A Logic-Based Approach to XML Data Integration with “lazy materialization”

Wolfgang May
Universität Göttingen

Workshop “Logic-based Methods for Information Integration”
Vienna, August 23, 2003
Project Background: F-Logic
- Experiences in deductive, object-oriented database languages
- early semi-structured/self-describing data model
- integration of semi-structured data in F-Logic/FLORID

XML: Internet-wide data format, distributed, autonomous sources

XPathLog: Logic Programming style language (2001)
Implementation: LoPiX (Logic Programming in XML)

materialization vs. virtual approach?
intermediate solution: partly materialized + linked
Example: Mondial

<mondial>
  <country car_code="B"
    capital="cty-Brussels"
    memberships="org-eu org-nato ..">
    <name>Belgium</name>
    <population>
      10170241
    </population>
    <city id="cty-Brussels"
      country="B">
      <name>Brussels</name>
      <population year="95">
        951580
      </population>
    </city>
  </country>
  <organization id="org-eu"
    headq="cty-Brussels">
    <name>Europ. Union</name>
    <abbrev>EU</abbrev>
    <members type="member"
      country="GR F E A D I B L . . . "/>
    <members type="applicant"
      country="AL CZ . . . ">
    </organization>
  <organization id="org-nato" headq="cty-Brussels">
  </organization>
</mondial>
Data Integration with XPathLog

XPathLog (VLDB/DBPL/FMLDO 2001):
Logic Programming style XML query/integration language
  * Datalog-style extension of XPath
    * XPathLog is the Horn fragment of XPath-Logic
    * Syntax and Semantics of queries extend XPath with variable bindings

?- //country[name/text()→N1 and @car_code→C]//city/name/text()→N2.
N1/“Belgium”  C/“B”  N2/“Brussels”
N1/“Belgium”  C/“B”  N2/“Antwerp”

:
Rules

Logic Programming language:

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

Rule Heads

Constructive semantics of definite XPathLog expressions for generating data and structure:

- only *child*, *sibling* and *attribute*-axes

“/” and “[...]” as Constructors:

- host[property→value] modifies host
- host/property remainder
  generates a new subelement *property* to *host* that satisfies remainder (possibly complex structure)
Example: Generating Structures

- complex expressions in the head:

```xml
/country[@car_code=“BAV” and name[text()=“Bavaria”]].

  <country car_code=“BAV”>
    <name>Bavaria</name>
  </country>
```

- complex expressions are decomposed into atoms (introduce new **local** variables)

```xml
//country→_C,
_C[@car_code=“BAV”], _C[name→_N],
_N[text()=“Bavaria”].
```
Data Integration

Requires operations that are not compatible with the XML model:

- add new subelement relationships
- “fuse” elements from different sources
- define synonyms for names
Adding Subelement Relationships

- Elements have multiple parents
- Efficient *in-place* restructuring and integration
Data Sources describing countries:
- cia: name, area, population and capital (by name)
- gs: cities with name, population
Synonyms

- some properties of the namespaced sources are taken to the result completely – with another name
  
  cia:name = name.  
  cia:area = area.  
  cia:population = population.  
  cia:text() = text().

- does not generate new element or attribute nodes,
- but “only” additional navigation paths
Element Fusion

- Elements represent the same real-world entity in different sources
- Fuse elements into a unified element:

\[ e_1 = e_2 \]

\[
\text{result[country} \rightarrow \text{C1], C1 = C2 :}
\]
\[
\text{cia/cia:country} \rightarrow \text{C1[@cia:name} \rightarrow \text{N]},
\]
\[
\text{gs/gs:country} \rightarrow \text{C2[@gs:name} \rightarrow \text{N}].
\]
C[@capital→City] :-
result/country→C[@cia:capital→Name and
city→City[name/text()=Name]].
Integration by Materialization

“Three-level”-Model

- **“basic” layer**: source(s) provide tree structures, optionally with namespaces
- **“internal” layer**: “XTreeGraph”
  - overlapping trees, multiple parents
  - add subelement links
  - fuse elements/merge subtrees
  - synonyms for properties
- **“export” layer**: result trees as views/projections:
  - a root node
  - signature (derivable from DTD or XML Schema)
Advantages

- Logic/rule-based syntax & semantics
  “what you write is what you get”
- Declarative semantics for generating XML with XPath
  - Expressive and flexible language
  - Queries against metadata/schema
  - Integration operators: element fusion and synonyms
- Concise, intuitive, executable specification = program

Current implementation: LoPiX

(Logic Programming in XML)

- Bottom-up stratified semantics
  Operational executable semantics/specification of integration process;
- Computes/materializes integrated database
Strategy: Selective “Lazy” Materialization

Combination with “Virtual”-Strategy
- do not materialize whole database,
- use subtrees of the original sources

```xml
<country car_code="D">
  name: reference to name element in cia:Germany
  area: reference to population element in cia:Germany
  capital: reference to cia:Berlin
  city-subelements: references to city subelements in gs:Germany
</country>
```

- less space-consuming
- incorporates data changes in autonomous sources
Partially Virtual Strategy

- representation in the database:
  - references into XML sources expressed by XPointer
  - integrated by XLinks,
  - interpreted as embedded view fragments
- actual “load” only when actually changed
  - materialize minimal fragment
  - represent unchanged parts by links
- do not change the language:
  “integrate” XLinks into data model and query language
Simple Links

- similar to the HTML `<A href="...">` construct.

Headquarters of organizations:

```xml
<!ELEMENT organization (... headq ... )>
<!ELEMENT headq EMPTY>
<!ATTLIST headq xlink:type (simple|extended|locator|arc) #FIXED "simple"
    xlink:href CDATA #REQUIRED >

<organization abbrev="EU">
    <headq xlink:href="file:cities-B.xml#/city[name='Brussels']/">
    ...
</organization>
```

query:

```
//organization[@abbrev="EU"]/headq/*
```
abstract data model makes the links transparent:
(WWW 2002, BTW 2003)

each link can be seen as a view definition
  integrates external schemata
  embedded views implicitly accessible as subtrees or by reference attributes
  (specified by attributes to the link element)
  queried by standard XPath/XPathLog expressions
regard link elements to be *transparent*

\[\text{query:} \quad \text{//organization[@abbrev="EU"]/headq/population}\]
Integration Strategy

- if a subtree is “used”, represent it as a link in the partially integrated internal database
- rule bodies: implicitly resolve the link
- rule heads/view updates: access view, and load as far as necessary into the internal database
- use of links also inside the internal database makes it possible to store data as an XML tree
- logical link semantics then simulates the XTreeGraph
- implementable based on standard XML software (extend XPath interpretation by resolving XLinks)
Questions ??