Handling XML with a Deductive Database System

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

Abstract

We propose an integration of XML with F-Logic, a deductive object-oriented database framework. The F-Logic data model is in fact a *semistructured* data model, exhibiting many similarities with the XML/DOM data model: there is a canonical mapping from XML to a fragment of F-Logic. The advantages of the integration are that the full expressiveness of F-Logic (and the functionality of the FLORID system) can be applied to the data, providing an intuitive language for view definitions, updates, schema reasoning, Web exploration, integration (also with non-XML sources) etc. Especially, extended XML features, such as XML Schema or XLink can easily be prototypically implemented and evaluated.

1 Introduction

XML is a data model for *semistructured* data where it plays the same role as the relational or the object-oriented model for classical databases. *Semistructured* means that the data is not subject to a fixed, homogeneous schema. Compared with the object-oriented model, objects can still be grouped by a class-like notion, but the properties of the objects in a class may vary. Traditionally, this has been solved by *null* values – properties which are not applicable are set to null. On the other hand, this leads to lots of explicit nulls, and, adding a potential property (which is possibly only relevant for a small number of objects) requires not only to add it to the schema, but also to add a null-valued property to the other objects.

In *semistructured data models*, properties can be omitted, meaning *implicit* null values. Thus, a property which is applicable only for some objects is not considered for the others. In some sense, an object is simply a collection of properties. The schema description is optional: semistructured data is inherently *self-describing*. When adding new properties, only the schema information (if available) has to be updated and objects with the new property can be added to the database without bothering old objects.

Semistructured data has been investigated for storing documents (SGML), especially Web documents (HTML), data integration (OEM) [GMPQ$^+$97] and knowledge representation (e.g., KIF; knowledge interchange format [KIF98]). An overview of such frameworks under the aspect of a Web data model is given in [Man98], where also F-Logic [KLW95] has been suggested for modeling semistructured data. Applications of F-Logic in this area have been described in [LHL$^+$98, MHLL99, May99a].

In the last years, XML has been designed and accepted as *the* framework for semistructured data. Web browsers support XML sources presented via CSS or XSL stylesheets, plenty of products for handling XML data are available, both in the commercial and research area (here, commercial always also means research, and most research projects at least aim also at commercial use). There is a broad spectrum of research topics, ranging from language extensions (XML Schema, XPointer, XLink) over applications in active research areas (information integration, document processing and maintenance) to commercial applications (XML support for databases, electronic data interchange, business-to-business projects). Implementations of specialized languages are available for XML querying, e.g., XQL [Rob99], XSL(T) [XSL99], XML-QL [DFF$^+$99], and Lorel [AQM$^+$97] and YATL [CDSS99] coming from the semistructured area. For writing applications for creating and manipulating XML data, the dominating language is Java, regarding the DOM model as a data structure where applications are built on. A specialized "XML database language", such as SQL for the relational model, is still missing.

In our paper we focus on the database point of view: we integrate XML with the F-Logic data model, obtaining a comprehensive framework for XML which covers the current XML semantics and supports also a class hierarchy, nonmonotonic inheritance, schema reasoning, and comes with a logic programming style language. The approach allows for investigating conceptual extensions (e.g., schema reasoning) and data manipulation and integration issues. Based on a generic mapping of XML sources to F-Logic, extensions can be can be done in F-Logic, which allows for a very direct, intuitive programming. The FLORID system [FHK$^+$97][1], allows for practical experiments. In contrast, similar experiments with, e.g., Java-based implementations of XML/DOM would require to change the Java code of the system itself.

Another advantage of an approach which integrates XML into a general framework is that it is then possible to integrate XML sources with other sources, e.g., HTML Web sources and traditional databases.

Structure of the paper. In the next section, we give a short overview of the XML data model and a generic translation into an object-oriented model; also an illustrating example is introduced. Section 3 gives an overview of F-Logic as the underlying framework of the paper, and presents the FLORID system and its basic Web model. The translation of XML sources into an F-Logic database is described in Section 4, starting with a generic transformation which is then extended to include the interpretation of DTDs for deriving schema knowledge; also the querying functionality in F-Logic is sketched. Section 5 describes how SGML *documents* are also handled uniformly in this framework. In Section 6 we illustrate the data integration functionality of our approach and show how sources with different contexts are modeled by a further extension of the transformation. An overview of the SGML and XML transformations currently implemented in FLORID is given in Section 7. Section 8 gives several pointers to related work and concludes the paper.

