Web Data Integration

Schema Mapping and Data Translation
Outline

1. The Two Basic Integration Situations
2. Correspondences
3. Schema Integration
4. Data Translation
5. Schema Matching
6. Schema Heterogeneity on the Web
7. References
Goal: Translate data from a set of source schemata into a **given target schema**.

- Top-down integration situation
- Triggered by concrete information need (= target schema)
The Schema Mapping Process

1. Find Correspondences

2. Interpretation

3. Query Generation

Source Schema

Target Schema

Source Data

Materialized Target Data

Transformation Queries (Transformation Program)
Goal: Create a new target schema that can represent all data from a set of given data sources.

- Bottom-up integration situation
- Triggered by the goal to fulfill different information needs based on data from all sources.
2. Correspondences

A correspondence relates a set of elements in a schema S to a set of elements in schema T.

- **Mapping** = Set of all correspondences that relate S and T
- Correspondences are **easier to specify** than transformation queries
  - domain expert does not need technical knowledge about query language
  - specification can be supported by user interface
  - step-by-step process with separate local decisions
- Correspondences can be annotated with a **transformation function**
  - transform and calculate values
  - normalize units of measurement
  - cast attribute data types
  - translate values using a translation table
Types of Correspondences

- **One-to-One Correspondences**
  - Movies.title $\rightarrow$ Items.name
  - Products.rating $\rightarrow$ Items.classification
  - Movie $\equiv$ Film (same semantic intention)
  - Athlete $\subseteq$ Person (inclusion: All athletes are also persons)

- **One-to-Many Correspondences**
  - Person.Name $\rightarrow$ split() $\rightarrow$ FirstName (Token 1) $\rightarrow$ Surname (Token 2)

- **Many-to-One Correspondences**
  - Products.basePrice * (1 + Locations.taxRate) $\rightarrow$ Items.price

- **Higher-Order Correspondences**
  - relate different types of data model elements
  - for example: Relations (classes) and attributes, see next slide
Examples of Higher-Order Correspondences

Relation-to-Attribute Correspondences

Man
Firstname  
Surname
Woman
Firstname  
Surname
Person
Firstname  
Surname
Sex

Attribute-to-Relation Correspondences

Person
Firstname  
Surname
Sex
= ‘m’

= ‘f’

Man
Firstname  
Surname
Woman
Firstname  
Surname
Types of Schema Heterogeneity that can be captured

- Modelling
  - Relation vs. Attribute
  - Attribute vs. Value
  - Relation vs. Value

- Naming of
  - Relations
  - Attributes

- Normalized vs. Denormalized

- Nesting vs. Foreign Keys

- Higher-order Correspondences
  - 1:1, 1:n, n:1 Correspondences
Example: Alternative Modelling

- **Man( Id, Firstname, Surname)**
- **Woman( Id, Firstname, Surname)**
- **Relation vs. Attribute**
- **Person( Id, Firstname, Surname, Male, Female)**
- **Relation vs. Value**
- **Attribute vs. Value**

- **Person( Id, Firstname, Surname, Sex)**
Defining Correspondences
Automatic Discovery of Correspondences

Schema Matching: Automatically or semi-automatically discover correspondences between schemata.

- Various schema matching methods exist (we will cover them later)
- Automatically finding a complete high-quality mapping is not possible in most real-world cases. Halevy: „It‘s plain hard.“ :-(
- In practice, schema matching is used to create candidate correspondences that are verified by domain experts afterwards.
- Realistic goal: Reduce the effort of the domain expert.
Create a **new target schema** that can represent all data from a set of given data sources.

- **Goals:**
  - **Completeness:** All elements of the source schemata should be covered.
    - not always necessary
  - **Correctness:** All data should be represented semantically correct.
    - integrity constraints, cardinalities, …
  - **Minimality:** The integrated schema should be minimal in respect to the number of relations and attributes.
    - redundancy-free
  - **Understandability:** The schema should be easy to understand.

- Various schema integration „procedures“ have been proposed in literature (see Leser/Naumann, Chapter 5.6).

- They are rather guidelines for the domain expert than concrete algorithms.
Schema Integration Steps

1. **Pre-Integration**
   - Convert sources into single data model (relational, XML, RDF, object-oriented)
   - If more than two schemata, decide in which order to integrate schemata

2. **Schema Comparison**
   - Find correspondences (manually or using schema matching)
   - Identity conflicts (normalization, nesting, relation vs. attribute vs. value, …)

3. **Schema Normalization**
   - Transformation of individual schemata in order to resolve conflicts
   - Change structure (normalization, nesting, relation vs. attribute vs. value, …)

4. **Schema Fusion**
   - Generate integrated schema based on the correspondences
   - Merge equivalent relations, resolve naming conflicts
   - Further restructuring and deletion of redundant attributes
Example of a Schema Integration Method


- Input
  - Two Schemata in **Generic Data Model**
    - Classes, Attributes, and Relationships
    - similar to Entity-Relationship-Model
    - subsumes many other data models (e.g. relational model)
  - **Correspondence Assertions (CA)**
    - Correspondences between classes, attributes, and relationships
    - Correspondences between paths of relationships between classes
    - in theory, the method can cover various types of correspondences
    - we here restrict ourselves to 1:1 equivalence correspondences

