XML in the World of (Object-)Relational Database Systems

Irena Mlynkova, Jaroslav Pokorny

{mlynkova,pokorny}@ksi.ms.mff.cuni.cz

Charles University

Faculty of Mathematics and Physics

Department of Software Engineering

Prague, Czech Republic

Introduction

- XML = a standard for representation and interchange of information
  - Growing usage of XML technologies
  - Growing demand for effective management of XML documents + querying XML data

⇒ Possibility: Storing and managing XML data using (O)RDBS

⇒ Advantage: To provide XML with missing database mechanisms (i.e. indexes, multi-user access, etc.)

Goals of This Presentation

Overview of mapping methods between XML documents and (O)R structures

- Description, classification and discussion
- Own schema-driven method

Content

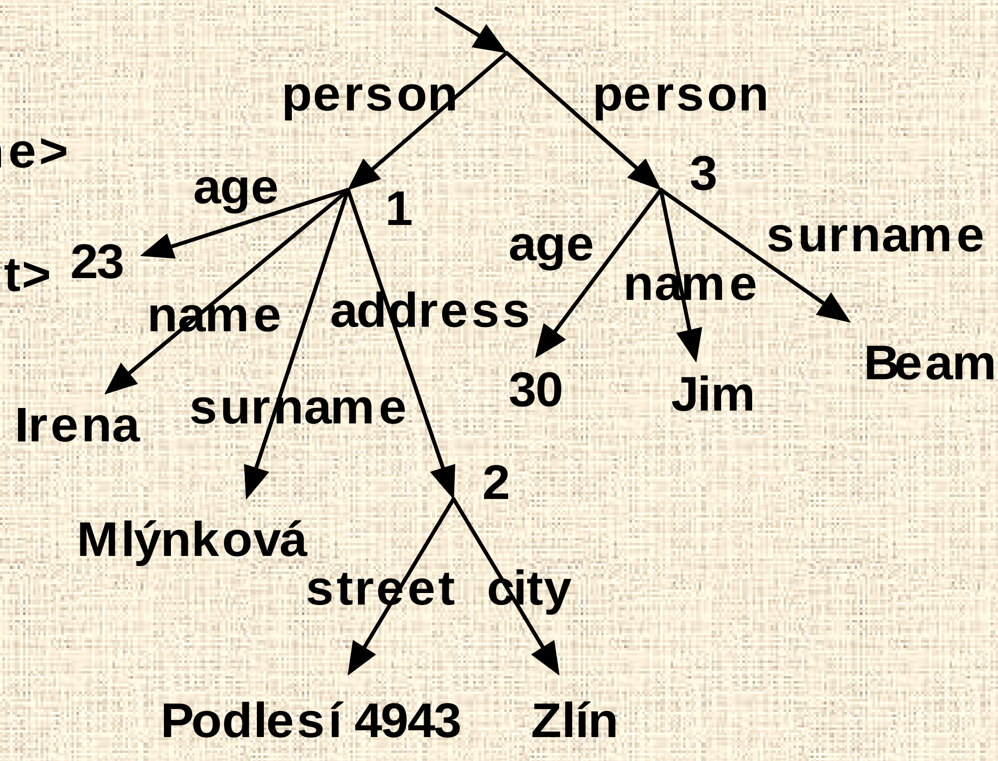
1. Generic methods
2. Schema-driven methods
3. User-defined methods
4. Conclusion

Generic Methods

- Do not use (possibly) existing XML schema
- Two approaches:
  - To create a general (O)R schema for any XML document regardless its structure
    - Views XML document as a tree
    - Problem: How to store the tree
  - To create a special (O)R schema for only a certain collection of XML documents
    - e.g. Table-based mapping

Generic-Tree Mapping (1)

```
<person id=1 age=23>
   <name>Irena</name>
   <surname>Mlýnková</surname>
   <address id=2>
      <street>Podlesí 4943</street>
      <city>Zlín</city>
   </address>
</person>
<person id=3 age=30>
   <name>Jim</name>
   <surname>Beam</surname>
</person>
...
```



