On an XML Data Model for Data Integration

Wolfgang May, Erik Behrends
Institut für Informatik
Universität Freiburg
Germany
{may|behrends}@informatik.uni-freiburg.de

FMII Workshop
Viterbo, 17.9.2001

TOPICS OVERVIEW

- Considerations on the Data Model: FMLDO/FMII '01
  independent from the programming language

  Updates

  – Problems with XML Data Model

  Data Integration

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

  ⇒ Fusing objects, merging their properties

  – not compatible with XML Data Model (DOM, XML Query Data Model)

  – graph data model

- XPathLog as a Database Programming Language: DBPL'01
- Application in "intelligent" data integration: KRDB '01
- Implementation: LoPiX
  VLDB Demonstration Track

OVERVIEW

- Updates in XML
- XTreeGraph: An Alternative Data Model
- Data Integration
- Language: XPathLog
- Implementation: LoPiX
- Conclusion

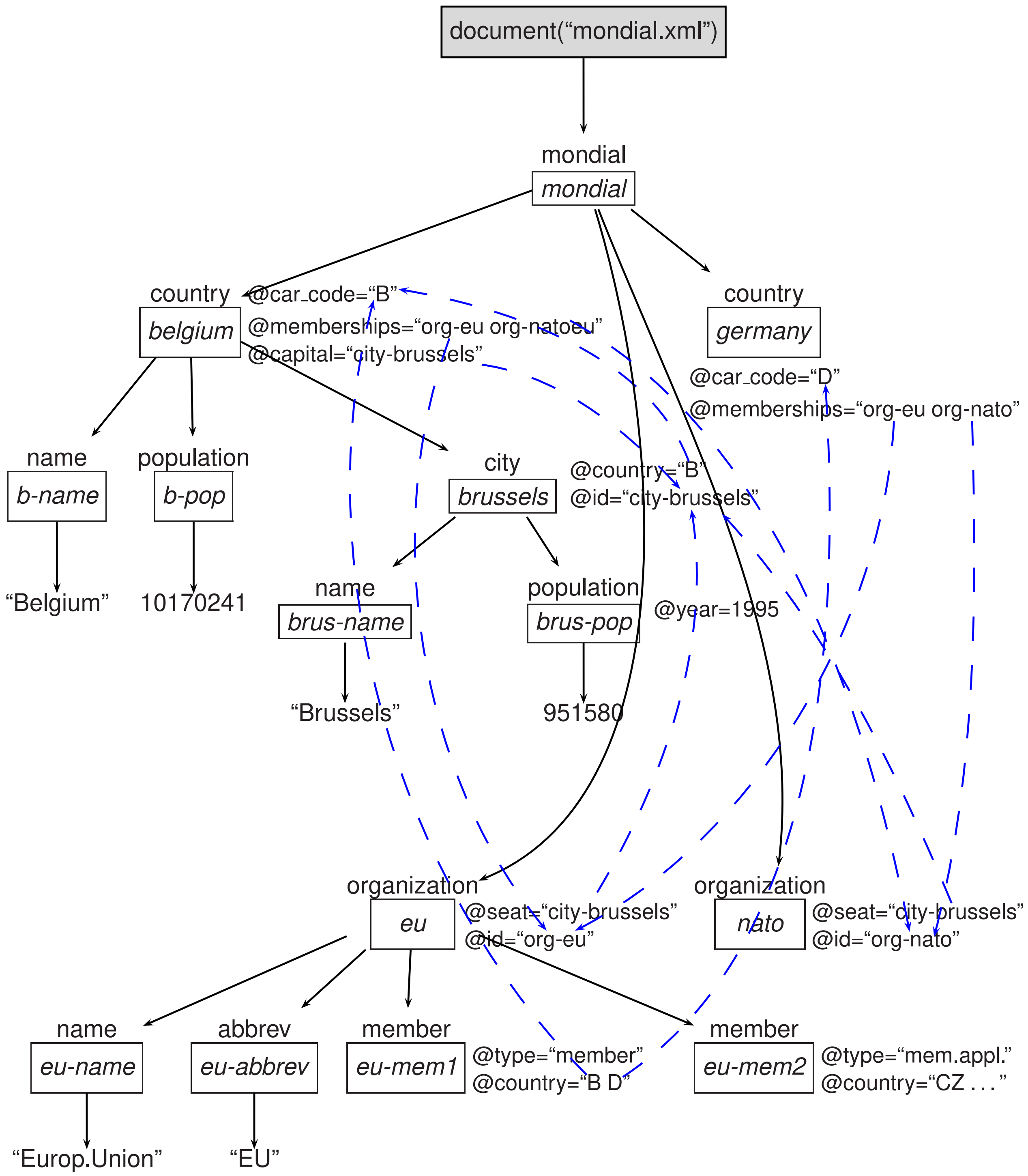
WHAT IS XML

- A *data model* for
  - representing semi-structured data
    - documents
    - “databases”
- most popular *representation*: ASCII;
  is *only one of the possible representations*.
- XML Data Model:
  - DOM (Document Object Model)
  - XML Query Data Model
  - XML instance consists of nested *elements* (tree/container structure)