2 XML

The XML data model is a hierarchical model. An XML *document* consists of a *root* element which then contains the whole document contents recursively structured as subelements which can also have attributes. In addition to the subelement relationship between elements, reference attributes can be defined using the id's of elements. A complete XML document comes with a DTD which describes its structure. The declaration of elements is given by

<!ELEMENT *elem_name* (*struct_spec*)>

where *struct_spec* describes the structure of the element (i.e., empty, plain data, or constructed from subelements according to *struct_spec* which is given in a regular-expression style). Additionally, attributes of elements are specified by

<!ATTLIST *elem_name* $attr\_name_1$ $attr\_type_1$ $attr\_constr_1$
: : :
$attr\_name_n$ $attr\_type_n$ $attr\_constr_n$ > .

Here, $attr\_type_i$ – roughly – distinguishes between value attributes and reference attributes, declares `id` attributes, and specifies if an attribute is scalar or multi-valued. Referencing is done via `ID` and `IDREF` attributes. $attr\_constr_i$ specifies if an attribute is mandatory or optional, also a simple kind of monotonic inheritance can be specified. We assume that the reader is familiar with the XML data model and the DTD syntax and give an exemplary XML instance below.

[1]available at `http://www.informatik.uni-freiburg.de/ dbis/florid/`.

Example 1 (Mondial Database: DTD and XML) *The following is an excerpt of an XML representation of the* MONDIAL *database [May99a]:*

```
<!ELEMENT mondial (country*,continent*,organization*)>
<!ELEMENT country (population, pop_growth, area, encompassed+,
                   language*, border*, city+)>
    <!ATTLIST country car_code ID #REQUIRED    name CDATA #REQUIRED
                      capital IDREF #REQUIRED  memberships IDREFS #IMPLIED
                      location CDATA #IMPLIED  industry NMTOKENS #IMPLIED>
<!ELEMENT population (#PCDATA)>
    <!ATTLIST population year CDATA #IMPLIED>
<!ELEMENT pop_growth (#PCDATA)>
<!ELEMENT encompassed (#PCDATA)>
    <!ATTLIST encompassed continent IDREF #REQUIRED>
<!ELEMENT border (#PCDATA)>    <!ATTLIST border country IDREF #REQUIRED>
<!ELEMENT language (#PCDATA)>  <!ATTLIST language name NMTOKEN #REQUIRED>

<!ELEMENT city (population*)>  <!ATTLIST city id ID #REQUIRED
                                              name CDATA #REQUIRED>
<!ELEMENT organization (member+)>
    <!ATTLIST organization abbrev ID #REQUIRED
                           name CDATA #REQUIRED
                           seat IDREF #IMPLIED>
<!ELEMENT member EMPTY> <!ATTLIST member type CDATA #REQUIRED
                                         country IDREFS #REQUIRED>
<!ELEMENT continent EMPTY>   <!ATTLIST continent name ID #REQUIRED>
```

```
<country car_code="CH" name="Switzerland" area="41290"
         capital="cty-CH-Bern" ... memberships="EU UN ..."
         location="Central Europe, east of France"
         industry="machinery chemicals watches textiles">
  <population year="1996">7207060</population>
  <pop_growth>0.59</pop_growth>
  <encompassed continent="Europe">100</encompassed>
  <language name="German">65</language>
  <language name="French">18</language>
  ...
  <border country="F">573</border>
  <border country="D">334</border>
  ...
  <city id="cty-CH-Bern" name="Bern">
    <population year="95">138600</population> </city>
  <city> ... </city>
   ...
</country>
<country car_code="F" name="France" capital="cty-F-Paris" ...>
   ...
</country>
...
<organization abbrev="EU" name="European Union" seat="cty-B-Brussels">
    <member type="full member" country="A D ..."/>
    <member type="candidate" country="AL BG ..."/>
    ...
</organization>
```

Generic Translation into an Object-Oriented Model. The XML/DOM data model directly induces a representation by complex objects:

- element types become classes,
- XML elements/nodes become objects,
- subelement relationships become object-valued properties,
- attributes also become properties; here, `ID`, `CDATA`, and `NMTOKEN` attributes become literal-valued properties, `IDREF` attributes become object-valued properties.

The *details* of the mapping now depend on the requirements and on the semantic concepts of the target framework. Possible criteria here are

- relevance of the ordering of subelements; i.e., document vs. database point of view,
- explicit representation of the original context, e.g., if several sources should be integrated,
- whether multivalued and scalar properties are distinguished, and
- schema modeling capabilities (especially, if XMLSchema is used instead of a DTD).

