tm3x0 in S27 dummy_river_24, tm3x1 in tm3x0.river dummy_located_23,
tm3x4 in S27 dummy_Country_10, tm3x5 in tm3x4.country dummy_province_9,
tm3x6 in S27 dummy_organiza_13
where tm3x4.country.organization.id = tm3x6.organization.id, tm3x5.province.id = tm3x1.located.province,
tm2x0.country.id = tm3x1.located.country,
satisfy sm2x0.COUNTRY.AREA = tm3x4.country.area, sm2x0.COUNTRY.CAPITAL = tm3x4.country.capital,
sm2x0.COUNTRY.CODE = tm3x4.country.id, sm2x0.COUNTRY.NAME = tm3x4.country.name,
sm2x0.COUNTRY.POPULATION = tm3x4.country.population, sm3x1.RIVER.LENGTH = tm3x6.river.length,
sm3x0.GEO_RIVER.RIVER = tm3x1.located.country, sm3x0.GEO_RIVER.PROVINCE = tm3x1.located.province,
sm3x1.RIVER.NAME = tm3x6.river.name,

Map 4:
for sm4x0 in S0 dummy_GEO_ISLA_25, sm4x1 in S0 dummy_ISLAND_26,
sm4x2 in S0 dummy_PROVINCE_5
where sm4x0.GEO ISLAND.ISLAND = sm4x1.ISLAND.NAME, sm4x2.PROVINCE.NAME = sm4x0.GEO ISLAND.PROVINCE,
sm4x2.PROVINCE.COUNTRY = sm2x0.COUNTRY.CODE,
exists tm4x0 in S27 dummy_island_26, tm4x1 in tm4x0.island dummy_located_25,
tm4x4 in S27 dummy_Country_10, tm4x5 in tm4x4.country dummy_province_9,
tm4x6 in S27 dummy_organiza_13
where tm4x4.country.organization.id = tm4x6.organization.id, tm4x5.province.id = tm4x1.located.province,
tm2x0.country.id = tm4x1.located.country,
satisfy sm2x0.COUNTRY.AREA = tm4x4.country.area, sm2x0.COUNTRY.CAPITAL = tm4x4.country.capital,
sm2x0.COUNTRY.CODE = tm4x4.country.id, sm2x0.COUNTRY.NAME = tm4x4.country.name,
sm2x0.COUNTRY.POPULATION = tm4x4.country.population, sm4x1.ISLAND.AREA = tm4x0.island.area,
sm4x1.ISLAND.COORDINATESLAT = tm4x0.island.latitude, sm4x0.GEO_ISLAND.COUNTRY = sm4x1.located.country,
satisfy sm4x0.GEO ISLAND.PROVINCE = tm4x1.located.province, sm4x1.ISLAND.COORDINATESLONG = tm4x0.island.longitude,
sm4x1.ISLAND.NAME = tm4x0.island.name,

Map 5:
for sm5x0 in S0 dummy_GEO_SEA_19, sm5x1 in S0 dummy_SEA_20,
sm5x2 in S0 dummy_PROVINCE_5
where sm5x2.PROVINCE.NAME = sm5x0.GEO SEA.PROVINCE, sm5x0.GEO SEA.SEA = sm5x1.SEA.NAME,
sm5x2.PROVINCE.COUNTRY = sm2x0.COUNTRY.CODE,
exists tm5x0 in S27 dummy_sea_19, tm5x1 in tm5x0.sea dummy_located_18,
tm5x4 in S27 dummy_Country_10, tm5x5 in tm5x4.country dummy_province_9,
tm5x6 in S27 dummy_organiza_13
where tm5x4.country.organization.id = tm5x6.organization.id, tm5x5.province.id = tm5x1.located.province,
tm2x0.country.id = tm5x1.located.country,
satisfy sm2x0.COUNTRY.AREA = tm5x4.country.area,
sm2x0.COUNTRY.CAPITAL = tm5x4.country.capital,
sm2x0.COUNTRY.CODE = tm5x4.country.id, sm2x0.COUNTRY.NAME = tm5x4.country.name,
sm2x0.COUNTRY.POPULATION = tm5x4.country.population, sm5x1.SEA.DEPTH = tm5x0.sea.depth,
sm5x0.GEO SEA.COUNTRY = tm5x1.located.country, sm5x0.GEO SEA.PROVINCE = tm5x1.located.province,
sm5x1.SEA.NAME = tm5x0.sea.name,
Designing Mappings

- Specifications are often impossible to understand through visual inspection
- Few tools are available to assist in understanding and designing alternative mappings
Designing Mappings

- Specifications are often **impossible to understand** through visual inspection
- **Few tools are available** to assist in understanding and designing alternative mappings
- **MUSE** is a tool designed towards this end
Designing Mappings

- Specifications are often **impossible to understand** through visual inspection
- Few tools are available to assist in understanding and designing alternative mappings
- **MUSE** is a tool designed towards this end
- In MUSE, we focus on declarative specifications

![Diagram](image.png)

**Advantages:**
- easier to reason about
- reusable for various tasks

**Declarative specification**

**Executable code**
(XSLT, XQuery, Java)
Our vision

- **MUSE is a mapping design wizard** that uses (real) data examples to help designers understand, design and refine schema mappings

- **MUSE leverages familiar data examples** to help understand mappings
  - real data examples are used whenever possible
  - otherwise, synthetic examples are constructed
Our vision

- MUSE is a mapping design wizard that uses (real) data examples to help designers understand, design and refine schema mappings.

- MUSE leverages familiar data examples to help understand mappings:
  - real data examples are used whenever possible
  - otherwise, synthetic examples are constructed

- Currently, MUSE has two features:
  - Muse-G: design grouping semantics
  - Muse-D: disambiguate alternative mappings
MUSE Workflow
MUSE Workflow

Real Source Instance (if available) → MUSE → Mapping Specification
MUSE Workflow

Real Source Instance (if available)

Mapping Specification

MUSE

Generation

Real/Synthetic Data Examples
MUSE Workflow

- Real Source Instance (if available)
- MUSE
- Real/Synthetic Data Examples
- Mapping Specification
- Examination
- Mapping designer inspects data examples
MUSE Workflow

Mapping Specification

Essentially Yes/No Answers

Real Source Instance (if available)

Generation

Real/Synthetic Data Examples

Examination

Mapping designer inspects data examples
MUSE Workflow

Real Source Instance (if available)