- Output: Integrated Schema
Integration Rules

Include into the target schema S

1. **Classes** with their attributes, that are not part of any class-class correspondence (classes without direct equivalent)

2. **Equivalent classes** and merge their attribute sets
   - Pick class / attribute names of your choice for equivalent classes / attributes

3. **Direct relationships** between equivalent classes
   - $A \equiv A', B \equiv B', A-B \equiv A'-B'$: Then include $A-B$

4. **Paths** between equivalent attributes and classes
   - $A \equiv A', B \equiv B', A-B \equiv A'-A_1'-\ldots-A_m'-B'$: Include the longer path
     - The shorter one is subsumed by the longer one
     - The longer one is more expressive with respect to cardinality
   - $A \equiv A', B \equiv B', A-A_1-\ldots-A_n-B \equiv A'-A_1'-\ldots-A_m'-B'$: Include both paths
     - As they represent different relationships to $B$

5. **Equivalences between classes and attributes** are included as relationships
   - again, prefer more expressive solution with respect to cardinality
Example

- \text{film} \equiv \text{spielfilm}
  - \text{id} \equiv \text{film\_id}
  - \text{titel} \equiv \text{name}

- \text{r\_name} \equiv \text{regisseur}

- \text{filmstudio} \equiv \text{s\_name}
Example

- film ≡ spielfilm
  - id ≡ film_id und titel ≡ name
- r_name ≡ regisseur
- filmstudio ≡ s_name
- filmstudio-fuehrt_regie-film ≡ s_name-studio-spielfilm
Example

- \texttt{film} \equiv \texttt{spielfilm}
  - \texttt{id} \equiv \texttt{film}\_id \text{ und } \texttt{titel} \equiv \texttt{name}

- \texttt{r}\_\texttt{name} \equiv \texttt{regisseur}

- \texttt{filmstudio} \equiv \texttt{s}\_\texttt{name}

- \texttt{filmstudio-fuehrt_regie-film} \equiv \texttt{s}\_\texttt{name}-\texttt{studio}-\texttt{spielfilm}

\texttt{r}\_\texttt{name}-\texttt{regisseur}-\texttt{fuehrt_regie-film} \equiv \texttt{regisseur}-\texttt{spielfilm}
Example

- Correspondences
  - \textit{film} = \textit{spielfilm}
    - \textit{id} = film\_id und \textit{titel} = name
  - \textit{r\_name} = \textit{regisseur}
  - \textit{filmstudio} = s\_name
  - \textit{filmstudio\-fuehrt\_regie\-film} = s\_name\-studio\-spielfilm
  - \textit{r\_name\-regisseur\-fuehrt\_regie\-film} = regisseur\-spielfilm

- Integrated Schema
  - Rule 2: equivalent classes \textit{film} and \textit{spielfilm} are merged (to \textit{film})
  - Rule 1: Classes without direct equivalent are included into the integrated schema (\textit{regisseur}, \textit{fuehrt\_regie}, \textit{studio})
Example

- **Correspondences**
  - film $\equiv$ spielfilm
    - id $\equiv$ film_id und titel $\equiv$ name
  - r_name $\equiv$ regisseur
  - filmstudio $\equiv$ s_name
  - filmstudio-fuehrt_regie-film $\equiv$ s_name-studio-spielfilm
  - r_name-regisseur-fuehrt_regie-film $\equiv$ regisseur-spielfilm

- **Integrated Schema**
  - Rule 4a: The path r_name-regisseur-fuehrt_regie-film is included, the path regisseur-spielfilm is left out.
  - Thus, r_name and regisseur are merged.
Example

- Correspondences
  - `film` ≡ `spielfilm`
    - `id` ≡ `film_id` und `titel` ≡ `name`
  - `r_name` ≡ `regisseur`
  - `filmstudio` ≡ `s_name`
  - `filmstudio-fuehrt_regie-film` ≡ `s_name-studio-spielfilm`
  - `r_name-regisseur-fuehrt_regie-film` ≡ `regisseur-spielfilm`

- Integrated Schema
  - Rule 4b: Both paths are included
    - `filmstudio` and `s_name` are not merged as they have a different relationship to the surrounding classes.

The movie was shot in this studio?

Studio owns rights?
Final Integrated Schema

- Schema Integration Goals:
  - **Completeness**: All elements of the source schemata should be covered.
  - **Correctness**: All data should be represented semantically correct.
  - **Minimality**: The integrated schema should be minimal in respect to the number of relations and attributes.
  - **Understandability**: The schema should be easy to understand.
4. Data Translation

Find Correspondences

Correspondences

We are here
Interpretation and Query Generation

Goal: Interpret correspondences in order to generate suitable data translation queries.