We will elaborate on these topics later. First, we give an overview of F-Logic as the target framework of our approach:

3 F-Logic and the FLORID System

F-Logic provides the integrated framework of our approach for representing Web sources in a unified object-oriented model, and for querying and data manipulation. A database is given as a set of facts, describing the properties of objects. The basic syntactic structure is as follows (we omit inheritable methods; for full F-Logic, the reader is referred to [KLW95]):

- The alphabet of an F-Logic language consists of a set $\mathcal{F}$ of *object constructors*, playing the role of function symbols, a set $\mathcal{P}$ of predicate symbols, a set $\mathcal{V}$ of variables, and several auxiliary symbols. Object constructors start with lowercase letters whereas variables start with uppercase ones.
- *id-terms* are composed as terms from object constructors and variables as usual.

Let O, C, D, Q_i, S, S_i, *SvM*, and *MvM* stand for id-terms.

- An *is-a assertion* is an expression of the form $O\!:\!C$ (object O is a member of class C), or $C :: D$ (class C is a subclass of class D).
- The following are *object atoms*:
  - $O[\mathit{SvM}@(Q_1,\ldots,Q_k) \to S]$: applying the *single-valued* method *SvM* with arguments $Q_1,\ldots,Q_k$ to O results in S,
    (for methods without arguments, we write $O[M \to S]$).
  - $O[\mathit{MvM}@(Q_1,\ldots,Q_k) \twoheadrightarrow \{S_1,\ldots,S_n\}]$: $\{S_1,\ldots,S_n\}$ are possible results when applying the *multi-valued* method *MvM* with arguments $Q_1,\ldots,Q_k$ to O.

  Similar, *signature atoms* describe the signatures of methods:
  - $C[M@(C_1,\ldots,C_k) \Rightarrow T]$ (and $C[M@(C_1,\ldots,C_k) \Rightarrow\!\!\!\!\!\Rightarrow T]$): applying the method M with arguments of classes $C_1,\ldots,C_k$ to an element of class C results in an element (or elements, respectively) of class T.
- An F-Logic *rule* is a logic rule of the form $\mathsf{head} \leftarrow \mathsf{body}$ over F-Logic's atoms, i.e., is-a assertions, object atoms, and signature atoms; an F-Logic *program* is a set of rules.

As shorthands, F-Logic allows the following constructs:

- Path expressions: The path expression $O.M@(\ldots)$ is *single-valued* and refers to the unique object S for which $O[M@(\ldots) \to S]$ holds, whereas $O..M@(\ldots)$ is *multi-valued* and refers to every S_i such that $O[M@(\ldots) \twoheadrightarrow \{S_i\}]$ holds.
- Molecules: atoms describing the same object can be grouped into molecules, i.e., $\mathsf{a{:}c[m \to v; n \twoheadrightarrow \{w_1,w_2\}]}$ for $\mathsf{a{:}c}$, $\mathsf{a[m \to v]}$ and $\mathsf{a[n \twoheadrightarrow \{w_1\}]}$, and $\mathsf{a[n \twoheadrightarrow \{w_2\}]}$.
- Molecules and path expressions can be nested.

F-Logic has been implemented in the FLORID system. The architecture of FLORID provides a generic Web interface for fetching Web documents into the system where F-Logic serves as an integrated data model and data manipulation language (cf. Figure 1). The Web interface implements the interpretation of specialized semantics of the individual -ML source languages, starting with ASCII and HTML/SGML parse-trees in FLORID 2.0 [LHL+98, May99b, MHLL99] and extended features for SGML and XML in version 2.7 described in this paper.

