A Framework for Generic Integration of XML Sources

Wolfgang May
Institut für Informatik
Universität Freiburg
Germany
may@informatik.uni-freiburg.de

KRDB Workshop, Rome
15.9.2001
OVERVIEW

- Integration operations (Warehouse strategy)
- Integration strategies
- Language: XPathLog
- Implementation: LoPiX
- Conclusion
TOPICS OVERVIEW

- Considerations on a Data Model for XML with updates/integration:
  FMLDO/FMII’01
  independent from the programming language

- XPathLog as an XML Database Programming Language: DBPL’01

- Implementation: LoPiX
  VLDB Demonstration Track

Application for Data Integration

- objects of different sources represent the same real-world object
  ⇒ Fusing objects, merging their properties

- different names and structure
  ⇒ Result views as projections

- Strategies for “intelligent” data-driven integration
  KRDB’01
Generic Integration of XML Sources

**Document vs. Database**

- integration of documents: tree, ordered
  “integration follows structure”
- integration of databases: graph, non-ordered
  semantics-driven integration process

**Database Integration**

- objects of different sources represent the same real-world object

  ⇒ Fusing objects, merging their properties
- synonyms, ontologies
- not compatible with XML Data Model (DOM, XML Query Data Model)
- requires a powerful language
- experiences with F-Logic for semi-structured data and data integration
SCENARIO

- Different autonomous sources, describing the same application area
  e.g., catalogs of Digital Photography
- Overlapping, but potentially incomplete and inconsistent
- No fixed integration mapping known
  ⇒ Data-Driven integration strategies
- Stepwise generation of an integrated database
- Warehouse vs. virtual approach
<producer name="Nikon">

<product type="digital camera" name="Coolpix880">
  mpix="3.34" price="1799.00">
  <zoom>
    <external focallength="8 20">
      <digital factor="4" />
    </external>
  </zoom>
  <accessory type="lens" name="WC-E24" />
  <accessory type="lens" name="TC-E2" />
  <accessory type="lens" name="TC-E3" />
</product>

<product type="wide angle adapter" name="WC-E24">
  factor="0.66" price="219.00">
</product>

<product type="teleconverter" name="TC-E2">
  factor="2" price="259.00">
</product>

<product type="teleconverter" name="TC-E3">
  factor="3" price="589.00">
</product>

:</producer>
<store name="shop1">
  <digitalcamera producer="Nikon"
      type="Coolpix880" price="1699.00"/>
  <digitalcamera producer="Nikon"
      type="Coolpix990" price="2399.00"/>
  <digitalaccessory producer="Nikon"
      type="WC-E24" price="199.00"/>
  <digitalaccessory producer="Nikon"
      type="TC-E2" price="269.00"/>
  <digitalcamera producer="Olympus"
      type="C3000" price="1599.00"/>
  ...  
</store>

- other resellers pages
- test reports etc
Generic Integration of XML Sources

Integration: “Three-level” Model

access multiple sources

- “basic” layer: source(s) provide tree structures,
- optionally with namespaces
  - *nikon*: producer’s tree
  - *shop1*, *shop2* etc: resellers trees

merge data from different sources

Abstract Operations

- fuse elements/merge subtrees
- introduce synonyms for properties
- connect elements and tree fragments from several sources by links
- generate elements

“internal” data model: XTreeGraph

- overlapping trees
- multiple parents
- references

“export” layer: result trees views defined as projections
**INTEGRATION: FUSING ELEMENTS AND SUBTREES**

**Situation**

- elements represent the same real-world entity in different sources
- fuse elements into a unified element: $e_1 = e_2$

**Resulting element**

1. globally replace $e_2$ in all properties by $e_1$.
2. $e_1$ is then an element of both source trees, i.e., positive queries against the original tree using the original namespace still yield at least the original answers,
3. $e_1$ collects the attributes of both original elements.
4. $e_1$ collects the subelements of both original elements.
SYNONYMS

• identify properties with the same semantics

\[ name_1 = name_2 \]

• take properties from the (namespaced) sources are completely added to the result – with another name

\[ namespace:name_1 = name_2 \]

• does not introduce new children or attribute nodes,
• “only” defines an alternative navigation path,
• does not change order of children
RESULT VIEWS

Projection by Signatures

Given:

- Database with (overlapping) trees
- signature specification (derivable from DTD or XML Schema),
- a root node \( r \).