Mapping Specification

Refinement
Grouping Semantics
Disambiguation

Essentially Yes/No Answers

Real/Synthetic Data Examples

Generation

MUSE

Examination

Mapping designer inspects data examples

Essentially Yes/No Answers
MUSE Workflow

MUSE

Real Source Instance (if available)

Real/Synthetic Data Examples

Mapping designer inspects data examples

Examination

Generation

Essentially Yes/No Answers

Refinement

Grouping Semantics

Disambiguation
MUSE Workflow

- Real Source Instance (if available)
- Mapping Specification
- Real/Synthetic Data Examples
- Examination
- Essentially Yes/No Answers
- Mapping designer inspects data examples
- Grouping Semantics
- Disambiguation

Generation

Refinement
MUSE Workflow

Mapping Specification

Real Source Instance (if available)

Real/Synthetic Data Examples

Essentially Yes/No Answers

Metal/Metals

Refinement

Grouping Semantics

Disambiguation

Generation

Mapping designer inspects data examples

Examination
MUSE Workflow

Real Source Instance (if available)

Mapping Specification

Refinement
Grouping Semantics
Disambiguation

MUSE

Essentially Yes/No Answers

Generation

Real/Synthetic Data Examples

Examination

Mapping designer inspects data examples
MUSE Workflow

Real Source Instance (if available)

Real/Synthetic Data Examples

MAPPING SPECIFICATION

Refinement

Grouping Semantics

Disambiguation

Essentially Yes/No Answers

Essentially

Mapping designer inspects data examples

Examination

Generation
MUSE Workflow

Mapping Specification

Real Source Instance (if available)

Real/Synthetic Data Examples

Generation

Examination

Essentially Yes/No Answers

Grouping Semantics

Disambiguation

Refinement

Mapping designer inspects data examples
MUSE Workflow

1. Real Source Instance (if available)
2. Real/Synthetic Data Examples
3. Mapping designer inspects data examples
4. Examination
5. Essentially Yes/No Answers
6. Grouping Semantics
7. Disambiguation
8. Refinement
10. Generation

Diagram:
- MUSE
- Real Source Instance
- Real/Synthetic Data Examples
- Mapping designer
- Examination
- Essentially Yes/No Answers
- Grouping Semantics
- Disambiguation
- Refinement
- Mapping Specification
**Example**

**CompDB:** $Rcd$

**Companies:** $Set \ of$

**Company:** $Rcd$

- cbranch
- cname
- location

**Projects:** $Set \ of$

**Project:** $Rcd$

- pid
- pname
- cbranch
- manager

**Employees:** $Set \ of$

**Employee:** $Rcd$

- eid
- ename
- contact
Example

CompDB: $Rcd$

Companies: $Set$ of
  Company: $Rcd$
    cbranch
    cname
    location
  Projects: $Set$ of
    Project: $Rcd$
      pid
      pname
      cbranch
      manager

Employees: $Set$ of
  Employee: $Rcd$
    eid
    ename
    contact

OrgDB: $Rcd$

Orgs: $Set$ of
  Org: $Rcd$
    oname
    Projects: $Set$ of
      Project: $Rcd$
        pname
        manager

Employees: $Set$ of
  Employee: $Rcd$
    eid
    ename
**Example**

**CompDB:** $Rcd$

- **Companies:** $Set$ of
  - **Company:** $Rcd$
    - cbranch
    - cname
    - location

- **Projects:** $Set$ of
  - **Project:** $Rcd$
    - pid
    - pname
    - cbranch
    - manager

**OrgDB:** $Rcd$

- **Orgs:** $Set$ of
  - **Org:** $Rcd$
    - oname

- **Projects:** $Set$ of
  - **Project:** $Rcd$
    - pname
    - manager

- **Employees:** $Set$ of
  - **Employee:** $Rcd$
    - eid
    - ename
    - contact
Example

CompDB: $Rcd$
- Companies: $Set of$
  - Company: $Rcd$
    - cbranch
    - cname
    - location
- Projects: $Set of$
  - Project: $Rcd$
    - pid
    - pname
    - cbranch
    - manager
- Employees: $Set of$
  - Employee: $Rcd$
    - eid
    - ename
    - contact

OrgDB: $Rcd$
- Orgs: $Set of$
  - Org: $Rcd$
    - oname
- Projects: $Set of$
  - Project: $Rcd$
    - pname
    - manager
- Employees: $Set of$
  - Employee: $Rcd$
    - eid
    - ename

Declarative Mapping
for
  c in CompDB.Companies
  p in CompDB.Projects
  e in CompDB.Employees
Example

CompDB: Rcd
Companies: Set of
  Company: Rcd
cbranch
cname
location
Projects: Set of
  Project: Rcd
  pid
  pname
  cbranch
  manager
Employees: Set of
  Employee: Rcd
eid
ename
contact

OrgDB: Rcd
Orgs: Set of
  Org: Rcd
  oname
Projects: Set of
  Project: Rcd
  pname
  manager
Employees: Set of
  Employee: Rcd
  eid
  ename

Declarative Mapping
for
c in CompDB.Companies
p in CompDB.Projects
e in CompDB.Employees
satisfy
  p.cbranch = c.cbranch
  e.eid = p.manager
Example

CompDB: \( Rcd \)
- **Companies**: \( Set \ of \ )
  - **Company**: \( Rcd \)
    - cbranch
    - cname
    - location

- **Projects**: \( Set \ of \ )
  - **Project**: \( Rcd \)
    - pid
    - pname
    - cbranch
    - manager

OrgDB: \( Rcd \)
- **Orgs**: \( Set \ of \ )
  - **Org**: \( Rcd \)
    - oname

- **Projects**: \( Set \ of \ )
  - **Project**: \( Rcd \)
    - pname
    - manager

- **Employees**: \( Set \ of \ )
  - **Employee**: \( Rcd \)
    - eid
    - ename

Declarative Mapping

for
c in CompDB.Companies
p in CompDB.Projects
e in CompDB.Employees

satisfy
p.cbranch = c.cbranch
e.eid = p.manager

exists
o in OrgDB.Orgs
p_1 in o.Projects
e_1 in OrgDB.Employees
Example

CompDB: Rcd
Companies: Set of
  Company: Rcd
cbranch
cname
location
Projects: Set of
  Project: Rcd
  pid
  pname
  cbranch
  manager
Employees: Set of
  Employee: Rcd
eid
ename
contact

OrgDB: Rcd
Orgs: Set of
  Org: Rcd
  oname
Projects: Set of
  Project: Rcd
  pname
  manager
Employees: Set of
  Employee: Rcd
  eid
  ename

