Standing on the Shoulders of Ants: Towards More Efficient XML-to-Relational Mapping Strategies

Irena Mlynkova
irena.mlynkova@mff.cuni.cz

Charles University
Faculty of Mathematics and Physics
Department of Software Engineering
Prague, Czech Republic
Introduction

• XML = a standard for data representation and manipulation
  ⇒ A boom of implementations
    • XML file systems, native XML databases, XML-enabled databases, ...
• XML-enabled databases
  • Most practically used though less efficient than native XML databases
  • Exploitation of tools and functions of traditional (O)RDBMSs
    • Reliable and robust
    • Long theoretical and practical history
  • Major DB vendors support XML
  • SQL standard: new part SQL/XML
DB-Based XML Processing Methods

• Key concern: Choice of the most efficient XML-to-relational mapping strategy
• Various classifications:
  • Generic (schema-oblivious) vs. schema-driven – omitting vs. exploiting XML schema
  • Fixed vs. adaptive – mapping on the basis of data model vs. target application
    • Sample data and queries
  • User-defined vs. user-driven – the amount of user involvement
    • User specifies target schema and required mapping vs. user locally modifies a default mapping
⇒ The most promising approach: adaptive
• Evaluates several mappings and chooses the optimal one
Goal of This Paper

Three improvements of adaptive strategies:

1. Improvement of searching the optimal strategy
   - Ant Colony Optimization
     - Finds better suboptimal solution than simple greedy search strategies currently used in the existing papers

2. Enhancing of the adaptation process with similarity of XML data
   - Classical improvement

3. Combination with user-driven techniques
   - User provides information on required mapping strategies for selected data fragments
Problem Statement

• To find the optimal mapping strategy for given XML schema $S_{\text{init}}$ into a set of relations $R = \{r_1, r_2, ..., r_m\}$

• Cost-driven adaptive strategies exploit:
  • Set of XML-to-XML schema transformations $T = \{t_1, t_2, ..., t_n\}$
  • $\forall i: t_i$ transforms schema $S$ to schema $S_i = t_i(S)$
  • Set of sample data $D_{\text{sample}}$ characterizing an application
    • XML documents $D = \{d_1, d_2, ..., d_k\}$ valid against $S_{\text{init}}$
    • XML queries $Q = \{q_1, q_2, ..., q_l\}$ over $S_{\text{init}}$
  • Cost function $f_{\text{cost}}$
    • Evaluates the cost of relational schema $R$ with regard to $D_{\text{sample}}$

• Aim: optimal relational schema $R_{\text{opt}}$
  • $f_{\text{cost}}(R_{\text{opt}}, D_{\text{sample}})$ is minimal
Example

- $T = \{t_{in}, t_{out}\}, Q = \emptyset$
  - Inlining, outlining

employee_1(name_first, name_middle, name_last, address_city, address_country, address_zip)

- $T = \{t_{in}, t_{out}, t_{shred}, t_{unshred}\}$
  - Inlining, outlining, shredding, unshredding
- $Q = \{\ldots, //employee/name, \ldots\}$
  - No query of the form //employee/name/first, //employee/name/middle or //employee/name/last

employee_2(name, address_city, address_country, address_zip)

<!ELEMENT employee (name, address)>
<!ELEMENT name (first, middle?, last)>
<!ELEMENT address (city, country, zip)>
<!ELEMENT first #PCDATA>

...
Improvement 1. Search Strategy

- Naïve search strategy:
  - To generate all possible transformations of $S_{init}$ and select the optimal one
- Problem: Searching for $R_{opt}$ is an NP hard problem
  - Constraints optimization problem (COP)

$\Rightarrow$ Current approaches use heuristics $\Rightarrow$ search for suboptimum
  - Typically a kind of a greedy search strategy
  - Get stuck in local suboptimums

- Our approach: Ant Colony Optimization (ACO)
ACO

- Idea: Artificial ants *iteratively* search a space of solutions and improve current suboptimum

