Handling XML with a Deductive Database System

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

Workshop Internet-Datenbanken
Berlin, 19.9.2000

Semistructured Data

Documents vs. Databases

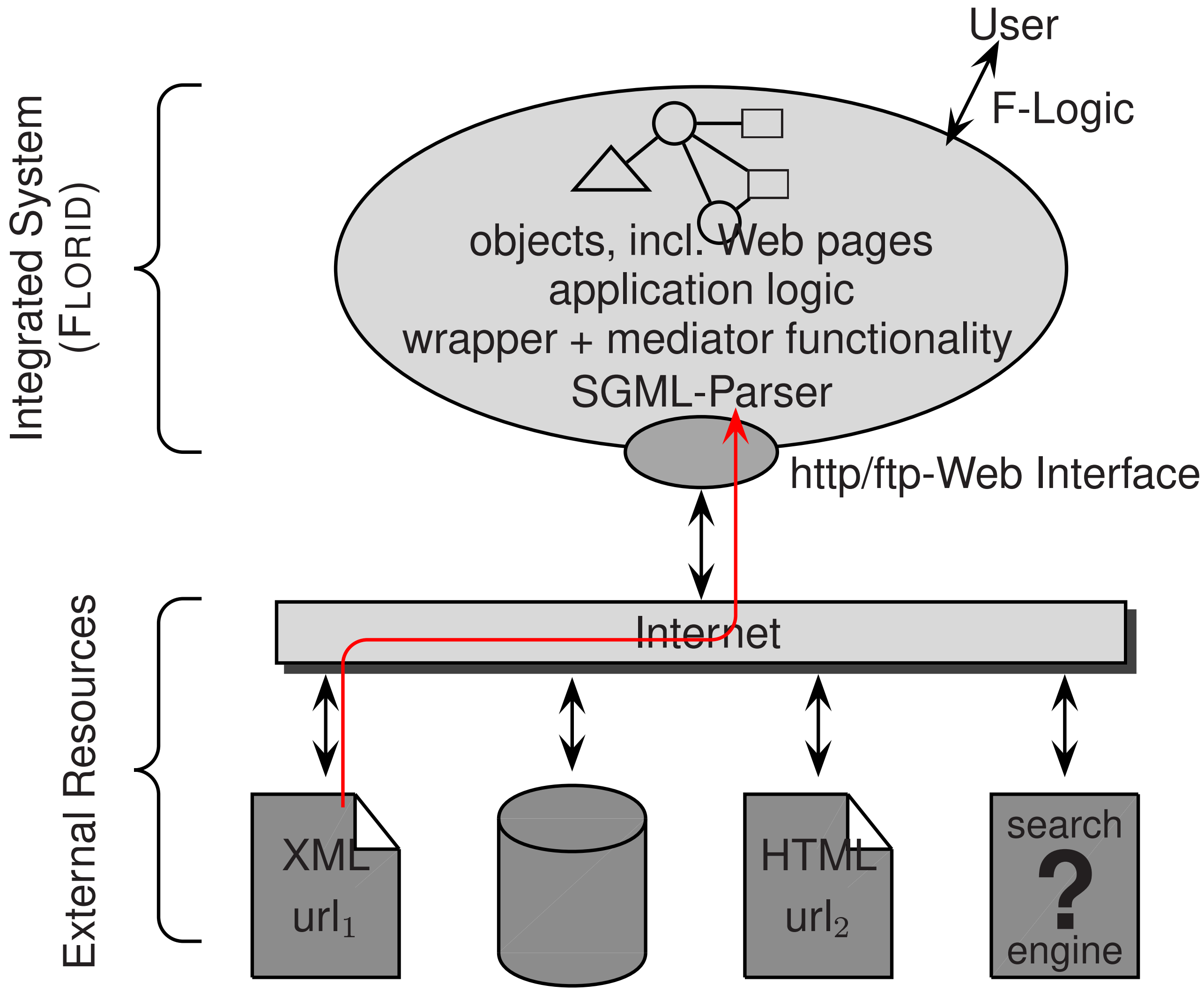
- Documents
  (HTML), SGML and some XML sources
  - parse-trees
  - nested structure and cross-references
  - parent-children-relationships
  - siblings with ordering
- XML "databases"
  - objects, graph-like structures
  - references
  - hierarchical structure and ordering not induced by the application domain

  application-specific tags
  ⇒ induce a database schema
- Main Topic: XML as a semistructured data(base) model