- Query Types: SQL Insert Into/Select Into, SPARQL Construct, XSLT
- Simple Example

```
SELECT artPK AS pubID
    heading AS title
null AS date
INTO PUBLICATION
FROM ARTICLE
```

- Challenges for more complex schemata
  - Correspondences are not isolated but embedded into context (tables, relationships)
  - How to join tables or merge/split attributes in order to overcome different levels of normalization?
  - Which join paths to choose if there are different possibilities?
  - How to union results from multiple source tables / joins (horizontal partitioning)?
Normalized ➔ Denormalized

- ARTICLE
  - artPK
  - title
  - pages
- AUTHOR
  - artFK
  - name
- PUBLICATION
  - pubID
  - title
  - date
  - author

Example SQL queries:

```
SELECT artPK AS pubID, title AS title, null AS date, null AS author
FROM ARTICLE
UNION
SELECT null AS pubID, null AS title, null AS date, name AS author
FROM AUTHOR
```
Normalized ➔ Denormalized

Better approach: Use foreign key relationship to join tables.

```sql
SELECT artPK AS pubID, title AS title, null AS date, name AS author
FROM ARTICLE, AUTHOR
WHERE ARTICLE.artPK = AUTHOR.artFK
```
INNER JOIN vs. OUTER JOIN

- **ARTICLE**
  - artPK
  - title
  - pages
- **AUTHOR**
  - artFK
  - name
- **PUBLICATION**
  - pubID
  - title
  - date
  - author

**Interpretation:** Do we want publications without author?

```
SELECT artPK AS pubID, title AS title, null AS date, name AS author
FROM ARTICLE LEFT OUTER JOIN AUTHOR
ON ARTICLE.artPK = AUTHOR.artFK
```
Which Join Path to choose?

Source Schema:

Target Schema:

**Interpretation:** Do we only want students that took the exam in the list?
Denormalized ➔ Normalized

**PUBLICATION**
- title
- date
- author

**ARTICLE**
- artPK
- title
- pages

**AUTHOR**
- artFK
- name

SELECT SK(title) AS artPK
    title AS title
    null AS pages
FROM PUBLICATION

SELECT SK(title) AS artFK
    author AS name
FROM PUBLICATION

**DISTINCT**

**SK()**: A Skolem function is used to generate unique keys from distinct values.
Horizontal Partitioning

Address
- id
- Addr

Professor
- id
- name
- salary

Student
- name
- GPA
- Yr

PayRate
- Rank
- HrRate

WorksOn
- name
- Proj
- hrs
- ProjRank

Personnel
- Sal

\[ c1: \text{Professor}(\text{Sal}) \rightarrow \text{Personnel}(\text{Sal}) \]
\[ c2: \text{PayRate}(\text{HrRate}) \ast \text{WorksOn}(\text{Hrs}) = \text{Personnel}(\text{Sal}) \]
UNION the salaries of Professors and Students

```sql
select P.HrRate * W.hrs
from PayRate P, WorksOn W
where P.Rank = W.ProjRank

UNION

select Sal
from Professor
```
Complete Algorithms for generating Queries

- Relational Case

- XML Case

- MapForce
  - implements another one which we will try in the exercise.

- Algorithms can not do the interpretation and thus need to be guided by the user.
5. Schema Matching

Schema Matching: Automatically or semi-automatically discover correspondences between schemata.

- Automatically finding a complete high-quality mapping is not possible in most real-world cases.
- In practice, schema matching is used to create candidate correspondences that are verified by domain experts afterwards.
- Schema matching methods focus on finding **1:1 correspondences**.
  - we restrict ourselves to 1:1 for now and speak about 1:n and n:1 later.
Outline Schema Matching

1. Challenges for finding Correspondences
2. Schema Matching Methods
   1. Label-based Methods
   2. Instance-based Methods
   3. Structure-based Methods
   4. Combined Approaches
3. Finding n:1 and 1:n Correspondences
4. Generating Correspondences from the Similarity Matrix
5. Summary and current Trends
Schema Matching

Source Schema

Interpretation

Logical Mapping

Query Generation

Find Correspondences

Correspondences

Source Data

Transformation Queries (Transformation Program)

Target Schema

Materialized Target Data

Transformation Queries (Transformation Program)

We are here
5.1 Challenges for finding Correspondences

- **Large Schemata**
  - >100 tables and >1000 attributes

- **Esoteric Naming Conventions and different Languages**
  - 4 character abbreviations: SPEY
  - city vs. ciudad vs. مدينه

- **Generic, automatically generated names**
  - attribute1, attribute2, attribute3 (product features in Amazon API)

- **Missing documentation**

- **„Strange“ Schemata**
  - Denormalization, Redundancies, …

- **Semantic Heterogeneity**
  - Synonyms, Homonyms, …
Problem Space: Large Schemata

Look at the scroll bar

Look at the depth of the schema
### Problem Space: Different Languages and Strange Names

#### Männer

<table>
<thead>
<tr>
<th>Vorname</th>
<th>Nachname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Felix</td>
<td>Naumann</td>
</tr>
<tr>
<td>Jens</td>
<td>Bleiholder</td>
</tr>
</tbody>
</table>

#### Frauen

<table>
<thead>
<tr>
<th>Vorname</th>
<th>Nachname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Melanie</td>
<td>Weis</td>
</tr>
<tr>
<td>Jana</td>
<td>Bauckmann</td>
</tr>
</tbody>
</table>

#### Personen

<table>
<thead>
<tr>
<th>Vorname</th>
<th>Nachname</th>
<th>male</th>
<th>female</th>
</tr>
</thead>
<tbody>
<tr>
<td>Felix</td>
<td>Naumann</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Jens</td>
<td>Bleiholder</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Melanie</td>
<td>Weiß</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jana</td>
<td>baukman</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vorname</th>
<th>Nachname</th>
<th>FN</th>
<th>NN</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Felix</td>
<td>Naumann</td>
<td>F</td>
<td>Naumann</td>
<td>M</td>
</tr>
<tr>
<td>Jens</td>
<td>Bleiholder</td>
<td>J</td>
<td>Bleiholder</td>
<td>M</td>
</tr>
<tr>
<td>Melanie</td>
<td>Weiß</td>
<td>M</td>
<td>Weiß</td>
<td>F</td>
</tr>
<tr>
<td>Jana</td>
<td>Bauckmann</td>
<td>J</td>
<td>Bauckmann</td>
<td>F</td>
</tr>
</tbody>
</table>
How do humans know?