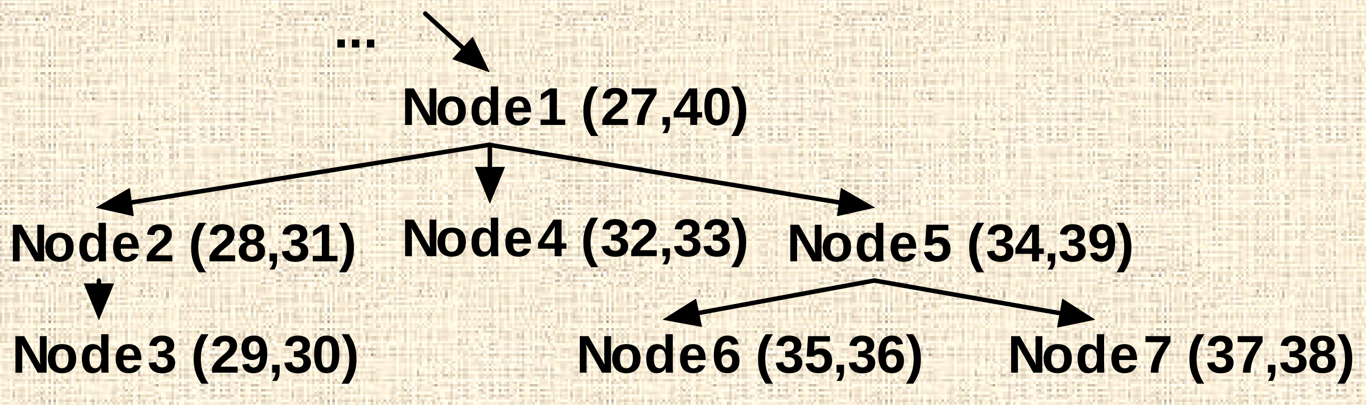
Generic-Tree Mapping (2)

- **Edge mapping:**
  - **Edge(source, ord, name, flag, target)**
    - **flag = whether the edge is internal or points to a leaf**
    - **ord = an ordinal number of the edge within sibling edges**
- **Attribute mapping**
  - **$\text{Edge}_{\text{name}}$(source, ord, flag, target)**
- **Universal mapping**
  - **The result of an outer join of all attribute tables**
- **Various combinations of previous cases**

Structure-Centred Mapping (1)

- Structure of all nodes: $v = (t, l, c, n)$
  - t = the node type (i.e. element, attribute, etc.)
  - l = the node label (i.e. name)
  - c = the node content (e.g. attribute values, etc.)
  - $n = \{v_1,\ldots,v_n\}$ = the list of successor nodes
- Problem: How to realize mapping of the lists of successor nodes:
  - Primary and foreign keys
  - DF values
  - SICF values

Structure-Centred Mapping (2)



- **A couple of min. and max. DF value**
  - **Time of visiting and leaving the node when traversing the tree in a depth-first (DF) manner**
  - **Determine relationships between the nodes**
    - **E. g. *v* is a descendant of *u*, if $v_{min}>u_{min}$ & $v_{max}<u_{max}$**

Other Methods

- Similar to previous cases + storing additional information
  - ⇒ Speeding up in special cases (e.g. path queries)
- *Simple-path mapping*
  - Each node retains (ID of) its simple path
  - More space needed
- *Monet mapping*
  - Nodes having the same (simple) path are stored into same table
  - Higher degree of fragmentation

Content

1. Generic methods
2. Schema-driven methods
3. User-defined methods
4. Conclusion

Schema-Driven Methods (1)

- Based on existing XML schema S_1
  1. S_1 is mapped to (O)R database schema S_2
  2. XML documents valid against S_1 are stored into relations of S_2
- The aim: To create an optimal S_2
  - With "reasonable" amount of relations
  - With structure corresponding to S_1
- Methods = improvements of the basic idea:

"Create one relation for each element composed of its attributes and map element-subelement relationships using keys and foreign keys."