Starting Point

- FLORID: a deductive object-oriented/semistructured database system which has been extended with
  - Web access
  - Web searching
  - Wrapping
  - Integration of Web sources (HTML)
    HTML-Legacy:
    Data represented by documents
    * wrapping parse-tree to application level model by using the parse-tree
- Next step:
  How to extend to XML sources?
  - direct mapping from XML to application-level internal model
- General questions:
  - (Other) approaches to integration of XML sources?
  - Languages/Systems for manipulating XML data?
  - extended concepts for XML (schema, links, classes)

Architecture of Web Access with FLORID



- Unified, integrated framework for wrappers and mediators
- F-Logic: unified data model, wrapper, mediator, and querying language
- Data Model: Representation of the Web fragment and application-level representation.
  Structure + Contents of the Web as a unit

Example: MONDIAL

```
<continent id="europe">
  <name>Europe</name>
  <area>9562488</area>
</continent>

<country car_code="D" capital="cty-Germany-Berlin"
  memberships="... org-CERN org-ESA org-EU org-IOC
                   org-NATO org-UN org-WHO ...">
  <name>Germany</name>
  <population>83536115</population>
  <encompassed continent="europe" percentage="100"/>
  <ethnicgroups percentage="95.1">German</ethnicgroups>
  <ethnicgroups percentage="2.3">Turkish</ethnicgroups>
  <ethnicgroups percentage="0.4">Greeks</ethnicgroups>
  <ethnicgroups percentage="0.7">Italians</ethnicgroups>
  ...
  <religions percentage="37">Roman Catholic</religions>
  <religions percentage="45">Protestant</religions>
  <languages percentage="100">German</languages>
  <border country="F" length="451"/>
  <border country="A" length="784"/>
  <border country="CZ" length="646"/>
  <border country="CH" length="334"/>
  <border country="PL" length="456"/>
  <border country="B" length="167"/>
  <border country="L" length="138"/>
  <border country="NL" length="577"/>
  <border country="DK" length="68"/>
```

```
  <province id="prov-Germany-2" capital="cty-Germany-9">
    <name>Baden Wurttemberg</name>
    <area>35742</area>
    <population>10272069</population>
    <city id="cty-Germany-9" province="prov-Germany-2">
      <name>Stuttgart</name>
      <longitude>9.1</longitude>
      <latitude>48.7</latitude>
      <population year="95">588482</population>
    </city>
    <city id="cty-Germany-24" province="prov-Germany-2">
      <name>Karlsruhe</name>
      <population year="95">277011</population>
    </city>
    ...
  </province>
  <province id="prov-Germany-3" capital="cty-Germany-Munich">
    <name>Bayern</name>
    <area>70546</area>
    <population>11921944</population>
    <city id="cty-Germany-Munich" province="prov-Germany-3">
      <name>Munich</name>
      <longitude>11.5667</longitude>
      <latitude>48.15</latitude>
      <population year="95">1244676</population>
    </city>
    ...
  </province>
  ...
</country>
```

```
<organization id="org-EU" seat="cty-Belgium-2">
  <name>European Union</name>
  <abbrev>EU</abbrev>
  <established>07 02 1992</established>
  <members type="member"
           country="GR F E A D I B L NL DK SF S IRL P GB"/>
  <members type="membership applicant"
           country="AL CZ H SK LV LT PL BG RO EW M CY"/>
</organization>
```

Data Model: Database Point of View

- XML instance represents several objects, their properties and relationships.
- same application area can be represented by *different* DTDs (deep hierarchical structure with subelements vs. reference attributes)

```
<country name="Germany">
    <city name="Berlin" ... />
    <city name="Hamburg" ... />
</country>
```

```
<country city="berlin hamburg">
    <name>Germany</name>
</country>
<city id="berlin" name="Berlin" .../>
<city id="hamburg" name="Hamburg" .../>
```

- $\Rightarrow$ data model and querying should be as independent as possible from document structure

  ?- country.city.name.