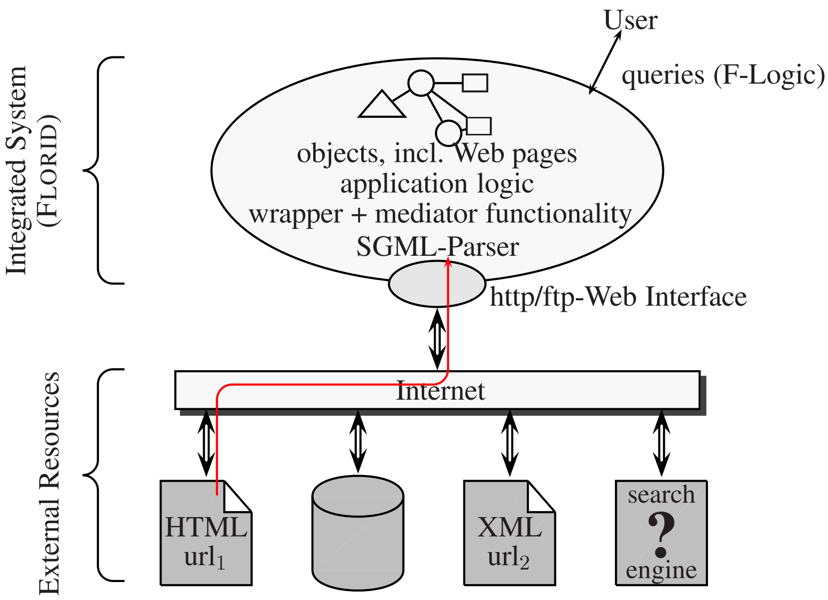

Figure 1: An Integrated Architecture for Web Access

4 Translating XML to F-Logic

In contrast to HTML sources, the intended representation of an XML source is not its parse-tree, but already an object-oriented model of the application domain which represents the contents from the *database* point of view. The model is completely based on the application-semantic tags and attribute names of the XML source. As stated above, elements define *objects* whose *properties* are described by subelements and attributes. The F-Logic data model does not distinguish between attributes and subelements – here, both are properties. Note that different "equivalent" XML representations (i.e., using differently structured DTDs) of the same contents will often yield the same object-oriented modeling.

4.1 The Basic XML Object-Oriented Model

XML without DTD/XMLSchema information. A simple generic translation is already possible, if there is no schema information on the data sources. Then, we cannot distinguish scalar properties from multivalued properties on the signature level (see Section 4.2):

- subelements are assumed to be multivalued (since it is now known how many subelements of some type may be given), and
- attributes are modeled as scalar properties: all attributes can only be handled as plain strings, i.e., `NMTOKENS` are not split, and `IDREF(S)` are neither split nor resolved.

In this case, an uninterpreted representation of the raw parsetree is obtained which has to be further wrapped by the application (`NMTOKEN(S)` and `IDREF(S)` attributes).

Example 2 (Basic Data Model) *For* MONDIAL*, we get, e.g. the following facts:*

mondial[country$\twoheadrightarrow${*switzerland*,*germany*,...}; organization$\twoheadrightarrow${*eu*,*un*,...}; ...].

switzerland:country[city$\twoheadrightarrow${*bern*,*geneva*,...}; capital$\rightarrow$"cty-CH-Bern"; area$\rightarrow$"41290";
population$\twoheadrightarrow${*pop-ch-96*}; pop_growth$\twoheadrightarrow${*pop-gr-ch*};
location$\rightarrow$"Central Europe, east of France";
industry$\rightarrow$"machinery chemicals watches textiles"; ...].

bern:city[name$\twoheadrightarrow${"Bern"}; ...]. *geneva*:city[name$\twoheadrightarrow${"Geneva"}; ...].
pop-ch-96:population[year$\rightarrow$"1996"; pcdata$\twoheadrightarrow${7207060}].
pop-gr-ch:pop_growth[pcdata$\twoheadrightarrow${0.59}].
eu:organization[abbrev$\rightarrow$"EU"; seat$\rightarrow$"cty-B-Brussels";
member$\twoheadrightarrow${*mem-eu-full*, *mem-eu-cand*, ...}].

mem-eu-full:member[type$\rightarrow$"full member"; country$\rightarrow$"A D ..."].
mem-eu-cand:member[type$\rightarrow$"candidate"; country$\rightarrow$"AL BG ..."].

Already on this stage, queries (and view definitions) can be stated using path expressions similar to both XQL [Rob99] and OQL [ASL89, CB00]. The structure of the path expressions with nested molecules is very similar to equivalent XQL queries[2]. In fact, querying the basic model uses only the following constructs (cf. Example 4):

- the path constructors ".." and "." for multivalued properties defined by subelements (which corresponds to the XQL "/"),
- the method applications "$\twoheadrightarrow$" and "$\rightarrow$" (which correspond to the use of subelement and attribute names in XPath filters), and
- the class-membership ":" (which corresponds to the XPath name() method).