Schema-Driven Methods (2)

- **Common basic principles and features:**
  - **Subelements with maxOccurs = 1 are mapped to tables of parent elements (so-called inlining).**
  - **Elements with maxOccurs > 1 are mapped to separate tables, element-subelement relationships are mapped using keys and foreign keys.**
  - **Alternative subelements are mapped to separate tables or to one universal table (with many nullable fields).**
  - **If it is necessary to preserve *the order of sibling elements*, the information is mapped to a special column.**
  - **Elements with *mixed content* are usually not supported.**
  - **A reconstruction of an element requires joining several tables.**

Schema-Driven Methods (3)

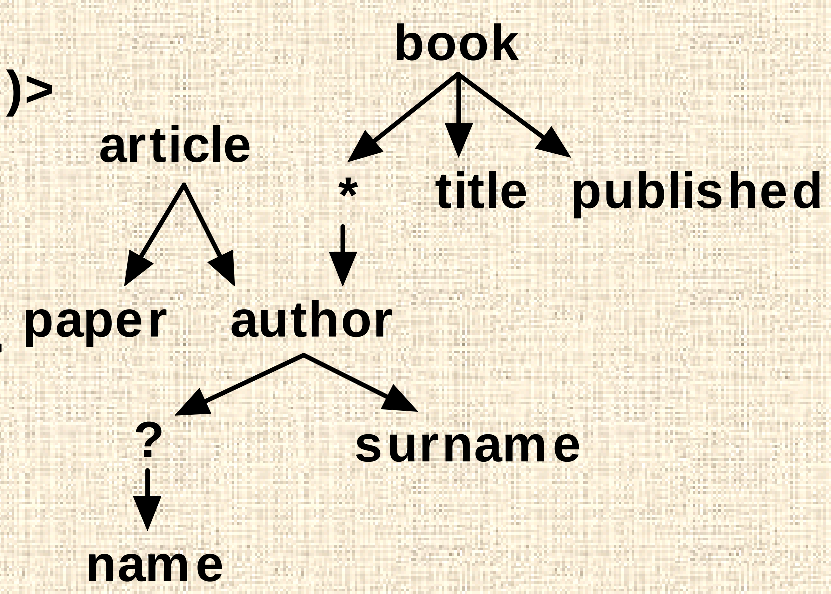
- **Classification:**
  - **S_1 – DTD vs. XML Schema**
  - **S_2 – (in this case) relational vs. object-relational**
  - **Flexibility:**
    - **Fixed methods = do not use other information than the source schema**
    - **Flexible methods = use additional information (query or element statistics, etc.) => an optimal schema for a certain application**
- **Usually based on some kind of auxiliary graph of S_1**

Basic, Shared, Hybrid (1)

- Based on a directed DTD graph
  - Nodes: elements, attributes, operators
  - Edges: element-attribute, element-subelement, element-operator relationships
- 3 algorithms = 3 improvements of the idea "to create one relation for each element"
- *Constraints-preserving inlining algorithm*
  - Based on Hybrid algorithm
  - Aim: to capture also the semantic constraints
    - SQL integrity constraints (e.g. UNIQUE, CHECK, etc.)

Basic, Shared, Hybrid (2)

<!ELEMENT author(name?,surname)>
<!ELEMENT name(#PCDATA)>
<!ELEMENT surname(#PCDATA)>
<!ELEMENT book(author*,title)>
<!ATTLIST book published CDATA>
<!ELEMENT title(#PCDATA)>
<!ELEMENT article(author)>
<!ATTLIST article paper CDATA>