- We recognize **naming conventions** and different languages
- use **table context**
- values look like first names and surnames
- values look similar
- if there is a first name, there is usually also a surname
- **persons** have first- and surnames
- **man** are persons

→ Recognizing these clues is hard for the computer.
5.2. Schema Matching Methods

1. **Label-based Methods**: Rely on the names of schema elements
2. **Instance-based Methods**: Use the actual data values
3. **Structure-based Methods**: Exploit the structure of the schema
4. **Combined Approaches**: Use combinations of above methods
Schema Matching Classification

5.2.1 Label-based Schema Matching Methods

- Given two schemata with the attribute (class) sets A and B
  - A={ID, Name, Vorname, Alter}, B={No, Name, First_name, Age}

- Approach
  1. generate cross product of all attributes (classes) from A and B
  2. for each pair calculate the similarity of the attribute labels
     - using some similarity function: Edit-distance, Jaccard, Soundex, etc.
     - we will cover similarity functions in detail in the block on Identity Resolution
  3. The most similar pairs are the matches

<table>
<thead>
<tr>
<th></th>
<th>ID</th>
<th>Name</th>
<th>Vorname</th>
<th>Alter</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>Name</td>
<td>0.1</td>
<td>1.0</td>
<td>0.6</td>
<td>0.3</td>
</tr>
<tr>
<td>First_name</td>
<td>0.2</td>
<td>0.6</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>Age</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
<td>0.7</td>
</tr>
</tbody>
</table>
Example Similarity Function: Edit / Levenshtein Distance

- Measures the minimum number of edits needed to transform one string into the other.

- Allowed edit operations:
  - insert a character into the string
  - delete a character from the string
  - replace one character with a different character

- Examples:
  - levensthein('Table', 'Cable') = 1 (1 Substitution)
  - levensthein('Table', 'able') = 1 (1 Deletion)

- Converting edit distance into a similarity

$$s(x, y) = 1 - \frac{d(x, y)}{\max(\text{length}(x), \text{length}(y))}$$
Problems of Label-based Schema Matching

- Semantic Heterogeneity is not recognized
  - the labels of schema elements only partly capture their semantics
  - synonyms and homonyms

- Problems with different naming conventions
  - Abbreviations: pers = person, dep = department
  - Combined terms and ordering: id_pers_dep vs. DepartmentPersonNumber
  - Different languages: city vs. ciudad vs. مدينة

- We need to apply **smart, application-specific tweaks**:
  1. Preprocessing: normalize labels in order prepare them for matching.
Pre-Processing of Labels

- Case and Punctuation Normalization
  - ISBN, IsbN, and I.S.B.N $\rightarrow$ isbn

- Explanation Removal
  - GDP (as of 2014, US$) $\rightarrow$ gdp

- Stop Word Removal
  - in, at, of, and, …
  - ex1:locatedIn $\rightarrow$ ex1:located
  - but: ex1:locationOf, ex2:locatedIn (Inverse Properties!)

- Stemming
  - ex1:located, ex2:location $\rightarrow$ both stemmed to 'locat'

- Tokenization
  - ex1:graduated_from_university $\rightarrow$ {graduated,from,university}
  - ex2:isGraduateFromUniversity $\rightarrow$ {is,Graduate,from,University}
  - tokens are then compared one-by-one using for instance Jaccard
Use Linguistic Resources for Pre-Processing / Matching

- Translate labels into target language
  - ciudad and مدينة into city

- Expand known abbreviations or acronyms
  - loc → location, cust → customer
  - using a domain-specific list of abbreviations or acronyms

- Expand with synonyms
  - add cost to price
  - using a dictionary with synonyms

- Expand with hypernyms (is-a relationships)
  - expand product into book, dvd, cd

- Use taxonomy/ontology containing hypernyms for matching
  - similarity = closeness of concepts within taxonomy/ontology
Useful External Resources

- **Google Translate**
  - recognizes languages and translates terms

- **WordNet**
  - provides synonyms and hypernyms for English words.

- **Wikipedia/DBpedia**
  - provides concept definitions, category system, DBpedia ontology, cross-language links
  - see Paulheim: WikiMatch. 2012.

- **The Web**
  - Google for terms, if result lists are similar then terms are similar
  - see Paulheim: WeSeE-Match. 2012.
5.2.2 Instance-based Schema Matching Methods

- Given two schemata with the attribute sets A and B and
  - all instances of A and B or
  - a sample of the instances of A and B

- Approach
  - Determine correspondences between A and B by examining which attributes in A and B contain similar values.

