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

(O)RDBMS

- 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_{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_{sample} characterizing an application
    - XML documents $D = \{d_1, d_2, ..., d_k\}$ valid against S_{init}
    - XML queries $Q = \{q_1, q_2, .., q_l\}$ over S_{init}
  - Cost function f_{cost}
    - Evaluates the cost of relational schema R with regard to D_{sample}
- Aim: optimal relational schema R_{opt}
  - $f_{cost}(R_{opt}, D_{sample})$ is minimal

Example

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

- $T = \{t_{in}, t_{out}\}$, $Q = \emptyset$
  - 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 = {…, //employee/name, …}
  - No query of the form //employee/name/first, //employee/name/middle or //employee/name/last

```
employee_2(name,
           address_city, address_country, address_zip)
```

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)

⇒ Current approaches use heuristics ⇒ 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 <u>iteratively</u> 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 <u>randomly</u>
      - 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

<!ELEMENT employee (name, address)>
<!ELEMENT name (first, middle?, last)>
<!ELEMENT address (city, country, zip)>
<!ELEMENT company (co-name, address)>
<!ELEMENT co-name (title, type?)>

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

company_1(co-name_title, co-name_type,
address_city, address_country, address_zip)

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

company_2(co-name,
address_city, address_country, 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 = complex storage strategy = annotation
  - Combination of cost-driven and user-driven approaches

⇒ The set of possible steps of ants: application of T + composite transformations = user-specified annotations

- Can be applied on schema fragments similar to the original annotated ones

⇒ Speeds-up the search process

Example

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

- Relation employee_2 can be derived using various sequences of transformations

employee_2(name,
address_city, address_country, address_zip)

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

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

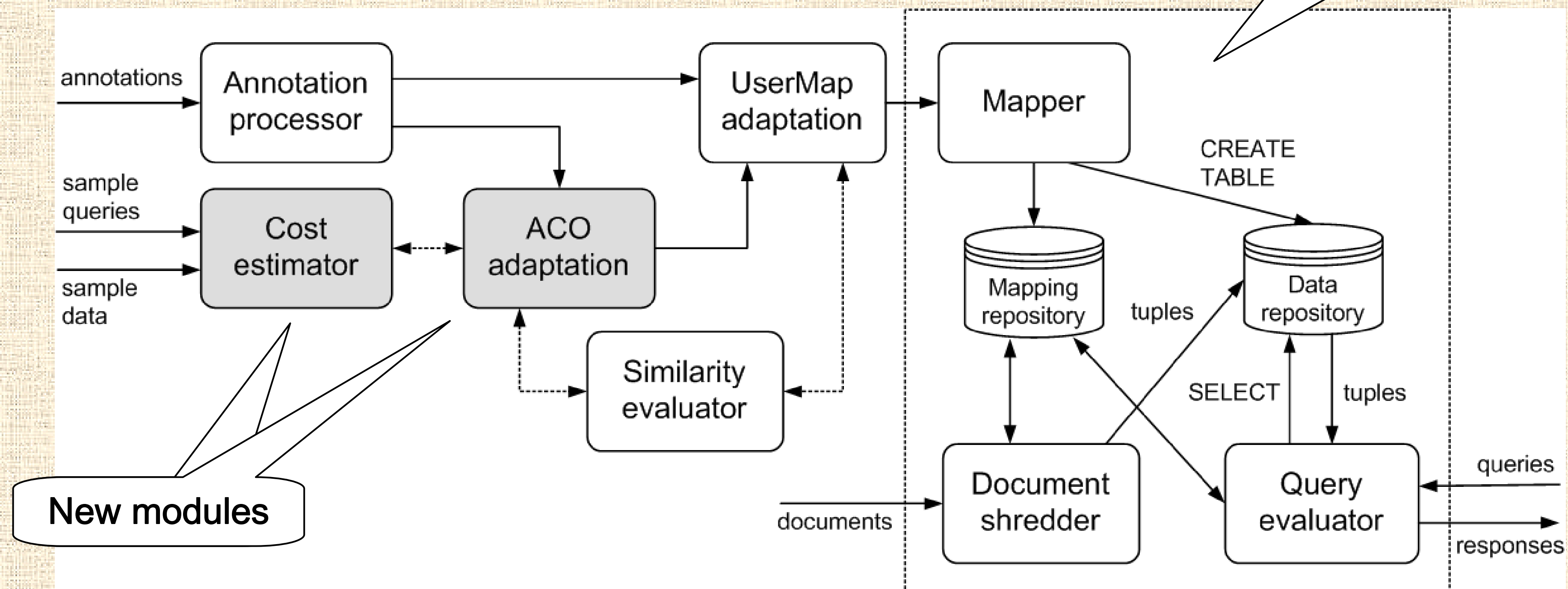
- If a user annotates element employee with fixed hybrid mapping (= employee_1)

s_3 = [t_{un}(name)]

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

System Architecture

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



Conclusion

- Current work:
  - Throughout implementation of the proposals
    - Enhancing of UserMap
    - Key aspect: cost estimator (evaluation of f_{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