- **Ant**
  - Searches a subspace of solutions until it “dies”
  - Spreads “pheromone”
    - Positive feedback = how good solution it has found so far
  - Exploits pheromones of other ants to select next step
    - Step = applying an XML-to-XML schema transformation
    - Selected *randomly*
      - Probability is given by $f_{cost}$ and pheromones of other ants

- In fact: Simple application of a general heuristic on a special case
Improvement 2. Fragments without Cost Feedback

- Aim of cost-driven methods: to find relational schema R optimal for queries in Q
- Problem: $S_{init}$ may involve fragments which occur on no access path of queries in Q
- Motivation for solution: system UserMap
  - User-driven mapping
  - Schema annotations are directly applied + regarded as “hints” how to store XML patterns
  - Idea: Iteratively searches for schema fragments similar to patterns and maps them in the same way
- Modification: annotated schema fragments = schema fragments on access paths of Q
Example

- \( T = \{ t_{in}, t_{out}, t_{shred}, t_{unshred} \} \)
- \( Q = \{ //employee/name \} \)
- Element company cannot be mapped adaptively ⇐ no feedback in Q

\[
\text{company}_1(\text{co-name_title}, \text{co-name_type}, \\
\text{address_city}, \text{address_country}, \text{address_zip})
\]

- Elements name and co-name are semantically similar \( \Rightarrow \) can be mapped in a similar way

\[
\text{company}_2(\text{co-name}, \\
\text{address_city}, \text{address_country}, \text{address_zip})
\]
Improvement 3. Schema Annotations

- Idea: Let us go even further…
- Motivation:
  - Simple schema fragments: D and Q
    - e.g. the previous examples
  - Complex ones: mapping strategy
    - e.g. unshredding for XHTML fragments
- Observation: Sequence of transformations form $T = \text{complex storage strategy} = \text{annotation}$
  - Combination of cost-driven and user-driven approaches
⇒ The set of possible steps of ants: application of $T + \text{composite transformations} = \text{user-specified annotations}$
- Can be applied on schema fragments similar to the original annotated ones
⇒ Speeds-up the search process
Example

• Relation employee_2 can be derived using various sequences of transformations

  employee_2(name,
              address_city, address_country, address_zip)

  \[s_1 = [t_{\text{in}}(\text{city}), t_{\text{in}}(\text{country}), t_{\text{in}}(\text{zip}), t_{\text{in}}(\text{address}), t_{\text{in}}(\text{first}), t_{\text{in}}(\text{middle}), t_{\text{in}}(\text{last}), t_{\text{in}}(\text{name}), t_{\text{un}}(\text{name})]\]

  \[s_2 = [t_{\text{un}}(\text{name}), t_{\text{in}}(\text{name}), t_{\text{in}}(\text{city}), t_{\text{in}}(\text{country}), t_{\text{in}}(\text{zip}), t_{\text{in}}(\text{address})]\]

• If a user annotates element employee with fixed hybrid mapping (= employee_1)

  employee_1(name_first, name_middle, name_last,
              address_city, address_country, address_zip)

  \[s_3 = [t_{\text{un}}(\text{name})]\]
System Architecture

- Improvements could be applied on any user-driven/cost-driven system
- We enhance UserMap (user-driven ⇒ we add cost-driven features)

XML-to-relational mapping

New modules

Annotation processor

Cost estimator

ACO adaptation

UserMap adaptation

Mapper

Mapping repository

Data repository

Document shredder

Query evaluator

CREATE TABLE

SELECT

tuples

tuples

queries

responses

documents

Similarity evaluator
Conclusion

• Current work:
  • Throughout implementation of the proposals
    • Enhancing of UserMap
    • Key aspect: cost estimator (evaluation of $f_{\text{cost}}$)
  • Application on real-world data
  ⇒ What is the impact of the improvements?

• Main advantages of improvements:
  • Avoid getting stuck in local suboptimal solutions
  • Find an optimal mapping for greater subset of source schema
  • Speed up the search process

• Future work: Persisting disadvantages of adaptive approaches
  • Plenty of information on application
    • User-unfriendly
  • Efficiency can worsen with minor changes in the application
Thank you