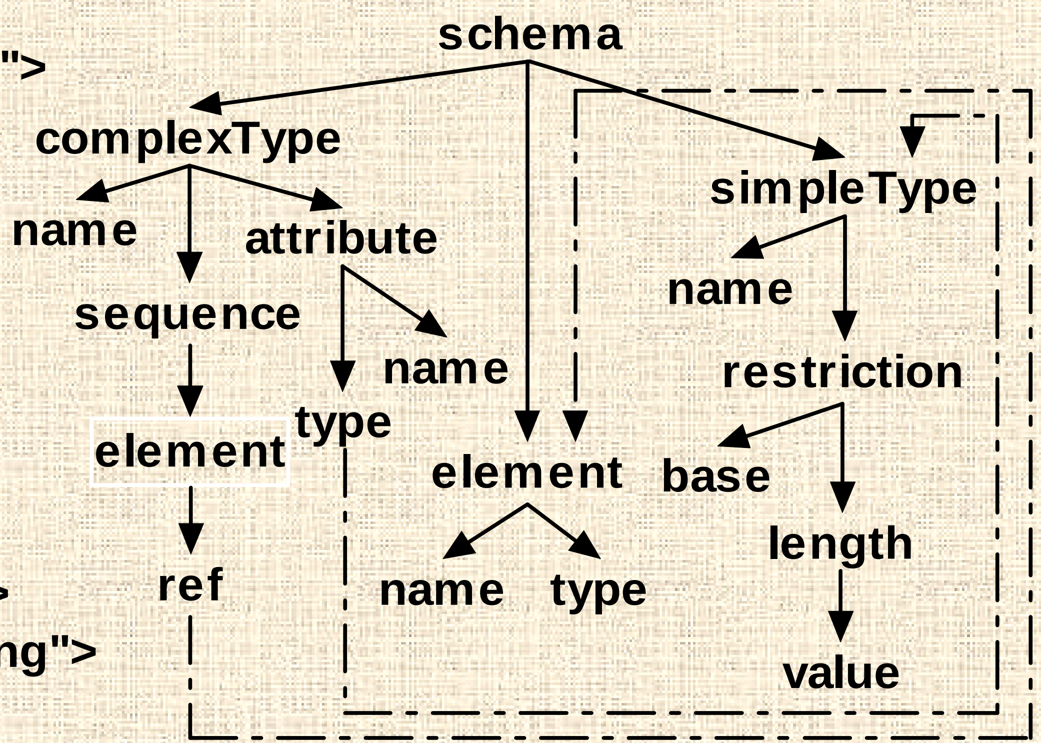


XMLSchemaStore Mapping (1)

- Based on a (directed) DOM graph
  - Nodes: XML Schema elements and attributes
  - Edges:
    - Element-subelement or element-attribute relationships
    - The “direction” of the usage of globally defined items
- Focus on:
  - OO features of XML Schema language
  - OR features of SQL:1999 (UDTs, typed tables, references, etc.)
    - S_2 = a set of typed tables “connected” using references

XMLSchemaStore Mapping (2)

```
<schema>
  <complexType name="T1">
    <sequence>
      <element ref="E1"/>
    </sequence>
    <attribute name="A1"
               type="T2"/>
  </complexType>
  <element name="E1"
           type="string">
  <simpleType name="T2">
    <restriction base="string">
      <length value="5"/>
    </restriction>
  </simpleType>
</schema>
```



Other Fixed Methods

- ***Object-relational mapping***
  - **Auxiliary graph = a tree of objects (classes) expressed in any object-oriented language**
    - **Elements ~ classes**
    - **Attributes ~ class properties**
- ***Constraints preserving mapping***
  - **Auxiliary graph = EER schema (extended entity-relationship model)**
    - **Elements ~ entities**
    - **Attributes ~ attributes of the entities**

LegoDB Mapping

- **Flexibility:**
  - **To explore a space of possible mappings**
  - **To select the best according to given statistics:**
    1. **Any possible XML-to-XML transformation is applied to S_1 resulting in S_T**
       - **E.g. inlining, outlining, union factorization, etc.**
    2. **XML-to-relational transformations (i.e. a fixed mapping) are applied to S_T resulting in S_2**
    3. **Against S_2 the given queries are estimated**
- **Infinite space => greedy evaluation strategy**

Hybrid Object-Relational Mapping

- **Two kinds of mapping:**
  - **Structured parts → relations**
  - **Semistructured parts → XML data type**
    - **Support of XML path queries and XML fulltext queries**
- **Flexibility: To determine (semi)structured parts**
  1. **A measure of significance is enumerated for each node**
     - **Based on DTD structure, existing XML data and queries**
  2. **All subgraphs with nodes below the given limit are considered as semistructured**

Content

1. Generic methods
2. Schema-driven methods
3. **User-defined methods**
4. Conclusion

User-defined methods

- Used in commercial systems
- Basic idea:
  1. User defines S_2
  2. User expresses required mapping using a system-dependent mechanism (e.g. a special query language, a declarative interface, etc.)
- Most flexible techniques
- Require large development effort + mastering of two distinct and complex technologies

Content

1. Generic methods
2. Schema-driven methods
3. User-defined methods
4. **Conclusion**

Conclusion

- Usability of methods depends on categories of XML documents (data/document-centric)
  - Focus on data-centric XML documents
  - Document-centric extensions
- Possible future focus:
  - The semantic constraints (XML Schema)
    - Transformation to SQL integrity constraints
  - Optimalizations
    - First attempts: flexible methods
    - No rules for a definition of a "good" XML schema (such as e.g. normal forms for relations)