- Concrete Methods
  1. Use Attribute Recognizers
  2. Calculate Value Overlap
  3. Feature-based Methods
  4. Duplicate-based Methods

<table>
<thead>
<tr>
<th>Table A</th>
<th></th>
<th>Table B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Felix</td>
<td></td>
<td>Felix</td>
</tr>
<tr>
<td>Jens</td>
<td></td>
<td>Jens</td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Naumann</td>
<td></td>
<td>Naumann</td>
</tr>
<tr>
<td>Bleiholder</td>
<td></td>
<td>Bleiholder</td>
</tr>
</tbody>
</table>
1. Attribute Recognizers
   - employ dictionaries, regexes or rules to recognize values of a specific attribute
     - Dictionaries fit attributes that only contain a relatively small set of values (e.g. age classification of movies (G, PG, PG-13, R), country names, US states
     - Regexes or rules fit attributes with regular values (e.g. area code – phone number).
     - similarity = fraction of the values of attribute B that match dictionary/rule of attribute A

2. Value Overlap
   - calculate the similarity of attribute A and B as the overlap of their values using the Jaccard similarity measure (or Generalized Jaccard):
   \[
   J(A, B) = \frac{|A \cap B|}{|A \cup B|}
   \]
Feature-based Methods

- Given two schemata with the attribute sets A and B and instances of A and B.

- Approach
  1. For each attribute calculate interesting features using the instance data, e.g.
     - attribute data type
     - average string length of attribute values
     - average maximal and minimal number of words
     - average, maximal and minimal value of numbers
     - standard derivation of numbers
     - does the attribute contain NULL values?
  2. generate the cross product of all attributes from A and B
  3. for each pair compare the similarity of the features.
Example: Feature-based Matching

- Features: Attribute data type, average string length
  - A = {(ID, NUM, 1), (Name, STR, 6), (Loc, STR, 18)}
  - B = {(Nr, NUM, 1), (Adresse, STR, 16), (Telefon, STR, 11)}

- Similarity measure: **Euclidean Distance** (NUM=0, STR=1)

<table>
<thead>
<tr>
<th>Nr</th>
<th>Adresse</th>
<th>Telefon</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Seeweg, Berlin</td>
<td>030-3324566</td>
</tr>
<tr>
<td>3</td>
<td>Aalstr, Schwedt</td>
<td>0330-1247765</td>
</tr>
<tr>
<td>4</td>
<td>Rosenallee, Kochel</td>
<td>0884-334621</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Müller</td>
<td>Danziger Str, Berlin</td>
</tr>
<tr>
<td>2</td>
<td>Meyer</td>
<td>Boxhagenerstr, Berlin</td>
</tr>
<tr>
<td>4</td>
<td>Schmidt</td>
<td>Turmstr, Köln</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nr</th>
<th>Adresse</th>
<th>Telefon</th>
</tr>
</thead>
<tbody>
<tr>
<td>d(&lt;0,1&gt;,&lt;0,1&gt;)</td>
<td>d(&lt;1,6&gt;,&lt;0,1&gt;)</td>
<td>d(&lt;1,18&gt;,&lt;0,1&gt;)</td>
</tr>
<tr>
<td>d(&lt;0,1&gt;,&lt;1,16&gt;)</td>
<td>d(&lt;1,6&gt;,&lt;1,16&gt;)</td>
<td>d(&lt;1,18&gt;,&lt;1,16&gt;)</td>
</tr>
<tr>
<td>d(&lt;0,1&gt;,&lt;1,11&gt;)</td>
<td>d(&lt;1,6&gt;,&lt;1,11&gt;)</td>
<td>d(&lt;1,18&gt;,&lt;1,11&gt;)</td>
</tr>
</tbody>
</table>
Discussion: Feature-based Methods

- Requires decision which features to use.
  - Good features depend on the data type and application domain.

- Requires decision how to compare and combine values.
  - e.g. cosine similarity, Euclidian distance (of normalized values), …
  - different features should have different weights
  - similarity metrics are domain-dependent

- Similar attribute values do not always imply same Semantics.
  - Phone number / Fax number
  - Employee name / Customer name
Duplicate-based Methods

- Classical instance-based Matching in vertical
  - Comparison of complete columns
  - Ignores the relationship between columns and tables

- Horizontal instance-based Matching
  1. Compare all tuple pairs
  2. Find (some) potential duplicates or use previous knowledge about duplicates
  3. Check which attribute values closely match in the duplicates
  4. Result: attribute correspondences per duplicate
  5. Final Matching: Use majority voting
Example: Vote of Two Duplicates

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>Michel</td>
<td>m</td>
<td>601- 4839204</td>
<td>601- 4839204</td>
<td></td>
</tr>
<tr>
<td>Sam</td>
<td>Adams</td>
<td>m</td>
<td>541- 8127100</td>
<td>541- 8121164</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B'</th>
<th>F</th>
<th>E'</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michel</td>
<td>maxim</td>
<td>601- 4839204</td>
<td>UNIX</td>
</tr>
<tr>
<td>Adams</td>
<td>beer</td>
<td>541- 8127164</td>
<td>WinXP</td>
</tr>
</tbody>
</table>

Votes of the two duplicates:
Using Duplicates for Cross-Language Infobox Matching

Source: Felix Naumann, ICIQ 2012 Talk
Discussion

- Works well if duplicates are known or easy to find
  - owl:sameAs statements in LOD cloud
  - shared IDs like ISBN or GenID