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>Belgium</name>
      <population year=“95”>951580</population>
    </city>
    :
 </country>
 <country car_code=“D” capital=“cty-Berlin”
              memberships=“org-eu org-nato . . . ”>
    :
 </country>
 <organization id=“org-eu” seat=“cty-Brussels”>
   <name>European Union</name>  <abbrev>EU</abbrev>
   <members type=“member” country=“GR F E A D I B L . . . ”/>
   <members type=“membership applicant” country=“AL CZ . . . ”/>
 </organization>
 <organization id=“org-nato” seat=“cty-Brussels” . . . >
   :
 </organization>
   :
</mondial>
```

EXAMPLE: MONDIAL XML TREE



Languages

- Addressing/Selection:
  XPath
- Querying and Transformation
  - XSLT (Transformation): XPath
  - XML-QL (1998)
  - Quilt/XQuery (1999/2000)
    * FLWR-Clauses
    * XPath
- Generation
  - instantiating XML Patterns
- W3C XML Query Requirements/Data Model/Algebra
- no XML update language
  "Updating XML" @ SIGMOD 2001

XML-QL

- 1998 proposal
- influenced by STRUDEL/STRUQL,
- Data Model: edge-labeled graph model taken from Strudel
- XML-QL does not employ navigation and paths, but XML patterns, matched against the document,
- *selection part* and *construction part* based on XML Patterns

  WHERE *xml-pattern*
  IN *url*
  CONSTRUCT *xml-pattern*

```
WHERE
    <country car_code=$id>
        <name>$name</>
    </>
    IN "www.../mondial.xml"
CONSTRUCT <country car_code=$id name=$name> </>
```

- no notion of XML's axes
- dereferencing only supportable by joins

QUILT/XQUERY

- influenced by
  - SQL (clauses, variable bindings)
  - XPath (addressing nodes)
  - XSLT and XML-QL (output generation)

FOR *variable* IN *xpath-expr*
LET *additional_variable* := *xpath-expr*
WHERE *filters*
RETURN *xml-expr*