  (leads to possible database-equivalence notions between DTDs and instances)

- ordering less important (or not at all)

Querying Language

- independent from actual representation
  (subelements vs. attributes)
- navigation
  - through the document hierarchy
  - following references

  should be equivalent/transparent.
- XML-model-based (like XML-QL) or abstract ssd model?
- Variable Bindings (Prolog) or structures (XML-QL)?

Data Manipulation Language

- declarative
- closely related to the querying language
- rule-based

Presentation of Results

- Prolog-style “answers” to queries
- XML-export for reuse of integrated data

XML: Hierarchy + References

- paths through the hierarchy,
- references are represented by named reference attributes and id's.

XML Problems: References

Current XML querying does not support derefencing:

- no implicit dereferencing
- XQL/XPath: dereferencing via the id(...) function
  ⇒ not possible to follow more than one IDREF in an expression.
- XML-QL: dereferencing via Joins or id(...)
- Quilt (not yet implemented): dereferencing via the "→"-operator:

  id("Germany")→@capital

⇒ different syntax for navigation, dependent on the *representation*:

- subelements: country/name, country/city
- attributes: country/@name, country/@capital
- reference attributes: country→@city, country→@capital

XML Problems: Multivalued Attributes

XML Attributes are not really multivalued:

XPath provides no construct for splitting NMTOKENS and IDREFS attributes (except string functions):

```
<!ELEMENT country (...)>
  <!ATTLIST country car_code ID #REQUIRED
                    industry NMTOKENS #IMPLIED
                    memberships IDREFS #IMPLIED>
```

```
<country car_code="CH"
         industry="machinery chemicals watches"
         memberships="org-efta org-un ...">
         ...
</country>
```

- [[id(“CH”)/@industry]] = “machinery chemicals watches”
  [[//country[@industry = 'watches']/@car_code]] = $\emptyset$
- but: IDREFs attributes are automatically split when id(...) is applied:
  [[id(id(“CH”)/@memberships)/@abbrev]] =
  = {“EFTA”, “UN”, . . . }
  [[//country[id(@memberships) = id('org-EFTA')]/@car_code]]
  = {“CH”, . . . }

Manipulation of XML Data

- “native” XML languages:
  - XSLT: functional-style transformation language XML $\rightarrow$ XML
  - XML-QL: querying language XML $\rightarrow$ XML

  no means for defining views,
- DOM/Java: computationally complete environment by mapping XML to Java types (DOM model)

$\Rightarrow$ no “XML database language”

XML Data Model?

- XML is a *representation*
  (lack of languages indicates that it is not a real data model?)

The XML "data model" is less expressive than the object-oriented model:

- no class *hierarchy*
- only very restricted inheritance concepts
- ... a typed model, complex objects ...

Mapping to an object-oriented Model

- classes: elements (all elements?)
  city, country, organization ...
  name?, population?, geo_coordinates?...
- properties:
  - literal-valued: country.name
  - object-valued: country.capital, organization.member
  - complex-valued: city.geo_coordinates
  - scalar or multivalued
- schema knowledge: country.capital isa city
- defaults: desert.ground = sand

Mapping XML to an object-oriented Model

Formal Framework: F-Logic

- is-a atoms: o : c
- subclass atoms: c :: d
- Method applications to objects:
  o[m→v] (scalar)
  o[m↠v] (multivalued)
  inheritable:
  c[m•→v]
  c[m•↠v]
- Signatures of methods:
  c[m⇒v] (scalar)
  c[m⇛v] (multivalued)
- Variables allowed at all positions
- Entities can act at the same time as classes, objects and methods
- Rules over atoms: <head> :- <body>.
- Program: a set of rules

Representation of XML in F-Logic

U.parse@(xml, _ , _ , _ , _) :- U:url, ...