Declarative Mapping

for
c in CompDB.Companies
p in CompDB.Projects
e in CompDB.Employees

satisfy
  p.cbranch = c.cbranch
  e.eid = p.manager

exists
  o in OrgDB.Orgs
  p1 in o.Projects
  e1 in OrgDB.Employees

satisfy
  p1.manager = e1.eid
Example

CompDB: $Rcd$
- Companies: $Set$ of $Company: Rcd$
  - cbranch
  - cname
  - location
- Projects: $Set$ of $Project: Rcd$
  - pid
  - pname
  - cbranch
  - manager
- Employees: $Set$ of $Employee: Rcd$
  - eid
  - ename
  - contact

OrgDB: $Rcd$
- Orgs: $Set$ of $Org: Rcd$
  - oname
- Projects: $Set$ of $Project: Rcd$
  - pname
  - manager
- Employees: $Set$ of $Employee: Rcd$
  - eid
  - ename

Declarative Mapping

for $c$ in CompDB.Companies
  $p$ in CompDB.Projects
  $e$ in CompDB.Employees

satisfy
  $p.cbranch = c.cbranch$
  $e.eid = p.manager$

exists
  $o$ in OrgDB.Orgs
  $p_1$ in $o.Projects$
  $e_1$ in OrgDB.Employees

satisfy
  $p_1.manager = e_1.eid$

where
  $c.cname = o.oname$
  $e.eid = e_1.eid$
  $e.ename = e_1.ename$
  $p.pname = p_1.pname$
Example

CompDB: Rcd

Companies: Set of
  Company: Rcd
    cbranch
cname
location

Projects: Set of
  Project: Rcd
    pid
pname
manager
cbranch

OrgDB: Rcd

Orgs: Set of
  Org: Rcd
    oname

Projects: Set of
  Project: Rcd
    pname
manager

Employees: Set of
  Employee: Rcd
    eid
ename

Grouping Projects:

Example source:

Companies
Redmond Microsoft USA
S. Valley Microsoft USA

Projects
P1 DB Redmond e4
P2 Web S. Valley e5
**Example**

**CompDB:** \( Rcd \)
- **Companies:** Set of
  - **Company:** \( Rcd \)
    - cbranch
    - cname
    - location

- **Projects:** Set of
  - **Project:** \( Rcd \)
    - pid
    - pname
    - manager

- **Employees:** Set of
  - **Employee:** \( Rcd \)
    - eid
    - ename
    - contact

- **Orgs:** Set of
  - **Org:** \( Rcd \)
    - oname

**OrgDB:** \( Rcd \)

- **Orgs:** Set of
  - **Org:** \( Rcd \)
    - oname

- **Projects:** Set of
  - **Project:** \( Rcd \)
    - pname
    - manager

- **Employees:** Set of
  - **Employee:** \( Rcd \)
    - eid
    - ename

**Grouping Projects:**

**Example source:**

**Companies**
- Redmond Microsoft USA
- S. Valley Microsoft USA

**Projects**
- P1 DB Redmond e4
- P2 Web S. Valley e5

**Orgs**
- Microsoft

**Projects:**
- DB e4
- Web e5

**Group by cbranch**

**Orgs**
- Microsoft

**Projects:**
- Web e5
Example

CompDB: \( Rcd \)
- Companies: \( Set \ of \) Company: \( Rcd \)
  - cbranch
  - cname
  - location
- Projects: \( Set \ of \) Project: \( Rcd \)
  - pid
  - pname
  - manager
- Employees: \( Set \ of \) Employee: \( Rcd \)
  - eid
  - ename
- OrgDB: \( Rcd \)
  - Orgs: \( Set \ of \) Org: \( Rcd \)
    - oname
  - Projects: \( Set \ of \) Project: \( Rcd \)
    - pmanager
  - Employees: \( Set \ of \) Employee: \( Rcd \)
    - eid
    - ename

Declarative Mapping
for
c in CompDB.Companies
p in CompDB.Projects
e in CompDB.Employees
satisfy
  p.cbranch = c.cbranch
  e.eid = p.manager
exists
  o in OrgDB.Orgs
  p1 in o.Projects
  e1 in OrgDB.Employees
satisfy
  p1.manager = e1.eid
where
  c.cname = o.oname
  e.eid = e1.eid
  e.ename = e1.ename
  p.pname = p1.pname

o.Projects = SKProjects(c.cbranch, c.cname, c.location)
Example

CompDB: \( Rcd \)
- Companies: Set of
- Company: \( Rcd \)
  - cbranch
  - cname
  - location

Projects: Set of
- Project: \( Rcd \)
  - pid
  - pname
  - manager

Employees: Set of
- Employee: \( Rcd \)
  - eid
  - ename
  - contact

OrgDB: \( Rcd \)
- Orgs: Set of
- Org: \( Rcd \)
  - oname

Projects: Set of
- Project: \( Rcd \)
  - pid
  - pname
  - manager

Employees: Set of
- Employee: \( Rcd \)
  - eid
  - ename

Declarative Mapping

for
c in CompDB.Companies
p in CompDB.Projects
e in CompDB.Employees
satisfy
p.cbranch = c.cbranch
e.eid = p.manager
exists
o in OrgDB.Orgs
p₁ in o.Projects
e₁ in OrgDB.Employees
satisfy
p₁.manager = e₁.eid
where
c.cname = o.oname
e.eid = e₁.eid
e.ename = e₁.ename
p.pname = p₁.pname

Grouping Function

\( o.\text{Projects} = \text{SKProjects}(c.cbranch, c.cname, c.location) \)

Group by what subset of \{cbranch, cname, location\}?
Muse-G: Grouping Semantics Design

- **Goal:** infer a grouping function that has the *same effect* as the one intended by the designer
Muse-G: Grouping Semantics Design

- **Goal:** infer a grouping function that has the same effect as the one intended by the designer
- Muse-G probes each possible grouping attribute: start with cbranch
Muse-G: Grouping Semantics Design

- **Goal:** infer a grouping function that has the *same effect* as the one intended by the designer
- Muse-G probes each possible grouping attribute: start with `cbranch`

**Example source**

### Companies
- Redmond  Microsoft  USA
- S. Valley  Microsoft  USA

### Projects
- P1  DB  Redmond  e4
- P2  Web  S. Valley  e5

### Employees
- e4  John  x234
- e5  Anna  x888
**Muse-G: Grouping Semantics Design**

- **Goal:** infer a grouping function that has the **same effect** as the one intended by the designer
- Muse-G probes each possible grouping attribute: start with `cbranch`