```
FOR $c IN document("mondial.xml")//country
LET $cities = $c/city
WHERE $c/@area > 1000000 and count($cities) > 10
RETURN
    <bigcountry area = $c/@area>
        <name>$c/@name</name>
        $cities
    </bigcountry>
```

State of the Art

- Navigation/addressing/selection: XPath
- two proposals of querying languages:
  W3C XQuery vs. XML-QL
- W3C XML Query Requirements/Data Model/Algebra
- no XML update language
  - fragments in Excelon/XUL
  - “Updating XML” @ SIGMOD 2001 [TIHW01]

XML DATA MODEL: DETAILS

- *abstract, informal* data model: tree, consisting of *nodes*.
- Specifications:
  - DOM (API)
  - XML Query Data Model (formal DM)
  - XML Schema ("DDL")
- unique *document node*; XPath: `document(`*url*`)`
- unique child: *root element*
  contains all other elements (recursively).
- *Element nodes*: inner nodes
  - name ("tag")
  - *element contents*: ordered list of children (*element nodes*, *text nodes*)
  - unordered set of *attribute nodes*
- *Text nodes*: leaves
- *Attribute nodes*: leaves,
  having a name, a type, and a value.
  - `CDATA`, `NMTOKEN`, `NMTOKENS`: scalar and multivalued value attributes
  - `ID`: a distinguished scalar `NMTOKEN` attribute. Uniquely identifies an element.
  - `IDREF`/`IDREFS`: scalar/multivalued *reference attribute*, value(s) must occur as `ID`

Constraints of the Data Model

- XML data exists only in form of (document) trees
- constraints too restrictive when dealing with data *manipulation* and *integration*:
- Node identity vs. element fusion
- referencing via ID vs. multiple sources
- Unique parent (integration) vs. element fusion
- Unique parent (inserting/copying) vs. reusing elements
- Unique element type vs. unifying terminologies

Minor flaws

- Handling attribute nodes with non-atomic values (often requires string manipulation)

Updating XML – The General Picture

Proposal for Variable-Based Languages in [TIHW01]

OO-style update methods associated with elements e.

- e.Delete($member$): remove a member (subelement (including PCDATA), an attribute, or a reference in an IDREF(S) attribute).
- e.Rename($member$,$name$): if $member$ is a non-PCDATA member it is given a new name.
- e.Insert($content$): add a subelement, an attribute, or a reference to an element.
- e.InsertBefore($child$,$content$), InsertAfter($child$,$content$).
- e.Replace($child$,$content$): equivalent to e.InsertBefore($child$,$content$) followed by e.Delete($child$).

PROPOSAL FOR XQUERY IN [TIHW01]

FOR *variable* IN *xpath-expr*

LET *additional_variable* := *xpath-expr*

WHERE *filters*

apply update method to variable

Update Methods

x.DELETE $*member*

x.RENAME $*member* TO *name*

x.INSERT *content* [BEFORE|AFTER $*child*]

x.REPLACE $*member* WITH *content*

Open Questions:

- Syntax for adding attributes?
- Rename to a name which already exists?
- if *content* is (or contains) an already existing XML element?

EXAMPLE

1. Add all member countries of the EU as subelements to the EU
2. Do this for all countries

   FOR $org IN //organization,
   WHERE $org/abbrev="EU"
   (FOR $country IN $org/members/@country→country
     $org.INSERT $country)