Standard XML-parsing DTD information. Standard XML parsers (in our case, FLORID employs sp [Cla]) implicitly use the DTD if available for enriching their output by the types of the attributes (i.e., `CDATA`, `TOKEN`, (which includes both `NMTOKEN(S)` and `ID`), or `IDREF`). With this information, `TOKEN` attributes are split at the whitespaces, and `IDREF` attributes are split and resolved. In this case, we also assume all attributes to be potentially multivalued.

Example 3 *The attributes* location *and* industry *of* country *(declared as* `CDATA` *and* `NMTOKENS`*, respectively), are translated into*

switzerland:country[car_code$\twoheadrightarrow${"CH"}; name$\twoheadrightarrow${"Switzerland"};
location$\twoheadrightarrow${"Central Europe, east of France"};
industry$\twoheadrightarrow${"machinery", "chemicals", "watches", "textiles"}].

Also, `IDREF(S)` *attributes are resolved; e.g., members of organizations:*

eu:organization[abbrev$\twoheadrightarrow${"EU"}; member$\twoheadrightarrow${*mem-eu-full*, *mem-eu-cand*, ...}].
mem-eu-full:member[type$\twoheadrightarrow${"full member"}; country$\twoheadrightarrow${*austria*,*germany*,...}].
mem-eu-cand:member[type$\twoheadrightarrow${"candidate"}; country$\twoheadrightarrow${*albania*,*bulgaria*,...}].

Since F-Logic allows for joins and variables on arbitrary positions, already queries in this basic translation are more expressive than XQL, also smoothly integrating navigation along references.

Example 4 (F-Logic for XML querying) *The following queries to the* MONDIAL *database give an overview of F-Logic as an XML querying language.*

1. *(select and combine data from different source elements; joins and references)*
   - *for every country, give its name, population, and population of its capital.*

[2] an implementation using an extended XQL syntax is in preparation.

?- _C:country[name↠N; population↠_[pcdata↠P1]]..capital..population[pcdata↠P2].

- *for every country, give its name and the capitals of its neighbors.*

  ?- _C:country[name↠N]..border..country..capital[name↠N2].

2. *(not-exists)*
   - *all countries which have no neighbors:*

     ?- C:country, not C..border.

   - *all countries which are not member of any organization where the USA are a member.*

     ?- C:country, U:country[car_code↠"USA"], not _:org[members↠{U, C}].

Comparison with XML/XQL. A striking "feature" of XML/XPath/XQL is that, *even if a DTD is used*, the data model does obviously not go beyond the simple stage, i.e., XPath/XQL is *not aware* of `NMTOKENS` or `IDREF(S)` declarations:

- //country[@car_code="CH"]/@industry[1] = "machinery chemicals watches textiles", i.e., the `NMTOKENS` are not split,
- //organization[abbrev="EU"]/member/@country[1] = "A D I L NL ...", i.e., the `IDREFS` are not split, but
- id(//organization[abbrev="EU"]/member/@country[1]) yields the country elements for Austria, Germany, Italy etc., i.e., when applying the id function; they are split, *and*
- if we add appropriate elements with id's "machinery", "chemicals", etc., id(//country[@car_code="CH"]/@industry) yields these objects! – thus, `NMTOKENS` can also be "dereferenced" in XPath.

In the next section, a transformation into a more detailed object-oriented model based on a DTD (or an XMLSchema declaration), distinguishing scalar and multivalued properties is described.

4.2 Exploiting the DTD

When the DTD is explicitly parsed, additional meta/schema knowledge can be extracted – describing properties of element *types*: Every element type e has a property contents (elements, pcdata, mixed, or empty), and every attribute a or subelement name s defines an object x such that e[attribute@(a)$\rightarrow x$] or e[subelement@(s)$\rightarrow x$] which has properties type (an elementtype for subelements, and literal, id, or ref, for attributes), card (scalar or multivalued), and minCard (0 (mandatory) or 1 (optional)).