**Example source**

<table>
<thead>
<tr>
<th>Companies</th>
<th>Projects</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redmond Microsoft</td>
<td>P1</td>
<td>e4 John x234</td>
</tr>
<tr>
<td>S. Valley Microsoft</td>
<td>P2</td>
<td>e5 Anna x888</td>
</tr>
</tbody>
</table>

**Target Scenario 1**

- **group by cbranch**
  - Orgs: Microsoft
  - Projects: DB e4
  - Projects: Web e5
  - Employees: e4 John
  - Employees: e5 Anna

**Target Scenario 2**

- **do not group by cbranch**
  - Orgs: Microsoft
  - Projects: DB e4
  - Projects: Web e5
  - Employees: e4 John
  - Employees: e5 Anna
Muse-G: Grouping Semantics Design

- **Goal:** infer a grouping function that has the same effect as the one intended by the designer
- Muse-G probes each possible grouping attribute: start with `cbranch`

---

**Example source**

- **Companies**
  - Redmond: Microsoft, USA
  - S. Valley: Microsoft, USA

- **Projects**
  - P1: DB, Redmond, e4
  - P2: Web, S. Valley, e5

- **Employees**
  - e4: John, x234
  - e5: Anna, x888

**Target Scenario 1**

- **Orgs**
  - Microsoft
- **Projects:**
  - DB: e4

**Target Scenario 2**

- **Orgs**
  - Microsoft
- **Projects:**
  - DB: e4
  - Web: e5

---

**Target Scenario 1**

- **Employees**
  - e4: John
  - e5: Anna

**Target Scenario 2**

- **Employees**
  - e4: John
  - e5: Anna
**Muse-G: Grouping Semantics Design**

- **Goal:** infer a grouping function that has the same effect as the one intended by the designer
- **Muse-G** probes each possible grouping attribute: start with `cbranch`

**Example source**

<table>
<thead>
<tr>
<th>Companies</th>
<th>Microsoft USA</th>
<th>S. Valley Microsoft USA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1 DB Redmond e4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2 Web S. Valley e5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Employees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e4 John x234</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e5 Anna x888</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Target Scenario 1**

- **group by cbranch**

<table>
<thead>
<tr>
<th>Orgs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td></td>
</tr>
<tr>
<td>Projects:</td>
<td></td>
</tr>
<tr>
<td>DB e4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Web e5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4 John</td>
<td></td>
</tr>
<tr>
<td>e5 Anna</td>
<td></td>
</tr>
</tbody>
</table>

**Target Scenario 2**

- **do not group by cbranch**

<table>
<thead>
<tr>
<th>Orgs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td></td>
</tr>
<tr>
<td>Projects:</td>
<td></td>
</tr>
<tr>
<td>DB e4</td>
<td></td>
</tr>
<tr>
<td>Web e5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4 John</td>
<td></td>
</tr>
<tr>
<td>e5 Anna</td>
<td></td>
</tr>
</tbody>
</table>
Muse-G: Grouping Semantics Design

- **Goal:** infer a grouping function that has the *same effect* as the one intended by the designer
- Muse-G probes each possible grouping attribute: start with `cbranch`

### Example source

<table>
<thead>
<tr>
<th>Companies</th>
<th>Redmond</th>
<th>Microsoft</th>
<th>USA</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Valley</td>
<td>Microsoft</td>
<td>USA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Projects</th>
<th>Projects: SK(Redmond,y)</th>
<th>Projects: SK(S. Valley,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>DB</td>
<td>Web</td>
</tr>
<tr>
<td>P2</td>
<td>Redmond</td>
<td>S. Valley</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th>Projects: SK(y)</th>
<th>y ⊆ { Microsoft, USA }</th>
</tr>
</thead>
<tbody>
<tr>
<td>e4 John</td>
<td>DB</td>
<td>e4</td>
</tr>
<tr>
<td>e5 Anna</td>
<td>Web</td>
<td>e5</td>
</tr>
</tbody>
</table>

Target Scenario 1

**group** by `cbranch`

- **Orgs:** Microsoft
- **Projects:** SK(Redmond,y), SK(S. Valley,y)
- **Employees:** e4 John, e5 Anna

Target Scenario 2

**do not group** by `cbranch`

- **Orgs:** Microsoft
- **Projects:** SK(y)
- **Employees:** e4 John, e5 Anna
The next probed attribute is **cname**

**Example source**

<table>
<thead>
<tr>
<th>Companies</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Valley</td>
<td>Microsoft</td>
<td>USA</td>
<td></td>
</tr>
<tr>
<td>Mt. View</td>
<td>Google</td>
<td>USA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Projects</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>DB</td>
<td>S. Valley</td>
<td>e4</td>
</tr>
<tr>
<td>P4</td>
<td>Web</td>
<td>Mt. View</td>
<td>e6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4</td>
<td>John</td>
<td>x234</td>
</tr>
<tr>
<td>e6</td>
<td>Kat</td>
<td>x331</td>
</tr>
</tbody>
</table>
## Muse-G: Second Question

The next probed attribute is **cname**

### Example source

<table>
<thead>
<tr>
<th>Companies</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Valley</td>
<td>Microsoft</td>
<td>USA</td>
</tr>
<tr>
<td>Mt. View</td>
<td>Google</td>
<td>USA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Projects</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>DB</td>
<td>S. Valley</td>
</tr>
<tr>
<td>P4</td>
<td>Web</td>
<td>Mt. View</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4</td>
<td>John</td>
<td>x234</td>
</tr>
<tr>
<td>e6</td>
<td>Kat</td>
<td>x331</td>
</tr>
</tbody>
</table>

### Target Scenario 1

**Orgs**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Projects:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>e4</td>
</tr>
<tr>
<td>Google</td>
<td></td>
</tr>
</tbody>
</table>

**Projects:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>e6</td>
</tr>
</tbody>
</table>

**Employees**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4</td>
<td>John</td>
</tr>
<tr>
<td>e6</td>
<td>Kat</td>
</tr>
</tbody>
</table>

### Target Scenario 2

**Orgs**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Projects:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>e4</td>
</tr>
<tr>
<td>Google</td>
<td></td>
</tr>
</tbody>
</table>

**Projects:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>e6</td>
</tr>
</tbody>
</table>

**Employees**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4</td>
<td>John</td>
</tr>
<tr>
<td>e6</td>
<td>Kat</td>
</tr>
</tbody>
</table>
### Muse-G: Second Question