- Country elements now have two "parents"
  - move?
    - destroys original tree as a side effect
    - case (2): how to deal with duplication?
  - deep-copy?
    - maintenance of reference attributes
      - incoming, outgoing
      - here: country/@capital, country/borders@country, org/@seat
    - case (2) even more problematic
- XML graph database
- different views of the database

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

$\Rightarrow$ Fusing objects, merging their properties

- synonyms, ontologies
- not compatible with XML Data Model (DOM, XML Query Data Model)
- graph vs. tree

Alternative Data Model: XTreeGraph

- extends DOM/XML Query Data Model
- Supports updates
- Tailored to data integration
- Graph “database”
  - *XML element nodes*: vertices
  - subelement relationship without restrictions,
  - text children are leaf literals
  - *XML attribute nodes*:
  - multivalued attributes (`NMTOKENS`, `IDREFS`) are split
  - reference attributes (`IDREF`/`IDREFS`) are resolved
- element may be subelement of several vertices

⇒ multiple trees, overlapping,
local instead of global “document” order

⇒ define views

XTREEGRAPH

Stores only child and attribute axis (DOM-style)

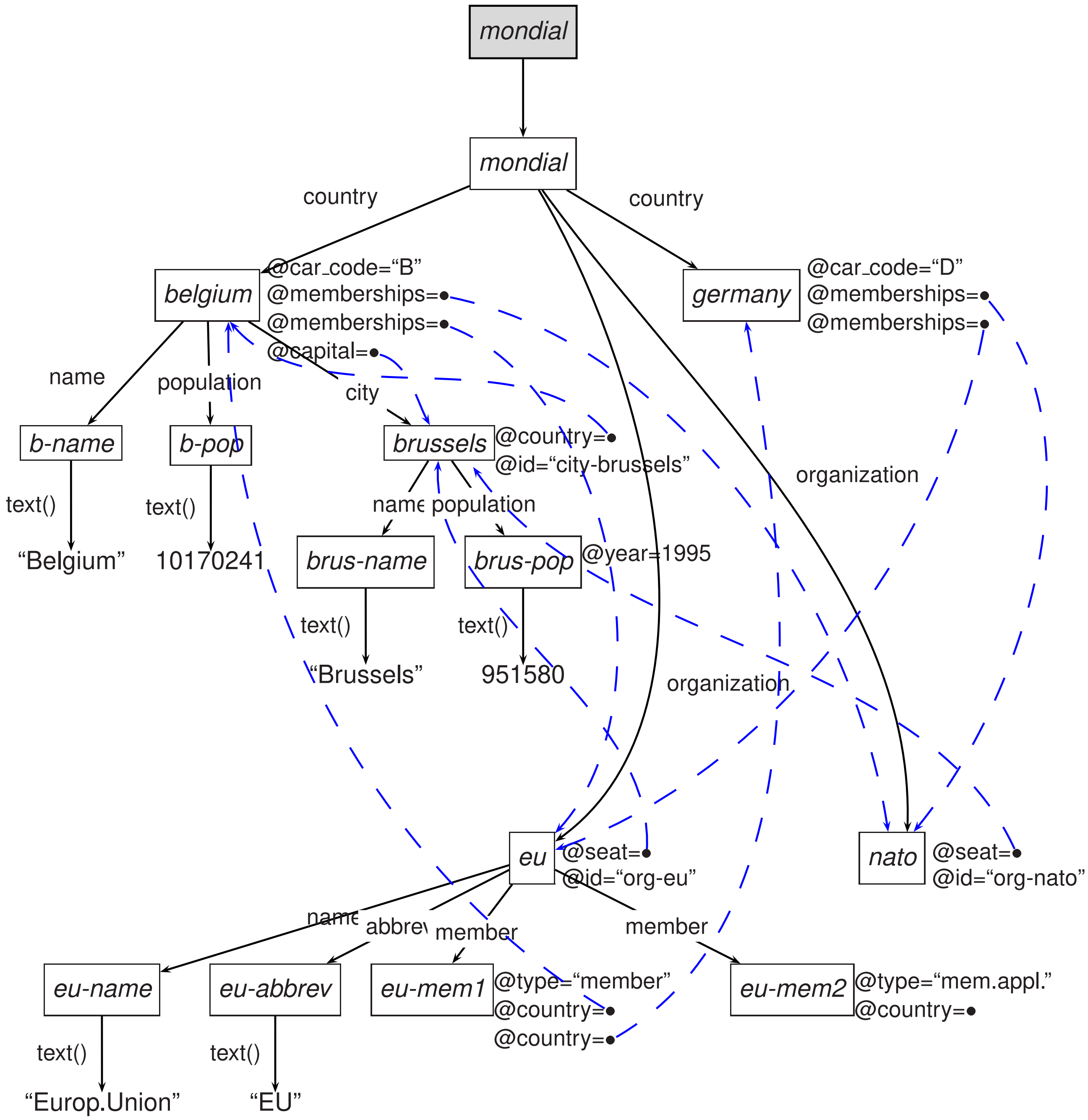
- $\mathcal{A}_\mathcal{X}(\mathtt{child}, x)$: list of $(name$,$element)$ pairs, including $(text$,$string)$ (edge-labeled graph)
- $\mathcal{A}_\mathcal{X}(\mathtt{attribute}, x)$: set of $(name$,$value)$ and $(name$,$element)$ pairs

- Mapping: XML instance $\rightarrow$ canonical XTreeGraph
- forest may become a graph through updates: $element$.INSERT $contents$ $\{$before|after$\}$ $child$

RELATIONSHIP WITH XML NOTIONS

- DOM API methods, XPath, XPointer, XML-QL, XQuery, XML Schema still apply
- XTreeGraph extends some features (E.g. DOM::giveParents)
- supports updates: $element$.INSERT $existing$-$element$ $\{$before|after$\}$ $child$
- supports data integration

Example: Mondial XTreeGraph



XML DATA INTEGRATION

- Exploits XTreeGraph properties

Abstract Operations

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

"THREE-LEVEL" MODEL

access multiple sources

- "basic" layer: source(s) provide tree structures,
- optionally with namespaces

merge data from different sources

- "internal" layer: XTreeGraph
  - overlapping trees
  - multiple parents
  - references

"export" layer: result trees views defined as projections

- root node
- signature

Case Study in Integration: The Mondial Geographic Database

XML representations of:

- CIA World Factbook: Countries
- CIA World Factbook: Organizations
- GlobalStatistics: Cities and Administrative Divisions
- ... some additional smaller sources

⇒ Integration

DATA INTEGRATION

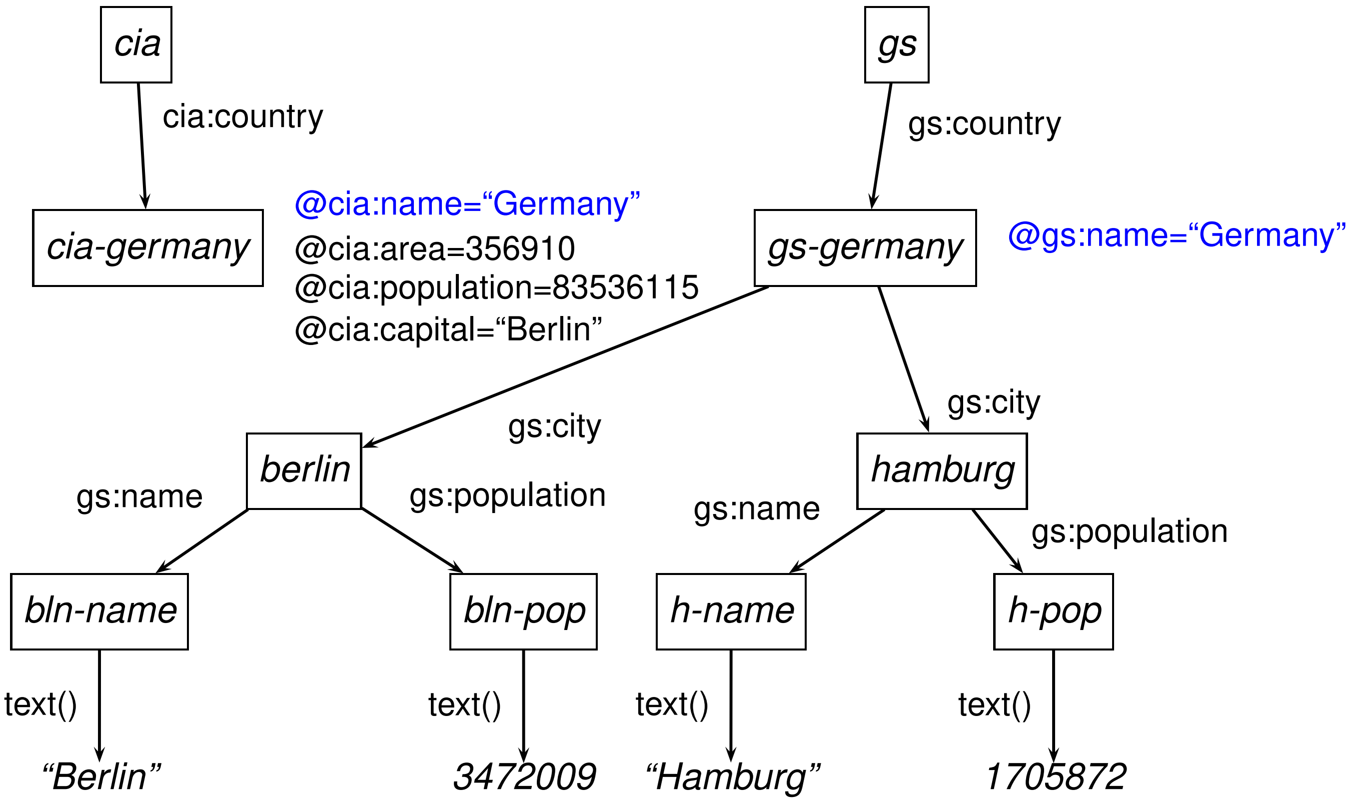
access multiple sources

- trees, optionally with namespaces

Example

Data Sources describing countries:

- cia: name, area, population and capital (by name)
- gs: cities with name, population



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 lists in $\mathcal{X}$ 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: $\mathcal{A}_{\mathcal{X}}(\text{attribute}, e_1)$ += $\mathcal{A}_{\mathcal{X}}(\text{attribute}, e_2)$.