- parses the contents of U as XML document:
- element types define classes, element instances define objects of these classes,
- subelement relationships define object-valued properties,
- attributes (CDATA, NMTOKENS, ID) define literal properties (scalar/multivalued),
- numerical values (XML knows only strings) are interpreted as numbers/integers
- IDREF/S attributes define object-valued properties (scalar/multivalued).

Representation of XML in F-Logic: Example

```
<!ELEMENT country (city+)>
  <!ATTLIST country car_code ID #REQUIRED
                    name CDATA #REQUIRED
                    capital IDREF #REQUIRED
                    industry NMTOKENS #IMPLIED
                    memberships IDREFS #IMPLIED>
```

```
<country car_code=“CH” name=“Switzerland”
         capital=“city-ch-bern”
         industry=“machinery chemicals watches”
         memberships=“org-EFTA org-UN ...”>
  <city id=“city-ch-bern”> ... </city>
  ...
</country>
```

```
<organization id=“org-EFTA” abbrev=“EFTA” .../>
```

Result:

ch:country[name$\rightarrow$“Switzerland”;
car_code$\rightarrow$“CH”; capital$\rightarrow$*bern*;
industry$\twoheadrightarrow$$\{$“machinery”, “chemicals”, “watches”$\}$;
memberships$\twoheadrightarrow$$\{$*efta*, *un*, $\ldots\}$;
city$\twoheadrightarrow$$\{$*bern*, $\ldots\}$].

bern:city[name$\rightarrow$“Bern”; $\ldots$].

un:organization[abbrev$\rightarrow$“UN”].

Metadata

- U.parse@(dtd, _) :- U:url, ...
  parses U as a DTD and generates F-Logic signature atoms.

- U.parse@(xml, _ , _ , $signature$, _) :- U:url, ...
- parses U as an XML document together with a DTD/XMLSchema document
  - sig: based on already stored F-Logic Signature atoms
  - dtd: based on DTD given in the XML instance
  - *url*: parses *url* as DTD or XMLSchema
- generates F-Logic signature atoms

  country[name⇒literal; car_code⇒literal;
  capital⇒city; industry⇒⇒literal;
  memberships⇒⇒organization; city⇒⇒city; ...]
- additionally: enumerations and default declarations

Defaults

- DTD provides information about default values:

  <!ELEMENT desert (...)>
  <!ATTLIST desert ...
  temperature NMTOKEN 'hot'
  ground (sand|boulders|rocks|snow) 'sand'>

- become inheritable properties of the class desert:

  desert[temperature$\bullet\!\!\rightarrow$"hot"; ground$\bullet\!\!\rightarrow$sand].

- Combination with data from the XML instance:

  <desert name="Sahara" .../>

  sahara: desert[name$\rightarrow$"Sahara";
  country$\twoheadrightarrow${*marocco*, *algeria*, ...}].

  automatically derives (nonmonotonic inheritance)

  sahara[temperature$\rightarrow$"hot"; ground$\rightarrow$sand].

- IDREF defaults need special treatment (target of the idref is not known in the DTD!)

Literal Values, PCDATA contents

- uniform treatment of PCDATA elements and attributes
- attribute:

  <country name="Germany"/>

  germany:country[name→"Germany"].

  ?- _:country[name→N]

  N/"Germany"

- subelement:

  <country>

  <name>Germany</name>

  </country>

  germany:country[name→*ger-name*].

  ger-name:name[pcdata→→"Germany"].

  ?- _:country[name→N].

  N/*ger-name*

  ?- _:country.name[pcdata→→N].

  N/"Germany"

- "Annotated Literals": PCDATA value replaces the object in answers (and similar situations):

  <country name="Germany"/>

  germany:country[name→"Germany"].

  ?- _:country[name→N].