- The next probed attribute is **cname**

#### Example source

<table>
<thead>
<tr>
<th>Companies</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Valley</td>
<td>Microsoft</td>
<td>USA</td>
</tr>
<tr>
<td>Mt. View</td>
<td>Google</td>
<td>USA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Projects</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>DB</td>
<td>S. Valley</td>
</tr>
<tr>
<td>P4</td>
<td>Web</td>
<td>Mt. View</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4</td>
<td>John</td>
<td>x234</td>
</tr>
<tr>
<td>e6</td>
<td>Kat</td>
<td>x331</td>
</tr>
</tbody>
</table>

#### Target Scenario 1

**group by cname**

<table>
<thead>
<tr>
<th>Orgs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projects:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB</td>
<td>e4</td>
<td></td>
</tr>
<tr>
<td>Google</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projects:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web</td>
<td>e6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4</td>
<td>John</td>
<td></td>
</tr>
<tr>
<td>e6</td>
<td>Kat</td>
<td></td>
</tr>
</tbody>
</table>

#### Target Scenario 2

**do not group by cname**

<table>
<thead>
<tr>
<th>Orgs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projects:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB</td>
<td>e4</td>
<td></td>
</tr>
<tr>
<td>Google</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projects:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web</td>
<td>e6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e4</td>
<td>John</td>
<td></td>
</tr>
<tr>
<td>e6</td>
<td>Kat</td>
<td></td>
</tr>
</tbody>
</table>
Muse-G: Second Question

- The next probed attribute is `cname`

Example source

<table>
<thead>
<tr>
<th>Companies</th>
<th>S. Valley</th>
<th>Microsoft</th>
<th>USA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mt. View</td>
<td>Google</td>
<td>USA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Projects</th>
<th>P1 DB</th>
<th>S. Valley</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P4 Web</td>
<td>Mt. View</td>
<td>e6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th>e4 John</th>
<th>x234</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e6 Kat</td>
<td>x331</td>
</tr>
</tbody>
</table>

Target Scenario 1

- **Orgs**: Microsoft
  - Projects: DB e4
  - Projects: Google
    - Projects: Web e6

- Employees: e4 John, e6 Kat

Target Scenario 2

- Orgs: Microsoft
  - Projects: DB e4
  - Projects: Google
    - Projects: Web e6

- Employees: e4 John, e6 Kat
Muse-G: Second Question

- The next probed attribute is **cname**

---

**Target Scenario 1**

**group by cname**

### Orgs
- **Microsoft**
  - Projects: SK(Microsoft,y)
    - DB: e4
- **Google**
  - Projects: SK(Google,y)
    - Web: e6

### Employees
- e4 John
- e6 Kat

---

**Target Scenario 2**

**do not group by cname**

### Orgs
- **Microsoft**
  - Projects: SK(y)
    - DB: e4
- **Google**
  - Projects: SK(y)
    - Web: e6

### Employees
- e4 John
- e6 Kat

\[ y \subseteq \{ \text{USA} \} \]
Muse-G: Second Question

- The next probed attribute is **cname**

**Example source**

<table>
<thead>
<tr>
<th>Companies</th>
<th>Projects</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Valley Microsoft USA</td>
<td>P1 DB S. Valley e4</td>
<td>e4 John x234</td>
</tr>
<tr>
<td>Mt. View Google USA</td>
<td>P4 Web Mt. View e6</td>
<td>e6 Kat x331</td>
</tr>
</tbody>
</table>

**Target Scenario 1**

- group by **cname**

<table>
<thead>
<tr>
<th>Orgs</th>
<th>Projects: SK(Company,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td>DB e4</td>
</tr>
<tr>
<td>Google</td>
<td>Web e6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>e4 John x234</td>
</tr>
<tr>
<td>e6 Kat x331</td>
</tr>
</tbody>
</table>

**Target Scenario 2**

- do not group by **cname**

<table>
<thead>
<tr>
<th>Orgs</th>
<th>Projects: SK(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td>DB e4</td>
</tr>
<tr>
<td>Google</td>
<td>Web e6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>e4 John x234</td>
</tr>
<tr>
<td>e6 Kat x331</td>
</tr>
</tbody>
</table>

y ⊆ { USA }

- The wizard continues to probe the remaining possible grouping attributes
Obtaining Source Examples

Running queries over the real source instance $I$
Example: probing on `cname`
Obtaining Source Examples

Running queries over the real source instance $I$

Example: probing on $cname$

**Query:**

$\text{Companies}(c_1,n_1,l_1) \land$

$\text{Projects}(p_1,pn_1,c_1,e_1) \land$

$\text{Employees}(e_1,en_1,cn_1) \land$
Obtaining Source Examples

Running queries over the real source instance $I$
Example: probing on `cname`

**Query:**

$\text{Companies}(c_1,n_1,l_1) \land$

$\text{Projects}(p_1,pn_1,c_1,e_1) \land$

$\text{Employees}(e_1,en_1,cn_1) \land$

$\text{Companies}(c_2,n_2,l_1) \land$

$\text{Projects}(p_2,pn_2,c_2,e_2) \land$

$\text{Employees}(e_2,en_2,cn_2) \land$
Obtaining Source Examples

Running queries over the real source instance $I$

Example: probing on $cname$

Query:

$\text{Companies}(c_1,n_1,l_1) \land$

$\text{Projects}(p_1,pn_1,c_1,e_1) \land$

$\text{Employees}(e_1,en_1,cn_1) \land$

$\text{Companies}(c_2,n_2,l_1) \land$

$\text{Projects}(p_2,pn_2,c_2,e_2) \land$

$\text{Employees}(e_2,en_2,cn_2) \land$

$n_1 \neq n_2$
Obtaining Source Examples

Running queries over the real source instance $I$

Example: probing on $c$name

Query:

$\text{Companies}(c_1,n_1,l_1) \land$

$\text{Projects}(p_1,pn_1,c_1,e_1) \land$

$\text{Employees}(e_1,en_1,cn_1) \land$

$\text{Companies}(c_2,n_2,l_1) \land$

$\text{Projects}(p_2,pn_2,c_2,e_2) \land$

$\text{Employees}(e_2,en_2,cn_2) \land$

$n_1 \neq n_2$

Real Example:

<table>
<thead>
<tr>
<th>Companies</th>
<th>Projects</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Valley</td>
<td>DB</td>
<td>e4 John</td>
</tr>
<tr>
<td>Mt. View</td>
<td>Web</td>
<td>x234</td>
</tr>
<tr>
<td>Google</td>
<td>S. Valley</td>
<td>e6 Kat</td>
</tr>
<tr>
<td>USA</td>
<td>Mt. View</td>
<td>x331</td>
</tr>
<tr>
<td>USA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Obtaining Source Examples