4. e_1 collects the subelements of both original elements: $\mathcal{A}_{\mathcal{X}}(\text{child}, e_1)$ += $\mathcal{A}_{\mathcal{X}}(\text{child}, e_2)$ (order optional).

SYNONYMS

- some properties of the namespaced sources are taken to the result completely – with another name

$$\textit{namespace:name}_1 = \textit{name}_2$$

defines the property $\textit{name}_2$ to have the same extension as the original property $\textit{namespace:name}_1$.

cia:name = name.

cia:area = area.

cia:population = population.

cia:text() = text().

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

INTEGRATION: FUSING ELEMENTS AND SUBTREES

Example

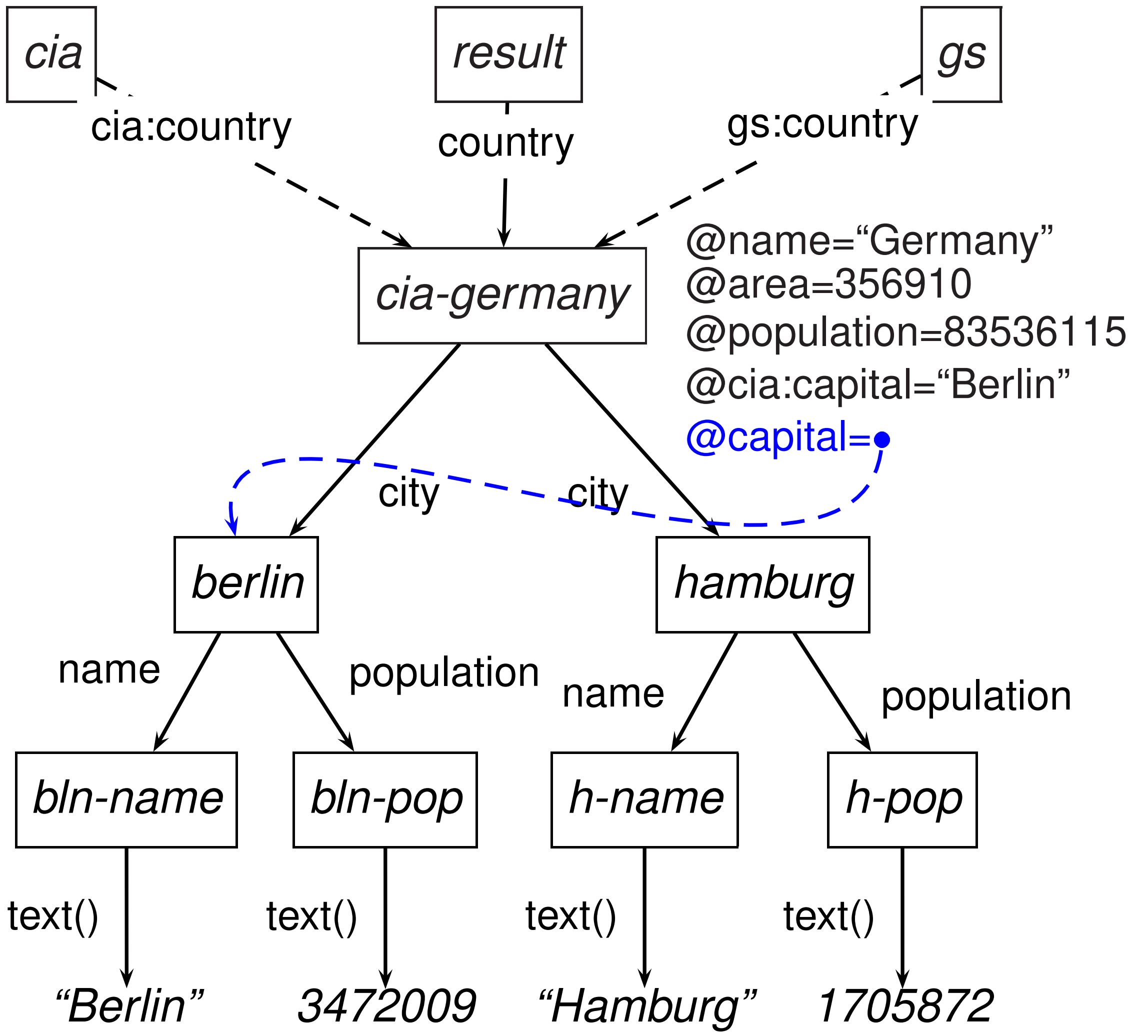
result[country→C1], C1 = C2 :-
  cia/cia:country→C1[@cia:name→N],
  gs/gs:country→C2[@gs:name→N].

Synonyms:

cia:name = name.	gs:city = city.
cia:area = area.	gs:name = name.
cia:population = population.	gs:population = population.
	gs:text() = text().

INTEGRATION: FUSING ELEMENTS AND SUBTREES

Example



C[@capital→City] :-

result/country→C[@cia:capital→Name and

city→City[name/text()=Name]].

Result Views

Projection by Signatures

Given:

- XTreeGraph $\mathcal{X}$,
- signature specification Sig (e.g., by a set of signature atoms [derivable from DTD or XML Schema]),
- a root node r.

Tree view $\mathcal{Y}$ rooted in r:

- r is a node in $\mathcal{Y}$,
- $\mathcal{A}_{\mathcal{Y}}(n, \text{child}) = list((v, e) \in \mathcal{A}_{\mathcal{X}}(n, \text{child}) \mid c[e \Rightarrow c'] \in Sig)$ and all these v belong to $\mathcal{Y}$
- $\mathcal{A}_{\mathcal{Y}}(n, \text{attribute}) =$
$$list((v, e) \in \mathcal{A}_{\mathcal{X}}(n, \text{attribute}) \mid c[@e \Rightarrow c'] \in Sig)$$

Integrity Constraints:

- $\mathcal{Y}$ may contain dangling references.
- $\mathcal{Y}$ may be cyclic/infinite.

Projection

Example

Projection of the result view tree:

result isa mondial.

mondial[country⇒country]. % signature

country[@name⇒string].

country[@area⇒numeric].

country[@population⇒numeric].

country[@capital⇒city].

country[city⇒city].

city[name⇒name].

name[text()⇒string].

city[population⇒population].

population[text()⇒numeric].

EXAMPLE REVISITED

Consider again the organizations example.

- (country) elements having multiple parents
- no problem with maintaining reference attributes (borders, capital, seat)
- each organization defines a result view

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
- XTreeGraph model
- support for 3-level integration approach

XPathLog by Examples

Pure XPath expressions

?- //country[name/text() = “Belgium”]//city/name/text().

true

Output Result Set

?- //country[name/text() = “Belgium”]//city/name/text()→N.

N/“Brussels”

:

Additional Variables

?- //country[name/text()→N1 and

@car_code→C]//city/name/text()→N2.

N2/“Brussels” C/“B” N1/“Belgium”

:

Dereferencing

?- //organization[@seat = members/@country/@capital]

/@seat/name/text()→N.

Schema Querying

?- //city/N.

N/name

:

XPathLog: Syntax

Extends the XPath syntax

XPathLog *reference expressions* are XPath *location paths*