Example 5 (Mondial Database: Schema) *The DTD of the* Mondial *database given above yields the following schema knowledge:*

```
t(mondial):elementtype[contents -> elements]
  t(mondial).sub@(country)[type -> t(country); card -> mvd; minCard -> 0].
  t(mondial).sub@(organization)[type -> t(organization); card -> mvd; minCard -> 0].
  t(mondial).sub@(continent)[type -> t(continent); card -> mvd; minCard -> 0].
t(country):elementtype[contents -> elements]
  t(country).sub@(population)[type -> t(population); card -> sc; minCard -> 1].
  t(country).sub@(pop_growth)[type -> t(pop_growth); card -> sc; minCard -> 1].
  t(country).sub@(encompassed)[type -> t(encompassed); card -> mvd; minCard -> 1].
  t(country).sub@(language)[type -> t(language); card -> mvd; minCard -> 0].
  t(country).sub@(border)[type -> t(border); card -> mvd; minCard -> 0].
  t(country).sub@(city)[type -> t(city); card -> mvd; minCard -> 0].
  t(country).attr@(car_code)[type -> id; card -> sc; minCard -> 1].
  t(country).attr@(name)[type -> literal; card -> sc; minCard -> 1].
  t(country).attr@(capital)[type -> ref; card -> sc; minCard -> 1].
```

```
  t(country).attr@(area)[type -> literal; card -> sc; minCard -> 1].
  t(country).attr@(memberships)[type -> ref; card -> mvd; minCard -> 0].
  t(country).attr@(location)[type -> literal; card -> sc; minCard -> 0].
  t(country).attr@(industry)[type -> literal; card -> mvd; minCard -> 0].
t(population):elementtype[contents -> pcdata].
  t(population).attr@(year)[type -> literal; card -> sc; minCard -> 0].
t(pop_growth):elementtype[contents -> pcdata].
t(encompassed):elementtype[contents -> pcdata].
  t(encompassed).attr@(continent)[type -> ref; card -> sc; minCard -> 1].
t(language):elementtype[contents -> pcdata].
  t(language).attr@(name)[type -> literal; card -> sc; minCard -> 1].
t(border):elementtype[contents -> pcdata].
  t(border).attr@(country)[type -> ref; card -> sc; minCard -> 1].
t(city):elementtype[contents -> elements].
  t(city).sub@(population)[type -> t(population); card -> mvd; minCard -> 0].
  t(city).attr@(name)[type -> literal; card -> sc; minCard -> 1].
  t(city).attr@(id)[type -> id; card -> sc; minCard -> 1].
t(organization):elementtype[contents -> elements].
  t(organization).sub@(member)[type -> t(member); card -> mvd; minCard -> 1].
  t(organization).attr@(abbrev)[type -> id; card -> sc; minCard -> 1].
  t(organization).attr@(name)[type -> literal; card -> sc; minCard -> 1].
  t(organization).attr@(seat)[type -> ref; card -> sc; minCard -> 0].
t(member):elementtype[contents -> empty].
  t(member).attr@(country)[type -> ref; card -> mvd; minCard -> 1].
  t(member).attr@(type)[type -> literal; card -> sc; minCard -> 1].
t(continent):elementtype[contents -> elements].
  t(continent).attr@(name)[type -> id; card -> sc; minCard -> 1].
```

Note that the signature information which is extracted from the DTD does not include the target classes of reference attributes (which is not known to the DTD).

4.3 Deriving the F-Logic Signature

The F-Logic data model provides *signature atoms* for describing the schema of a database which are derived from the above schema knowledge. In contrast to the basic model presented in Section 4.1, cardinalities are considered now. XML element types simultaneously define classes (collecting the elements) and methods (of the parent element, resulting in the appropriate subelements) which result in that class:

country[city⇒⇒city]. *% method city applied to objects of class*
% country results objects of class city
switzerland[city→→{*bern*,*geneva*,…}]. *bern*:city. *geneva*:city.

`CDATA`, `NMTOKEN(S)`, and `ID` attributes define methods with literal results (i.e., strings and numbers):

country[name⇒literal; area⇒literal; industry⇒⇒literal].
switzerland[name→"Switzerland"; area→"41290"; industry→→{"machinery", …}].
organization[abbrev⇒literal]. *eu*[abbrev→"EU"].

References. For reference attributes (`IDREF(S)`), we pay for the XML decision that references are taken from a global set of `ID`s without specifying which should be the type of the referenced element (e.g., the DTD does not tell that the capital must reference a city – referencing an organization would also be valid). Here, only dependent on the XML instance, it can be derived, which classes (i.e., in XML terms, element types) are referenced. In our example, the signatures of the reference attributes are as follows, specifying the class of the referenced objects:

country[capital⇒city; memberships⇒⇒organization]. org[seat⇒city].
switzerland[capital→*bern*; memberships→→{*eu*, *un*,…}]. *eu*[seat→*brussels*].