Running queries over the real source instance \( I \)
Example: probing on `cname`

**Query:**

\[
\begin{align*}
\text{Companies}(c_1, n_1, l_1) \land \\
\text{Projects}(p_1, pn_1, c_1, e_1) \land \\
\text{Employees}(e_1, en_1, cn_1) \land \\
\text{Companies}(c_2, n_2, l_1) \land \\
\text{Projects}(p_2, pn_2, c_2, e_2) \land \\
\text{Employees}(e_2, en_2, cn_2) \land \\
\ n_1 \neq n_2
\end{align*}
\]

**Real Example:**

<table>
<thead>
<tr>
<th>Companies</th>
<th>Projects</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Valley</td>
<td>Microsoft</td>
<td>USA</td>
</tr>
<tr>
<td>Mt. View</td>
<td>Google</td>
<td>USA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Projects</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>e4 John x234</td>
</tr>
<tr>
<td>P4</td>
<td>e6 Kat x331</td>
</tr>
</tbody>
</table>

**Synthetic Example:**

<table>
<thead>
<tr>
<th>Companies</th>
<th>Projects</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_1 n_1 l_1</td>
<td>p_1 pn_1 c_1 e_1</td>
<td></td>
</tr>
<tr>
<td>c_2 n_2 l_1</td>
<td>p_2 pn_2 c_2 e_2</td>
<td></td>
</tr>
</tbody>
</table>

| e_1 en_1 cn_1 |
| e_2 en_2 cn_2 |
Muse-G with FDs

- Considering *functional dependencies* in the source can reduce the number of questions posed to the designer
Muse-G with FDs

- Considering functional dependencies in the source can reduce the number of questions posed to the designer.

- Two mappings $M_1, M_2$ have the same effect if for any source instance $I$, the result of exchanging $I$ with $M_1$ is the “same” as the result of exchanging $I$ with $M_2$.

Homomorphically equivalent
Muse-G with FDs

- Considering **functional dependencies** in the source can reduce the number of questions posed to the designer.

- Two mappings \( M_1, M_2 \) have the **same effect** if for any source instance \( I \), the result of exchanging \( I \) with \( M_1 \) is the “same” as the result of exchanging \( I \) with \( M_2 \).

**Proposition.** If a FD \( A \rightarrow B \) holds, then a mapping \( M \) that groups by \( A \) has the same effect as a mapping \( M \) that groups by \( A \cup C \), where \( C \subseteq B \).
Muse-G with FDs

- Considering functional dependencies in the source can reduce the number of questions posed to the designer.

- Two mappings $M_1$, $M_2$ have the same effect if for any source instance $I$, the result of exchanging $I$ with $M_1$ is the “same” as the result of exchanging $I$ with $M_2$.

**Proposition.** If a FD $A \rightarrow B$ holds, then a mapping $M$ that groups by $A$ has the same effect as a mapping $M$ that groups by $A \cup C$, where $C \subseteq B$.

- Suppose `cbranch` is a key, then we may save some questions.
  - If the designer chooses Scenario 1 (including `cbranch` in the grouping function), probing on `cname` or `location` is no longer necessary.
Muse-G: Properties

**Proposition (Completeness).** If there are $n$ possible grouping attributes for a nested set $S$, then the questions asked by Muse-G explore the entire space of $2^n$ grouping functions. Muse-G asks at most $n$ questions to infer the desired grouping semantics for $S$. 
Muse-G: Properties

**Proposition (Completeness).** If there are \( n \) possible grouping attributes for a nested set \( S \), then the questions asked by Muse-G explore the entire space of \( 2^n \) grouping functions. Muse-G asks at most \( n \) questions to infer the desired grouping semantics for \( S \).

**Proposition (Small examples).** At each probe, Muse-G constructs a source example of size at most twice the number of conjuncts in the for clause of the mapping.
Muse-G: Properties

Proposition (Completeness). If there are $n$ possible grouping attributes for a nested set $S$, then the questions asked by Muse-G explore the entire space of $2^n$ grouping functions. Muse-G asks at most $n$ questions to infer the desired grouping semantics for $S$.

Proposition (Small examples). At each probe, Muse-G constructs a source example of size at most twice the number of conjuncts in the for clause of the mapping.

- Incremental design: group more or less starting from an existing grouping function
Muse-G: Properties

**Proposition (Completeness).** If there are $n$ possible grouping attributes for a nested set $S$, then the questions asked by Muse-G explore the entire space of $2^n$ grouping functions. Muse-G asks at most $n$ questions to infer the desired grouping semantics for $S$.

**Proposition (Small examples).** At each probe, Muse-G constructs a source example of size at most twice the number of conjuncts in the for clause of the mapping.

- Incremental design: group more or less starting from an existing grouping function
- Design for a specific source instance: reduce the number of questions
  - Muse-G identifies attributes whose inclusion/exclusion as arguments of grouping functions is inconsequential
Ambiguous Mappings

CompDB: $Rcd$

Projects: Set of

Project: $Rcd$
  pid
  pname
  manager
  tech-lead

Employees: Set of

Employee: $Rcd$
  eid
  ename
  contact
Ambiguous Mappings

CompDB: $Rcd$
- Projects: $Set$ of
  - Project: $Rcd$
    - pid
    - pname
    - manager
    - tech-lead
- Employees: $Set$ of
  - Employee: $Rcd$
    - eid
    - ename
    - contact

OrgDB: $Rcd$
- Projects: $Set$ of
  - Project: $Rcd$
    - pname
    - supervisor
    - email
Ambiguous Mappings

**CompDB:** $Rcd$

- **Projects:** Set of $Rcd$
  - **Project:** $Rcd$
    - pid
    - pname
    - manager
    - tech-lead

- **Employees:** Set of $Rcd$
  - **Employee:** $Rcd$
    - eid
    - ename
    - contact

**OrgDB:** $Rcd$

- **Projects:** Set of $Rcd$
  - **Project:** $Rcd$
    - pname
    - supervisor
    - email
Ambiguous Mappings

CompDB: $Rcd$

- **Projects**: $Set of$
  - **Project**: $Rcd$
    - pid
    - pname
    - manager
    - tech-lead

- **Employees**: $Set of$
  - **Employee**: $Rcd$
    - eid
    - ename
    - contact