  N/"Germany"

  ?- _:country.name[pcdata→→N].

  N/"Germany"

Annotated Literals

<city name="Berlin">
<population year="1995">3472009</population>
</city>

berlin:city[name→"Berlin"; population↠*bln-pop-95*].
bln-pop-95:population[year→1995; pcdata↠3472009].
?- _:city[population↠P].
P/3472009
?- _:city[population↠P[year→Y]].
P/3472009 Y/1995

- automatically resolved in
  - answers,
  - literal comparisons, functions, and conversions ($<$, $>$, strlen, strcat, ...)
- although, the *variable* is always bound to the object (e.g., for use in the rule head).
- above,
  ?- 3472009[year→1995]
  does not hold (not a property of the integer object 3472009)

Querying

- queries return optionally
  - sets of variable bindings, or
  - F-Logic molecules
- compare with XPath with return operators.

Simple Navigation

- Car codes of countries with the names of all cities:

```
?- _:country[car_code->N]..city[name->CN].
XPath:
  //country[@car_code?]/city/name.text()?
or
  //country[@car_code?]/city/@name?
```

- XPath: query depends on representation as subelements or attributes.

Querying: Dereferencing

- XPath: via id(...) function
- for all organizations, their name and abbreviation, and the name of their seat city.

```
?- _:organization[name->N; abbrev->A].seat[name->SN].
XPath:
  id(//organization[@name? AND @abbrev?]/@seat)
    /@name?
```

- for all organizations, the name, and all members together with their membertype.

```
?- _:organization[name->N]
                 ..member[type->T]
                         ..country[name->CN].
XPath:
  id(//organization[@name?]/member[@type?]/@country)
    /@name?
```

- all together:

```
?- _:organization[name->N; abbrev->A;
                  seat->_[name->SN]]
                 ..member[type->T]..country[name->CN].
```

- not possible in XPath:
  - impossible to apply two independent id(...)
  - no joins

Querying: Aggregations, Multiple Joins

- exploit the full power of the deductive database system
- in general not expressible in XPath/XSL
- for all organizations, the sum of inhabitants of all member countries, grouped by membership types:

```
?- _O:organization[name->N]..member[type->T],
   SP = sum{P [_O,T];
            _O..member[type->T]
                      ..country[population->P]}.
```

Multiple Sources

- U.parse@(xml, _ , _ , _ , $context$)
- $context$ identifies source (similar but independent from namespaces),
- all data is labeled with the context:

  u.parse@(xml,nil,nil,dtd,mondial).
  belgium:(mondial.country)[(mondial.capital)→*brussels*].
- useful for data integration

Integrating Multiple Sources

```
cia:source.
gs:source.

C1 = C2, C1:country :-
   C1:(cia.country)[name@(cia)->N],
   C2:(gs.country)[name@(gs)->N].
%% ... further rules for fusing countries ...

X[country->C] :-
   X:(Source:source.city)[country@(Source)->C].

X1 = X2, X1:city :-
   X1:(cia.city)[name@(cia)->N; country->C],
   X2:(gs.city)[name@(gs)->N; country->C].
%% ... further rules for fusing cities ...
```

- fused objects collect properties of original objects

XML-Parsing in Florid: Ordering

- U.parse@(xml, _ , ordering, _ , _)
  generates additional parse-tree representation, including ordering.

XML-Output

- ?- sys.theOMAccess.export@("xml", _ , _).
  outputs all facts which match the current signature atoms in XML format.

Conclusion

- mapping XML Databases to an object-oriented model
- investigations use an existing system
- useful querying and restructuring functionality
- practicability

⇒ Logic Programming Language for Database-XML
(no XML in-place manipulation language except Java/DOM available, only transformation languages)
  - mainly a matter of terminal symbols and operators, *not* of the parsetree of the language

Further Work/Perspectives

- extension for XMLSchema (based on built-in XML Parsing)
- Equivalence and overlapping of XML Instances/Schemas
- Algorithms for integration of XML Instances