Tree view rooted in \( r \):

- the root node \( r \),
- attributes and subelements (recursively) filtered according to the signature

Integrity Constraints:

- result may contain dangling references.
- result may be cyclic/infinite.
STRATEGIES

- use a reference tree
- describe keys (names, codes, titles etc.)
- Identify corresponding concepts and elements in different trees
  (element types, attribute names etc.)
- candidate sets of corresponding elements
  products in the nikon tree and in the reseller’s trees
- verify correspondence and fuse elements
- identify corresponding properties by values based on “known” identical elements
  – properties which have to be identified
  – properties which correspond, but have to be compared
  – technical data vs. prices of different resellers
- generalize to all elements of a given type
  from nikon products to olympus products in the reseller tree
- analogies: semantics of related element types
- detect mappings between properties (price in DM, Dollars; lengths in cm, inch)
- generalize relationships between sources
Strategies, Details, Extensions

- Well-founded semantics for detecting sets of corresponding elements in “graph” databases interfering, negative dependencies between candidate sets, deep-equality etc.

- Statistical methods/Data Mining for handling inconsistencies property coincides for 95% of all identified objects

- perhaps consider another input database for these data confidence measures

- include meta-knowledge
  - ontologies
  - domain-dependent knowledge (units, currencies, taxes, languages)

⇒ requires a powerful language
WAREHOUSE VS. VIRTUAL REVISITED

- Apply strategies to an excerpt of the database in the warehouse approach
- Derive a mapping
- Apply mapping to complete databases in a virtual integration strategy
LANGUAGE PROPOSAL: XPathLog

Design Decisions

- experiences with F-Logic for semi-structured data and data integration
- declarative rule-based language with bottom-up semantics
- extend XPath with variable-bindings
- XTreeGraph data model
- support for 3-level integration approach
XPATHLog by Examples

Pure XPath expressions

?- //producer[@name = "Nikon"]
   //product[@name="Coolpix 880”]/@mpix.
true

Output Result Set

?- //producer[...]/product[...]/@mpix→M.
M/3.34

Additional Variables

?- //producer[...]/product[@name=N]/@price→P.
   N/“Coolpix 880"   P/1799.00
   N/“WC-E24”       P/219.00
   :

Dereferencing

?- //producer[...]/product/accessory/@name/@price→P.

Schema Querying

?- //product[@type="camera”]/@Prop.
   Prop/name
   Prop/mpix
   Prop/price
**XPathLog Rules**

\[
\text{head}(V_1, \ldots, V_n) \leftarrow \text{body}(V_1, \ldots, V_n)
\]

Constructive semantics of XPath expressions

- **Definite** XPathLog atoms:
  - use only the child, sibling, and attribute axes
  - no negation, function applications, aggregation, and *proximity position predicates*

“/” and “[…]” act as **constructors**:

- **host[property→value]** modifies **host**
- **property** of the form
  - child::name
  - child(i)::name
  - preceding/following-sibling::name
  - preceding/following-sibling(i)::name
  - attribute::name

⇒ **unambiguous insertions**
Generic Integration of XML Sources

**IMPLEMENTATION: LoPiX**

- developed using major components from FLORID
**CONCLUSION**

- specialized integration operations for XML data: fusing, linking, synonyms
- not compatible with DOM/XML Query Data Model: unique-parent
- graph data model suitable & necessary for updates and integration
- 3-level integration process
- manually written integration programs vs. high-level, generic, heuristics-based strategies