OrgDB: $Rcd$

- **Projects**: $Set of$
  - **Project**: $Rcd$
    - pname
    - supervisor
    - email

for $p$ in CompDB.Projects
for $e_1$ in CompDB.Employees
for $e_2$ in CompDB.Employees
satisfy
    $e_1.eid = p.manager$
    $e_2.eid = p.tech-lead$
Ambiguous Mappings

CompDB: $Rcd$

Projects: $Set$ of

Project: $Rcd$
  pid
  pname
  manager
  tech-lead

Employees: $Set$ of

Employee: $Rcd$
  eid
  ename
  contact

OrgDB: $Rcd$

Projects: $Set$ of

Project: $Rcd$
  pname
  supervisor
  email

for
  p in CompDB.Projects
  e₁ in CompDB.Employees
  e₂ in CompDB.Employees

satisfy
  e₁.eid = p.manager
  e₂.eid = p.tech-lead

exists
  p₁ in OrgDB.Projects
for p in CompDB.Projects
    e₁ in CompDB.Employees
    e₂ in CompDB.Employees
    satisfy
        e₁.eid = p.manager
        e₂.eid = p.tech-lead
    exists
        p₁ in OrgDB.Projects
    where
        p.pname = p₁.pname
Ambiguous Mappings

CompDB: $Rcd$

- Projects: Set of
  - Project: $Rcd$
    - pid
    - pname
    - manager
    - tech-lead

Employees: Set of

- Employee: $Rcd$
  - eid
  - ename
  - contact

OrgDB: $Rcd$

- Projects: Set of
  - Project: $Rcd$
    - pname
    - supervisor
    - email

for $p$ in CompDB.Projects
  $e_1$ in CompDB.Employees
  $e_2$ in CompDB.Employees

satisfy
  $e_1.eid = p.manager$
  $e_2.eid = p.tech-lead$

exists
  $p_1$ in OrgDB.Projects

where
  $p.pname = p_1.pname$
  $p_1.supervisor = e_1.ename ~or~ e_2.ename$
Ambiguous Mappings

**CompDB:** $Rcd$
- **Projects:** Set of
  - **Project:** $Rcd$
    - pid
    - pname
    - manager
    - tech-lead
- **Employees:** Set of
  - **Employee:** $Rcd$
    - eid
    - ename
    - contact

**OrgDB:** $Rcd$
- **Projects:** Set of
  - **Project:** $Rcd$
    - pname
    - supervisor
    - email

---

- This mapping is *ambiguous*
- There are *four alternative interpretations*

```
for p in CompDB.Projects
  e₁ in CompDB.Employees
  e₂ in CompDB.Employees
  satisfy
    e₁.eid = p.manager
    e₂.eid = p.tech-lead
  exists
    p₁ in OrgDB.Projects
    where
      p.pname = p₁.pname
      p₁.supervisor =
        e₁.ename or e₂.ename
      p₁.email =
        e₁.contact or e₂.contact
```

---

**Ambiguous Elements**

- $e₁.ename$
- $e₁.contact$
- $e₂.ename$
- $e₂.contact$
Muse-D: Disambiguating Mappings

- **Key idea:** provide an example that illustrates the alternative interpretations in a compact way
Muse-D: Disambiguating Mappings

- **Key idea:** provide an example that illustrates the alternative interpretations in a **compact way**

**Projects**

```
P1 DB  e4  e5
```

**Employees**

```
e4  John  john@ibm
e5  Anna  anna@ibm
```
Muse-D: Disambiguating Mappings

**Key idea:** provide an example that illustrates the alternative interpretations in a **compact way**

**Projects**

- P1  DB  e4  e5

**Employees**

- e4  John  john@ibm
- e5  Anna  anna@ibm

**Orgs**

- Projects:
  - DB  John  john@ibm
  - Anna  anna@ibm
Muse-D: Disambiguating Mappings

- **Key idea:** provide an example that illustrates the alternative interpretations in a *compact way*

Projects
- P1  DB  e4  e5

Employees
- e4  John  john@ibm
- e5  Anna  anna@ibm

Orgs
- Projects:
  - DB  John  john@ibm
  - Anna  anna@ibm

- The mapping designer makes **one choice for each ambiguous element**
- Each decision removes one ambiguity
Muse-D: Disambiguating Mappings

- **Key idea:** provide an example that illustrates the alternative interpretations in a **compact way**

- **Projects**
  - P1: DB, e4, e5

- **Employees**
  - e4: John, john@ibm
  - e5: Anna, anna@ibm

- **Orgs**
  - DB: John, john@ibm
  - Anna, anna@ibm

- The mapping designer makes **one choice for each ambiguous element**

- Each decision removes one ambiguity

  - E.g., choosing “Anna” as the supervisor and “john@ibm” as the email

  \[
  p_1.\text{supervisor} = \begin{cases} 
  e_1.\text{ename} & \text{or} \\
  e_2.\text{ename} & 
  \end{cases}
  \]

  \[
  p_1.\text{email} = \begin{cases} 
  e_1.\text{contact} & \text{or} \\
  e_2.\text{contact} & 
  \end{cases}
  \]
Muse-D: Disambiguating Mappings

- **Key idea:** provide an example that illustrates the alternative interpretations in a compact way

**Projects**
- P1 DB e4 e5

**Employees**
- e4 John john@ibm
- e5 Anna anna@ibm

**Orgs**
- Projects:
  - DB John john@ibm
  - Anna anna@ibm

- The mapping designer makes one choice for each ambiguous element
- Each decision removes one ambiguity
  - E.g., choosing “Anna” as the supervisor and “john@ibm” as the email

\[
\begin{align*}
p_1.\text{supervisor} & = e_2.\text{ename} \\
p_1.\text{email} & = e_1.\text{contact} \text{ or } e_2.\text{contact}
\end{align*}
\]
Muse-D: Disambiguating Mappings

- **Key idea:** provide an example that illustrates the alternative interpretations in a **compact way**

<table>
<thead>
<tr>
<th>Projects</th>
<th>Orgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>DB</td>
</tr>
<tr>
<td>e4</td>
<td>John</td>
</tr>
<tr>
<td>e5</td>
<td>Anna</td>
</tr>
</tbody>
</table>

- **Employees**
  - e4 John john@ibm
  - e5 Anna anna@ibm

- **Orgs**
  - John john@ibm
  - Anna anna@ibm

- The mapping designer makes **one choice for each ambiguous element**
- Each decision removes one ambiguity
  - E.g., choosing “Anna” as the supervisor and “john@ibm” as the email

\[
p_1.\text{supervisor} = e_2.\text{ename} \quad \quad p_1.\text{email} = e_1.\text{contact}
\]
Obtaining Source Examples