- Can correctly distinguish very similar attributes
  - Telephone number <> fax number, Surname<>Maiden name

- In general, duplicate detection is tricky and computationally expensive
  - we will discuss this later in the block identity resolution
5.2.3 Structure-based Schema Matching Methods

- Addresses the following problem:

- **Attribute-Attribute-Matching**
  - Instance-based: Values of all attributes rather similar
  - Label-based: Labels of all attributes rather similar
  - All matching are about equally good 😊
Better approach: Exploit the attribute context

- Attributes that co-occur in one relation often (but not always) also co-occur in other relations.
- High similarity of neighboring attributes and name of relation increases similarity of attribute pair.
- Base similarities: label-based and/or instance based
- Simple calculation: Weighted sum of attribute similarity with similarity of attributes in same relation and similarity of relation names.
- Alternative calculation: Similarity Flooding algorithm (Paper see references)
5.2.4 Combined Approaches

- **Hybrid Approaches**
  - integrate different clues into single similarity function
  - Clues: e.g. labels and instance data

- **Ensembles**
  1. apply different base matchers
  2. combine their results
Example of the Need to Exploit Multiple Types of Clues

### Realestate.com

<table>
<thead>
<tr>
<th>listed-price</th>
<th>contact-name</th>
<th>contact-phone</th>
<th>office</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$250K</td>
<td>James Smith</td>
<td>(305) 729 0831</td>
<td>(305) 616 1822</td>
<td>Fantastic house</td>
</tr>
<tr>
<td>$320K</td>
<td>Mike Doan</td>
<td>(617) 253 1429</td>
<td>(617) 112 2315</td>
<td>Great location</td>
</tr>
</tbody>
</table>

- If we use only names
  - contact-agent matches either contact-name or contact-phone

### Homes.com

<table>
<thead>
<tr>
<th>sold-at</th>
<th>contact-agent</th>
<th>extra-info</th>
</tr>
</thead>
<tbody>
<tr>
<td>$350K</td>
<td>(206) 634 9435</td>
<td>Beautiful yard</td>
</tr>
<tr>
<td>$230K</td>
<td>(617) 335 4243</td>
<td>Close to Seattle</td>
</tr>
</tbody>
</table>

- If we use only data values
  - contact-agent matches either contact-phone or office

- If we use both names and data values
  - contact-agent matches contact-phone
How to Combine the Predictions of Multiple Matchers?

- Simple approaches: Use \texttt{avg()}, \texttt{min()} or \texttt{max()} function.

- When to use which combiner?
  
  - average combiner: when we do not have any reason to trust one matcher over the others
  
  - maximum combiner: when we trust a strong signal from matchers, i.e., if a matcher outputs a high value, we are relatively confident that the two elements match
  
  - minimum combiner: when we want to be more conservative
More complex Types of Combiners

- weighted-sum combiners
  - give weights to each matcher, according to its importance
  - you may learn the weights using
    - known correspondences as training data
    - linear regression or decision tree learning algorithms
    - we will cover learning weights in detail in the block about Identity Resolution

- use hand-crafted rules
  - e.g., if $s_i$ is address, return the score of the data-based matcher
    otherwise, return the average score of all matchers
5.3 Finding Many-to-One and One-to-Many Correspondences

- Up till now all methods only looked for 1:1 correspondences
- But real-world setting often require n:1 and 1:n or even n:m correspondences

- Question:
  - How to combine values?
  - Lots of functions possible.

- Problem:
  - Should we test $1.2 \times A + 2 \times B - 32 \equiv C$
  - … unlimited search space!

```
Vorname concat() \rightarrow \text{Name}
Nachname
```

```
Name \rightarrow \text{extract()} \rightarrow \text{Vorname}
Name \rightarrow \text{extract()} \rightarrow \text{Nachname}
```

```
Name \rightarrow \text{extract()} \rightarrow \text{concat()} \rightarrow \text{First name}
Title \rightarrow \text{extract()} \rightarrow \text{Last name}
```
Search for Complex Correspondences


- Employs specialized searchers:
  - Text searcher: uses only concatenations of columns
  - Numeric searcher: uses only basic arithmetic expressions
  - Date searcher: tries combination of numbers into dd/mm/yyyy pattern

- Key challenge: Control the search.
  - Start searching for 1:1 correspondences
  - add additional attributes one by one to sets
  - consider only top k candidates at every level of the search
  - termination based on diminishing returns
An Example: Text Searcher

**Mediated-schema**

<table>
<thead>
<tr>
<th>homes.com</th>
<th>price</th>
<th>num-baths</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>listed-price</td>
<td>agent-id</td>
<td>full-baths</td>
<td>half-baths</td>
</tr>
<tr>
<td>320K</td>
<td>532a</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>240K</td>
<td>115c</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Best match candidates for **address**
  - (agent-id, 0.7), (concat(agent-id, city), 0.75), (concat(city, zipcode), 0.9)
5.4 Generating Correspondences from the Similarity Matrix

- Input: matrix containing (combined) attribute similarities
- Output: A set of (candidate) correspondences

**Local Single Attribute Strategies:**

1. **Thresholding**
   - all attribute pairs with sim above a threshold are returned as correspondences
   - domain expert checks correspondence afterwards and selects the right ones

2. **TopK**
   - give domain expert TopK correspondences for each attribute

3. **Top1**
   - directly return the best match as correspondence
   - very optimistic, errors might frustrate domain expert
Alternative: Global Matching

- Looking at the complete mapping (all correct correspondences between A and B) gives us the additional restriction that one attribute in A should only be matched to one attribute in B.