```
[0]   ReferenceExpr ::= AbsLocPath
                      | ConstLocPath
[2b]  ConstLocPath ::=  constant "/" RelLocPath
                      | variable "/" RelLocPath
```

Extend *LocationSteps*

```
[4] Step ::= AxisSpec NodeTest Pred*
           | AxisSpec NodeTest Pred* "->" Var Pred*
           | AxisSpec Var Pred*
           | AxisSpec Var Pred* "->" Var Pred*
```

- navigation by dereferencing `IDREF` attributes
- Predicates over reference expressions
- Query semantics extends semantics for XPath by P.Wadler (1999) with variable bindings

SEMANTICS OF XPATHLOG

Example

$expr$ =

//organization→O[member/@country[@car_code→C and name/text()→N]]/abbrev/text()→A.

results in

list((“UN”, {(O/*un*, A/“UN”, C/“AL”, N/“Albania”),
(O/*un*, A/“UN”, C/“GR”, N/“Greece”),
: }),
(“EU”, {(O/*eu*, A/“EU”, C/“D”, N/“Germany”),
(O/*eu*, A/“EU”, C/“F”, N/“France”),
: }),
:)

XPathLog Rules

head($V_1,\ldots,V_n$) :- 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:

- $host[property{\rightarrow}value]$ modifies $host$
- $host/property\ remainder$
  inserts new element $host/property$ which satisfies $remainder$
- $property$ of the form
  - child::$name$
  - child(i)::$name$
  - preceding/following-sibling::$name$
  - preceding/following-sibling(i)::$name$
  - attribute::$name$

$\Rightarrow$ unambiguous insertions

Rule Heads: Updates

Generate and Extend Attributes

C[@datacode→“de”], C[@memberships→O] :-
//country→C[@car_code=“D”],
//organization→O[abbrev/text()→“EFTA”].

Create new elements: free elements

/country[@car_code→“BAV”].

Add subelement relationships

C[@capital→X and city→X and city→Y] :-
//country→C[@car_code→“BAV”],
//city→X[name/text()=“Munich”],
//city→Y[name/text()=“Nurnberg”].

Generation of Elements by Path Expressions

C/name[text()→“Bavaria”] :-