Literal Values. The only literal type known in XML is string, occurring as `PCDATA`, `CDATA`, `NMTOKEN`, or `ID`; especially, numeric values are not supported. In the derived F-Logic model, we distinguish numeric values from strings. On the data level, every string representing a number is automatically converted to a number (for applying string operations, numbers are silently casted into strings, so there is no problem with this solution). For the signature level, depending on the XML instance, the result datatype is numeric if *all* values are numbers, string otherwise:

country[name⇒string; area⇒numeric].

Note that with string::literal and numeric::literal, this does not conflict with the above country[name⇒literal; area⇒literal]. In this aspect, proposals like XMLSchema [XML99] which distinguish literal data types will bring a significant progress.

Text and Mixed Contents. XML distinguishes between elements (i.e., nodes of the form <*tagname*> ... </*tagname*>) and their `PCDATA` contents. In our translation, `PCDATA` elements without attributes, are directly mapped to their literal value, e.g.,

country[pop_growth⇒numeric].   *switzerland*[pop_growth→0.59].

`PCDATA` elements with attributes behave as *annotated Literals*, i.e., they are modeled as complex objects with a text property

country[population⇒population].   population[year⇒numeric; pcdata⇒numeric].
switzerland[population→*pop-ch-96*].   *pop-ch-96*:population[year→1996; pcdata→7207060],

but act as their literal values when used in string/numeric comparisons, arithmetics, or in the output. Thus, the user can treat country..population like a numerical value which is additionally annotated with a year attribute:

Example 6 (Annotated Literals) *country..population is an annotated Literal:*

?- _:country[car_code→"CH"; population↠P].
P/7207060
?- _:country[car_code→"CH"; population↠P[year→Y]].
P/7207060   Y/1996
?- _:country[car_code→"CH"]..population[year→1996] > 7000000.
true

Although, 7207060[year→1996] *does* not *hold.*

4.4 Querying Document Contents and Structure

Queries on the database resulting from the refined translation have to consider the distinction between scalar and multivalued methods. As a fully equipped deductive database language, F-Logic also provides the usual aggregation operators and allows for view definitions.

Example 7 *The following examples continue the ones given in Example 4.*

- *(Aggregation)*
  - *for every country and year, give its population and the number of people living in cities in this country.*

    ?- C:country[name→N; population↠P1[year→Y]],
       S=sum{P [C,Y]; C..city[population↠P[year→Y]]}.

  - *all political organizations which have their seat in the country which is the largest among all members.*

    ?- O:org[name→N; seat→_SC], _C:country[city↠_SC; area→_A],
       _A=max{A [O]; O..members[type→"member"]..country[area→A]}.

- *(relational division) give all political organizations which have at least one member on each continent.*

  ?- O:org, _N1 = count{CT [O]; O..members..country..encompassed[continent→CT]},
  _N2 = count{CT; CT:continent}, _N1 = _N2.

  Equivalently, the query can be answered using views in a stratified program:

  is_on(O,CT) :- (O:org)..members..country..encompassed[continent→CT].
  is_not_on(O,CT) :- O:org, CT:continent, not is_on(O,CT).
  ?- O:org, not is_not_on(O,_).

- *(transitive closure) give all pairs (c_1, c_2) of countries s.t. c_2 is reachable from c_1 by land (transitive closure of* border *relationship).*

  C[reachable ↠C2] :- C..border[country→C2].
  C[reachable ↠C2] :- C..reachable..border[country→C2].
  ?- C[reachable ↠C2].

The F-Logic representation of XML sources also allows for querying the document structure:

Example 8 *The following queries ask for all properties of the class* country *which result in non-literal objects, and for all properties of any class which result in a* country*:*

?- country[M⇒⇒C], not C = literal.
?- C[M⇒⇒country].

5 XML Parse-Trees and SGML Sources

The object-oriented translation presented above does not contain information about the original parse-tree, i.e., the ordering is (intentionally) lost. When accessing SGML sources from the *document* point of view, the intended model is simply the parse-tree model. Also, when accessing XML documents where the ordering of subelements is relevant, suitable information about the parse-tree has to be added. We illustrate the representation of parse-trees from the document point of view for a "typical" SGML source: Different kinds of parse-nodes are induced by the tags, e.g., sections, subsections, paragraphs, tables, lists etc.

Every element name <name> of the SGML document defines a class name which contains all elements of the SGML document with this tag; name – as a class – is a subclass of the class node, name – now as an object – becomes an element of the class tag. Instances of node have attributes and subelements (children). Children are generically represented according to the signature

node[child@(tag,int)⇒node]

where the tag position holds the subelement's tag, and the int position enumerates the children of each node. Attributes also define properties of their node:

node[attribute⇒string].