Running queries over the real source instance
Obtaining Source Examples

Running queries over the real source instance

Query:
Projects($p_1$, $p_{n_1}$, $e_1$, $e_2$) $\land$
Employees($e_1$, $e_{n_1}$, $c_{n_1}$) $\land$
Employees($e_2$, $e_{n_2}$, $c_{n_2}$) $\land$
Obtaining Source Examples

Running queries over the real source instance

Query:
Projects(p₁,pn₁,e₁,e₂) ∧
Employees(e₁,en₁,cn₁) ∧
Employees(e₂,en₂,cn₂) ∧
en₁ ≠ en₂ ∧ cn₁ ≠ cn₂
Obtaining Source Examples

Running queries over the real source instance

Query:

Projects(p₁,pn₁,e₁,e₂) \land
Employees(e₁,en₁,cn₁) \land
Employees(e₂,en₂,cn₂) \land
en₁ \neq en₂ \land cn₁ \neq cn₂

Non-empty result

Real Example:

Projects
P1 DB e4 e5

Employees
e4 John john@ibm
e5 Anna anna@ibm
Obtaining Source Examples

Running queries over the real source instance

Query:
Projects\((p_1, p_{n_1}, e_1, e_2) \land \)\nEmployees\((e_1, e_{n_1}, e_{n_1}) \land \)\nEmployees\((e_2, e_{n_2}, e_{n_2}) \land \)\nen_{1} \neq en_{2} \land cn_{1} \neq cn_{2}

Real Example:
Projects
P1 DB e4 e5
Employees
e4 John john@ibm
e5 Anna anna@ibm

Synthetic Example:
Projects
p_1 p_{n_1} e_1 e_2
Employees
e_1 en_{1} cn_{1}
e_2 en_{2} cn_{2}
Muse-D: Properties

- For each ambiguous mapping, the designer is presented with a single example.
Muse-D: Properties

- For each ambiguous mapping, the designer is presented with a single example.

**Proposition (Completeness).** The single example differentiates among all the alternative interpretations of the ambiguous mapping. The mapping designer has to make a number of choices equal to the number of ambiguous elements.
For each ambiguous mapping, the designer is presented with a single example.

**Proposition (Completeness).** The single example differentiates among all the alternative interpretations of the ambiguous mapping. The mapping designer has to make a number of choices equal to the number of ambiguous elements.

**Proposition (Small examples).** The number of tuples in the example source instance is the number of conjuncts in the for clause of the mapping.
## Experiments: Setting

<table>
<thead>
<tr>
<th>Mapping Scenarios</th>
<th>Size of real source instance</th>
<th>Sets with refinable grouping</th>
<th>Number of mappings</th>
<th>Ambiguous mappings</th>
<th>Alternative interpretations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mondial</td>
<td>1MB</td>
<td>8</td>
<td>26</td>
<td>7</td>
<td>208</td>
</tr>
<tr>
<td>DBLP</td>
<td>2.6MB</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>TPCH</td>
<td>10MB</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Amalgam</td>
<td>2MB</td>
<td>2</td>
<td>14</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>

- Mapping system: Clio
- Query evaluation: DB2 v9, Saxon-B 8.9
- MUSE implementation: Java 6
### Experiments: Muse-G

<table>
<thead>
<tr>
<th>Mapping scenario</th>
<th>Average # of grouping attributes</th>
<th>Number of questions (average)</th>
<th>% times found real example</th>
<th>Average time to get real example (s)</th>
<th>Grouping strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mondial</td>
<td>13.1</td>
<td>2.6</td>
<td>38%</td>
<td>0.014</td>
<td>$G_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.5</td>
<td>41%</td>
<td>0.187</td>
<td>$G_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.9</td>
<td>40%</td>
<td>0.015</td>
<td>$G_3$</td>
</tr>
<tr>
<td>DBLP</td>
<td>11</td>
<td>1.5</td>
<td>17%</td>
<td>0.450</td>
<td>$G_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>11%</td>
<td>0.337</td>
<td>$G_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.5</td>
<td>17%</td>
<td>0.454</td>
<td>$G_3$</td>
</tr>
<tr>
<td>TPCH</td>
<td>26.7</td>
<td>1.5</td>
<td>0%</td>
<td>0.785</td>
<td>$G_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17</td>
<td>12%</td>
<td>0.893</td>
<td>$G_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.5</td>
<td>0%</td>
<td>0.782</td>
<td>$G_3$</td>
</tr>
<tr>
<td>Amalgam</td>
<td>14.1</td>
<td>2</td>
<td>29%</td>
<td>0.013</td>
<td>$G_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>52%</td>
<td>0.043</td>
<td>$G_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>52%</td>
<td>0.030</td>
<td>$G_3$</td>
</tr>
</tbody>
</table>

$G_1$: group by all possible attributes (cbranch,cname,location,pid,...)

$G_2$: group by all attributes exported above the set (cname)

$G_3$: group by all exported attributes (cname, pname, eid, ename)
## Experiments: Muse-D

<table>
<thead>
<tr>
<th>Mapping Scenario</th>
<th>Alternatives encoded</th>
<th>Number of questions</th>
<th>Size of source example (# of tuples)</th>
<th>Number of ambiguous values in target instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mondial</td>
<td>208</td>
<td>7</td>
<td>3–4</td>
<td>4–5</td>
</tr>
<tr>
<td>TPCH</td>
<td>16</td>
<td>1</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>
Related Work

- There are many related works, we focus here on the closest one
- We were inspired by DataViewer [Yan et al. 01]

<table>
<thead>
<tr>
<th>Feature</th>
<th>System</th>
<th>MUSE</th>
<th>DataViewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples to be analyzed</td>
<td></td>
<td>Compact representation (e.g. 7)</td>
<td>As many as alternative interpretations (e.g. 208)</td>
</tr>
<tr>
<td>Grouping Semantics</td>
<td></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Data Models</td>
<td></td>
<td>Relational and XML</td>
<td>Relational</td>
</tr>
<tr>
<td>Mapping Language</td>
<td></td>
<td>Source-target mappings (GLAV)</td>
<td>SQL</td>
</tr>
</tbody>
</table>
Conclusion

- **MUSE: a mapping design wizard**
  - Use examples to understand, design, refine schema mappings
  - Work with **complete and small data examples** rather than complex specifications
  - Focus on two important aspects of a mapping specification
    - Grouping semantics
    - Interpretation of ambiguous mappings
Thank you