- **Goal of Global Matching**
  - Find optimal set of disjunct correspondences
  - Avoid correspondence pairs of the form $A \equiv C$ and $B \equiv C$

- **Approach:**
  - Find set of bipartite pairs with the maximal sum of their similarity values

- **Example:**
  - $A \equiv D$ and $B \equiv C$ have the maximal sum of their similarity values
  - Ignores that $\text{sim}(A,C) = 1$
Elements of $S = \text{men}$, elements of $T = \text{women}$

$\text{sim}(i,j) =$ the degree to which $A_i$ and $B_j$ desire each other

Goal: Find a stable match combination between men and women

A match combination would be unstable if
- there are two couples $A_i = B_j$ and $A_k = B_l$ such that $A_i$ and $B_l$ want to be with each other, i.e., $\text{sim}(i, l) > \text{sim}(i, j)$ and $\text{sim}(i, l) > \text{sim}(k, l)$

Algorithm to find stable marriages
- Let match=$\{\}$
- Repeat
  - Let $(i,j)$ be the highest value in sim such that $A_i$ and $B_j$ are not in match
  - Add $A_i = B_j$ to match

Example: $A = C$ and $B = D$ form a stable marriage.
5.5. Summary

- Schema Matching is active research area with lots of approaches
  - Yearly competition: Ontology Alignment Evaluation Initiative (OAEI)

- Quality of the correspondences depends on difficulty of problem
  - Many approaches work fine for toy-problems, but fail for larger schemas
  - No commercial implementations of the methods

- Thus it is essential to keep the domain expert in the loop.
  - Active Learning
    - Learn from user feedback while searching for correspondences
  - Leveraging the Crowd
    - Mechanical Turk
    - Click log analysis of query results
    - DBpedia Mapping Wiki
  - Spread the manual integration effort over time
    - Pay-as-you-go integration in data spaces (see next slide)
The Dataspace Vision

Alternative to classic data integration systems in order to cope with growing number of data sources.

Properties of dataspaces

- may contain any kind of data (structured, semi-structured, unstructured)
- require no upfront investment into a global schema
- provide for data-coexistence
- give best effort answers to queries
- rely on pay-as-you-go data integration


6. Schema Heterogeneity on the Web

1. Role of Standards
   1. RDFa/Microdata/Microformats
   2. Linked Data

2. Self-Descriptive Data on the Web
6.1 Role of Standards

For publishing data on the Web, various communities try to avoid schema-level heterogeneity by agreeing on standard schemata (also called vocabularies or ontologies).

- **Schema.org**
  - 200+ Types: Event, organization, Person, Place, Product, Review, ...

- **Open Graph Protocol**
  - 25 Types: event, product, place, website, book, profile, article

- **Linked Data Context**
  - various widely used vocabularies.
  - FOAF, SKOS, Music Ontology, …
- Widely used Vocabularies
  - og: Open Graph Protocol
  - foaf: Friend of a Friend
  - sioc: Semantically interlinked Online Communities
  - dv: Google Data Vocabulary

<table>
<thead>
<tr>
<th>Class</th>
<th>PLDs Total #</th>
<th>PLDs in Alexa #</th>
<th>#</th>
<th>%</th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 og: &quot;article&quot;</td>
<td>183,046</td>
<td>17,002</td>
<td>35.24</td>
<td>10.37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 og: &quot;blog&quot;</td>
<td>58,971</td>
<td>5,820</td>
<td>11.35</td>
<td>10.37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 og: &quot;website&quot;</td>
<td>56,573</td>
<td>9,533</td>
<td>10.89</td>
<td>16.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 foaf:Image</td>
<td>44,644</td>
<td>2,794</td>
<td>8.60</td>
<td>4.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 sioc:Item</td>
<td>33,141</td>
<td>2,188</td>
<td>6.38</td>
<td>3.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 sioc:UserAccount</td>
<td>19,331</td>
<td>1,327</td>
<td>3.72</td>
<td>2.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 og: &quot;product&quot;</td>
<td>19,107</td>
<td>3,389</td>
<td>3.68</td>
<td>6.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 skos:Concept</td>
<td>13,477</td>
<td>1,135</td>
<td>2.59</td>
<td>2.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 dv:Breadcrumb</td>
<td>9,054</td>
<td>2,123</td>
<td>1.74</td>
<td>3.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 sioc:Post</td>
<td>6,994</td>
<td>691</td>
<td>1.35</td>
<td>1.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 og: &quot;company&quot;</td>
<td>6,758</td>
<td>1,067</td>
<td>1.30</td>
<td>1.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 dv:Review-aggregate</td>
<td>6,236</td>
<td>1,410</td>
<td>1.20</td>
<td>2.51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 dv:Rating</td>
<td>4,139</td>
<td>845</td>
<td>0.80</td>
<td>1.51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 sioc:BlogPost</td>
<td>3,936</td>
<td>308</td>
<td>0.76</td>
<td>0.55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 sioc:Comment</td>
<td>3,339</td>
<td>456</td>
<td>0.64</td>
<td>0.81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 og: &quot;activity&quot;</td>
<td>3,303</td>
<td>606</td>
<td>0.64</td>
<td>1.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 vcard:Address</td>
<td>3,167</td>
<td>401</td>
<td>0.61</td>
<td>0.71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 gr:BusinessEntity</td>
<td>3,155</td>
<td>392</td>
<td>0.61</td>
<td>0.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 dv:Organization</td>
<td>2,502</td>
<td>367</td>
<td>0.48</td>
<td>0.65</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Only two vocabularies are used!