//country→C[@car_code=“BAV”].

SEMANTICS OF RULE HEADS

Attributes

C[@datacode→"de"], C[@memberships→O] :-

  //country→C[@car_code="D"],

  //organization→O[abbrev/text()→"EFTA"].

$\Downarrow$

<country $\boxed{\text{datacode="de"}}$ car_code="D"
  memberships="org-eu org-un $\boxed{\text{org-efta}}$ ...">
  ⋮
</country>

- C: host element
- O: target of reference
- extends $\mathcal{A}_{\mathcal{X}}(\text{attribute}, \textit{germany})$

Semantics of Rule Heads

Create “free” elements

/country[@car_code→“BAV”].

⇓

<country car_code=“BAV”> </country>

SEMANTICS OF RULE HEADS

Add subelement relationships

C[@capital→X and city→X and city→Y] :-

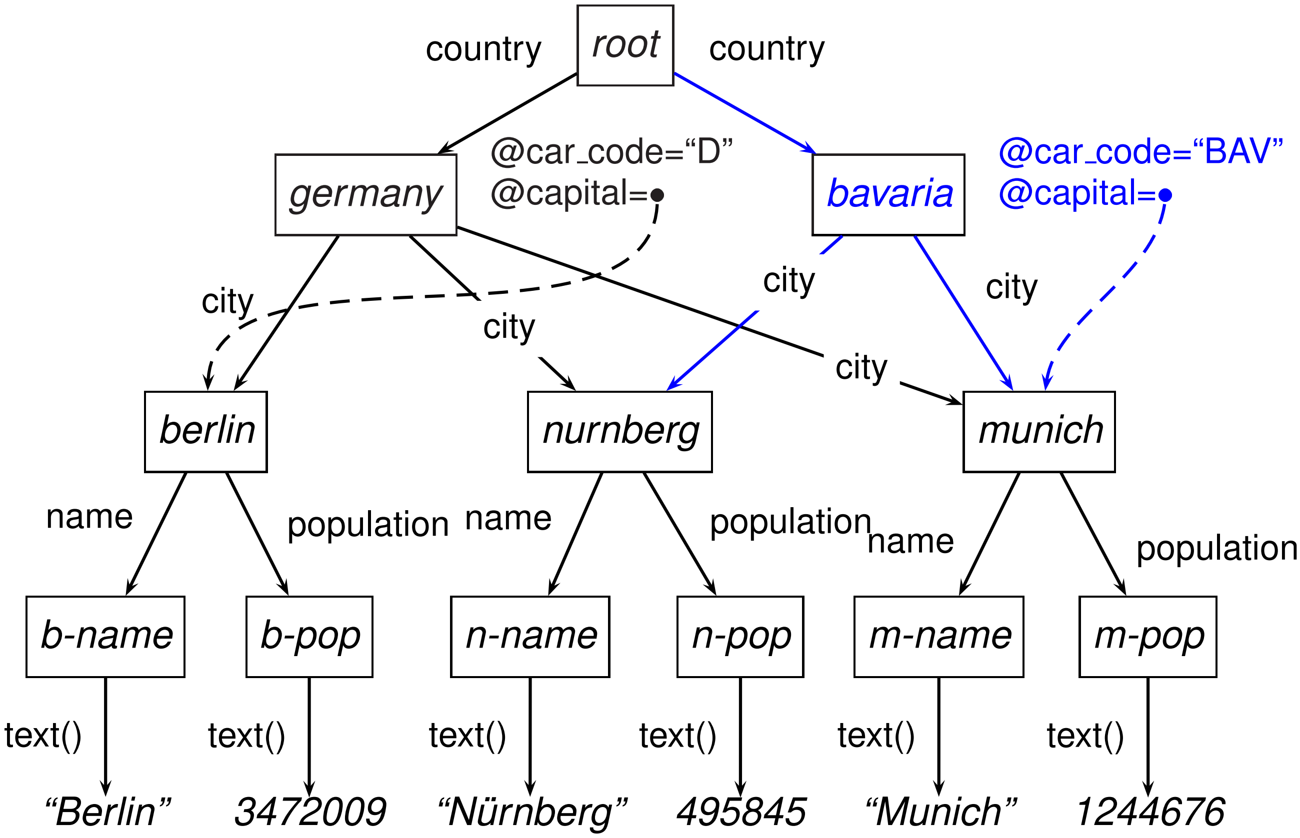
//country→C[@car_code→“BAV”],

//city→X[name/text()=“Munich”],

//city→Y[name/text()=“Nurnberg”].

- city elements are *linked* as subelements.
- extends $\mathcal{A}_{\mathcal{X}}(\mathrm{child}, \textit{bavaria})$ with $(\textit{munich}, \mathrm{city})$ and $(\textit{nurnberg}, \mathrm{city})$.
- crucial for efficient in-place restructuring and integration

Example: Linking



- elements may have multiple parents

SEMANTICS OF RULE HEADS

Generation of Elements by Path Expressions

C/name[text()→"Bavaria"] :-
    //country→C[@car_code="BAV"].

$\Downarrow$

<country car_code="BAV" capital="city-munich">
  <city>. . . </city>
  <city>. . . </city>
  <name>Bavaria</name>
</country>

Atomized:

C[name→_N], _N[text()→"Bavaria"] :-
  root[descendant::country→C], C[@car_code="BAV"].

- variable binding $\mathsf{C}/\textit{bavaria}$.
- insert $\textit{bavaria}[\mathsf{child::name}\rightarrow x_1]$:
  add an (empty) name subelement (x_1, name) to $\mathcal{A}_{\mathcal{X}}(\textit{bavaria}, \mathsf{child})$.
- bind the local variable $\_N$ to x_1.
- generate the text contents to x_1:
  add ("Bavaria",text) to $\mathcal{A}_{\mathcal{X}}(x_1, \mathsf{child})$.

XPathLog: Extension Concepts

Classes, Class hierarchy

- class hierarchy not supported by DTD
- XMLSchema datatype and element type definitions
- default and fixed values (DTD, XSD)
- XPathLog supports class hierarchy and non-monotonic value inheritance

XPathLog: Extension Concepts

Signatures

- class-property-class relationships
- structural inheritance
- XPathLog: “lightweight” signature formalism:

  country[@car_code⇒string].

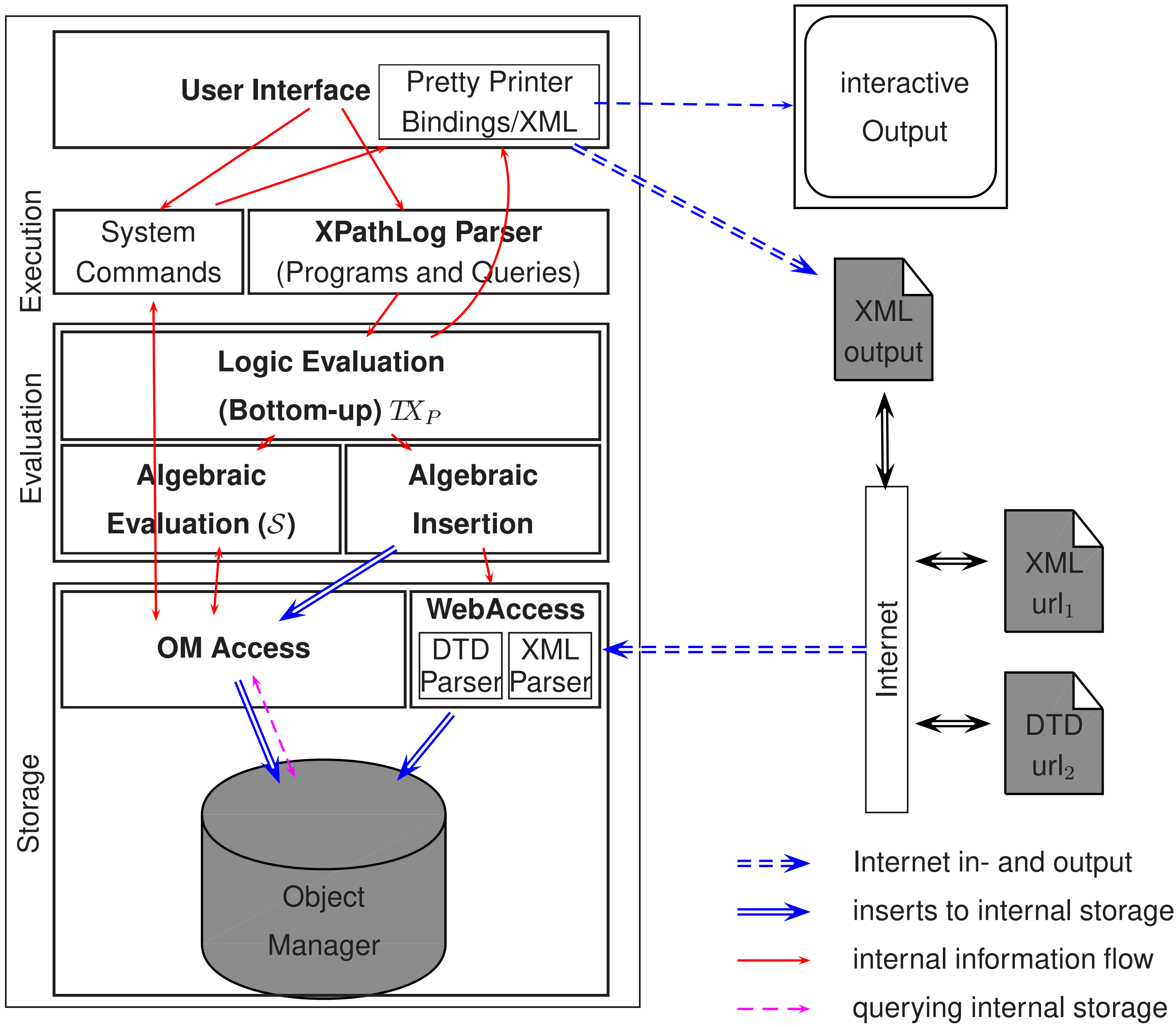
  country[@area⇒numeric].

  country[@capital⇒city].

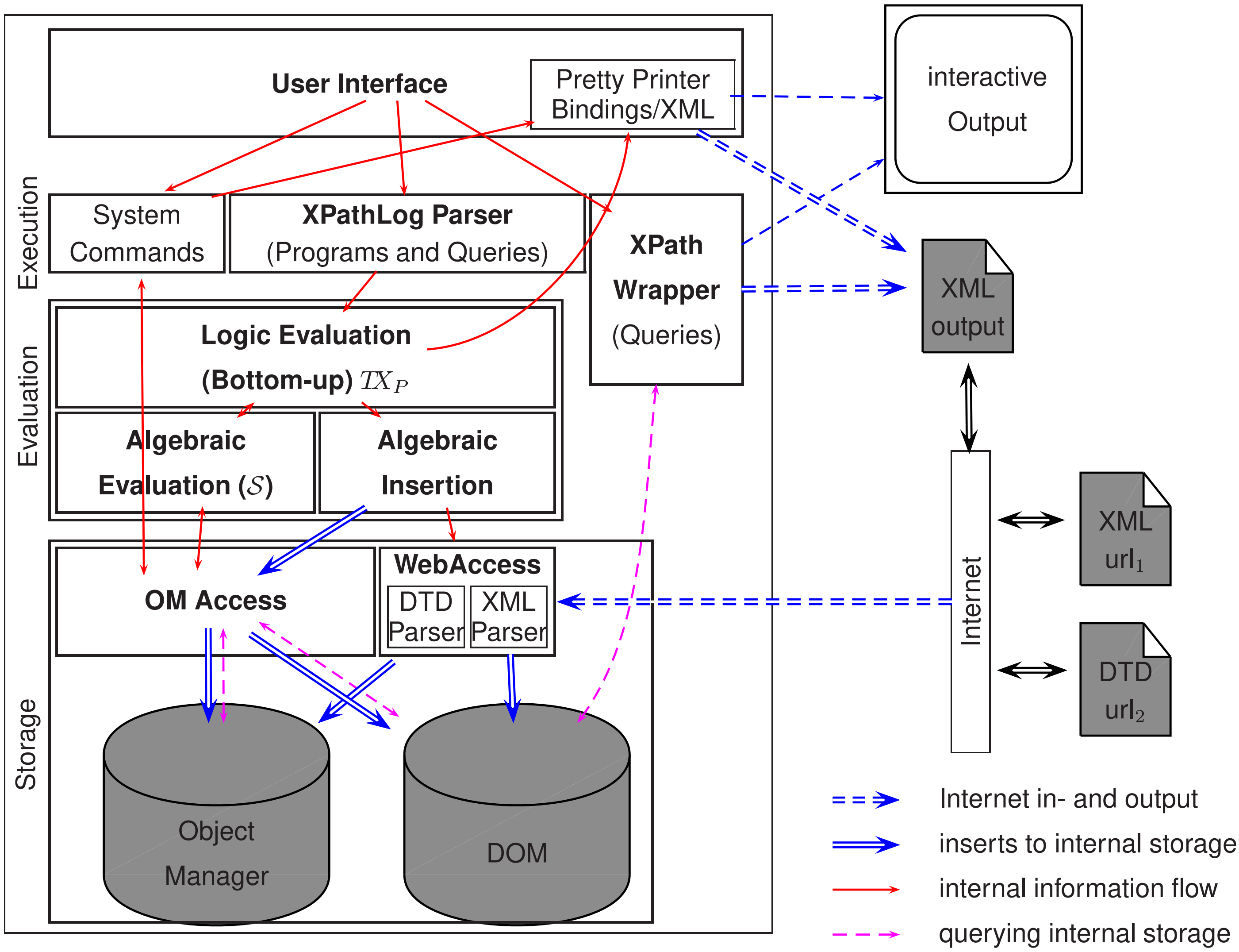
  country[city⇒city].
- derivable from DTD or XML Schema
- used for defining projections of the internal database

IMPLEMENTATION: LOPIX

- developed using major components from FLORID



ARCHITECTURE: DUAL MEMORY LOPIX



- DOM: XML documents
- OM: predicates, class hierarchy, signature
- DOM: no index structures
  $\Rightarrow$ stored in the OM, supported by OMAccess

$\Rightarrow$ only minor changes in OMAccess

Results

- declarative semantics for generating XML with XPath
- graph data model suitable & necessary for updates and integration
- firstly implemented XML updates
- DOM/XML Query Data Model not suitable:
  unique-parent, no fusion, synonyms, linking
- Updates in XML: requires to copy trees
  Problems with IDREFs

Conclusion

- practicable approach
  (implementation + case study)
  - Faster than Kweelt, XML-QL, and Excelon for queries
- data model
- linking, fusing, synonyms (also memory-saving)
- powerful, flexible language
  - (metadata/schema querying)
  - element creation
  - element fusion
- extension concepts
  (classes, signatures)
- data-driven Web access,
  extensible for XLinks

Contents