1. schema = Schema.org

2. datavoc = Google‘s Rich Snippet Vocabulary (deprecated since 2011)
Class / Property Distribution

-> A small set of classes / properties is used.

-> Heterogeneity on schema level is manageable.
Vocabularies in the LOD Cloud

Data sources mix term from commonly used and proprietary vocabularies.

- Some terms from non-proprietary vocabularies: 191 (64.75 %) of the 295 sources
- Only proprietary vocabularies: 104 (35.25 %) of the 295 sources
- Common Vocabularies

<table>
<thead>
<tr>
<th>Vocab</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc</td>
<td>92</td>
<td>31.19 %</td>
</tr>
<tr>
<td>foaf</td>
<td>81</td>
<td>27.46 %</td>
</tr>
<tr>
<td>skos</td>
<td>58</td>
<td>19.66 %</td>
</tr>
<tr>
<td>geo</td>
<td>25</td>
<td>8.47 %</td>
</tr>
<tr>
<td>akt</td>
<td>17</td>
<td>5.76 %</td>
</tr>
<tr>
<td>bibo</td>
<td>14</td>
<td>4.75 %</td>
</tr>
<tr>
<td>mo</td>
<td>13</td>
<td>4.41 %</td>
</tr>
<tr>
<td>vcard</td>
<td>10</td>
<td>3.39 %</td>
</tr>
<tr>
<td>sioc</td>
<td>10</td>
<td>3.39 %</td>
</tr>
<tr>
<td>cc</td>
<td>8</td>
<td>2.71 %</td>
</tr>
</tbody>
</table>

Source: http://lod-cloud.net/state/
6.2 Self-Descriptive Data

Data sources in the LOD context try to increase the usefulness of their data and ease data integration by making it self-descriptive.

Aspects of self-descriptiveness

1. Enable clients to retrieve the schema
2. Reuse terms from common vocabularies / ontologies
3. Properly document your proprietary schema
4. Publish schema mappings for proprietary terms
5. Provide provenance metadata
6. Provide licensing metadata
Reuse Terms from Common Vocabularies

1. Common Vocabularies
   - **Friend-of-a-Friend** for describing people and their social network
   - **SIOC** for describing forums and blogs
   - **SKOS** for representing topic taxonomies
   - **Organization Ontology** for describing the structure of organizations
   - **GoodRelations** provides terms for describing products and business entities
   - **Music Ontology** for describing artists, albums, and performances
   - **Review Vocabulary** provides terms for representing reviews

2. Common sources of identifiers (URIs) for real world objects
   - **LinkedGeoData** and **Geonames** locations
   - **GeneID** and **UniProt** life science identifiers
   - **DBpedia** wide range of things
Enable Clients to retrieve the Schema

Clients can resolve the URIs that identify vocabulary terms in order to get their RDFS or OWL definitions.

Some data on the Web

```
<http://richard.cyganiak.de/foaf.rdf#cygri>
  foaf:name "Richard Cyganiak" ;
  rdf:type <http://xmlns.com/foaf/0.1/Person> .
```

Resolve unknown term

```
http://xmlns.com/foaf/0.1/Person
```

RDFS or OWL definition

```
<http://xmlns.com/foaf/0.1/Person>
  rdf:type owl:Class ;
  rdfs:label "Person";
  rdfs:subClassOf <http://xmlns.com/foaf/0.1/Agent> ;
  rdfs:subClassOf <http://xmlns.com/wordnet/1.6/Agent> .
```
The documentation of a vocabulary is published on the Web in machine-readable form and can be used as a clue for schema matching.

- **Name of a Vocabulary Term**
  - ex1:name rdfs:label "A person's name"@en .
  - ex2:hasName rdfs:label "The name of a person"@en .
  - ex2:hasName rdfs:label „Der Name einer Person"@de .

- **Additional Description of the Term**
  - ex1:name rdfs:comment "Usually the family name"@en .
  - ex2:name rdfs:comment "Usual order: family name, given name"@en .
Publish Correspondences on the Web

Vocabularies are (partly) connected via vocabulary links.

Vocabulary Link

<http://dbpedia.org/ontology/Person>
owl:equivalentClass
<http://xmlns.com/foaf/0.1/Person> .

- Terms for representing correspondences
  - owl:equivlentClass, owl:equivalentProperty,
  - rdfs:subClassOf, rdfs:subPropertyOf
  - skos:broadMatch, skos:narrowMatch
Deployment of Vocabulary Links

Vocabulary links:
Vocabularies referencing "foaf" (119)

Vocabularies referenced by "mo" (17)

Summary: Structuredness and Standard Conformance
7. References

- **Schema Integration**

- **Interpretation and Data Translation**

- **Schema Matching**
References

- Schema Matching (continued)

- Data Spaces

- Heterogeneity/Schema Standardization on the Web