The possible values of tag and attribute are given by the DTD or by the document instance (hopefully, equivalent – if the document is valid).

Example 9 (SGML parse-tree) *We illustrate the schema data for a simple sectioning markup. Let*

section$_3$ = (<section id="sec-architecture" author="X.Y.">
header$_3$ *para*$_{31}$ *figure*$_{31}$ *para*$_{32}$ . . . </section>) ,
header$_3$ = (<header> The Architecture </header>) ,
para$_{3i}$ = (<paragraph> . . . </paragraph>) ,
figure$_{31}$ = (<figure id="fig-architecture" src="pictures/architecture.ps"/>) .

Then, the following F-Logic representation of the parse-tree is generated:

section:tag. section::node. header:tag. header::node.
paragraph:tag. paragraph::node. figure:tag. figure::node.
section$_3$:section[author→"X.Y."; id→"sec-architecture";
child@(header,1)→*header*$_3$; child@(par,2)→*para*$_{31}$;
child@(figure,3)→*figure*$_{31}$; child@(par,4)→*para*$_{32}$].

figure$_{31}$:figure[id="fig-architecture"; src="pictures/architecture.ps"].

We can also define

Element[Subelement↠X] :- Element:node, Element[child@(Subelement,N)→X]

yielding

section$_3$[header↠{*header*$_3$}]

for enabling navigation by tag names as presented before for XML sources.

Here, it is a crucial feature of F-Logic that objects can simultaneously occur as objects, classes and methods (see, e.g., header above). This allows for powerful schema integration rules based on meta-reasoning.

Adding parse-tree information to the database view of XML. Optionally, the parse-tree information can also be added to the XML transformation as described in Section 4 (this does not add any new objects, only new links).

Example 10 *The parse-tree links add, e.g., the following facts to the database:*

eu:organization[abbrev→"EU"; name→"European Union";
child@(member,1)→*mem-eu-full*; child@(member,2)→*mem-eu-cand*].

6 Information Integration and Context Information

In contrast to pure XML handling environments, the integration of XML with an existing general-purpose object-oriented database system provides useful additional functionality:

- support of a generic Web model (e.g., extracting the Web skeleton from HTML sources),
- integration of XML/SGML sources with arbitrary HTML and text sources from the Web,
- exploitation of the Web-exploring capabilities of FLORID for searching for potential sources, and
- a complete environment of a deductive programming language.

Integration of XML documents. Already when only XML sources are concerned, integration is in general not a simple combination of XML documents but requires at least partial restructuring and generation of intermediate results: sources are based on different DTDs and often also use different ontologies. Here, XML querying and transformation languages such as XSL(T) [XSL99], XQL [Rob99], and XML-QL [DFF+99] have a severe drawback: They do not allow for view definitions. In contrast, our rule-based approach allows for arbitrary intermediate computations and definitions of views which can be specified declaratively.

For information integration, objects originating from different sources (*contexts)* are considered. Then, it is often important for the integration process to know the origin of the stored information. For this, all information can optionally be parameterized by the source context (e.g., a database identifier which can also have properties as an object of its own):

- all classes defined by tags are associated with a context object,
- all methods defined by subelements or attributes become parameterized with the context,
- all links representing the parse-tree are parameterized with the context.

Example 11 *The information of the mondial database is tagged with its origin as follows:*

eu:(mondial.organization)[abbrev@(mondial)↠"EU";
member@(mondial)↠{*mem-eu-full*, *mem-eu-cand*, ...};
child@(member,1,mondial)→*mem-eu-full*;
child@(member,2,mondial)→*mem-eu-cand*].

mem-eu-full:(mondial.member)[type@(mondial)↠{"full member"};
country@(mondial)↠{*austria*,*germany*,*france*,*italy*,...}

mem-eu-cand:(mondial.member)[type@(mondial)↠{"candidate"};
country@(mondial)↠{*albania*,*bulgaria*,...}

If different contexts are used, the context objects can be provided with additional information, and also information for context changing can be specified (cf. [GBMS98]).

Example 12 (Contexts) *Consider an enterprise which intends to integrate databases from several countries. Then, the contexts are, e.g.,* germany *and* usa*, with additional currency information (wrt. a reference currency). Also, some context-dependent information is given (extracted from, e.g., XML sources):*

context[currency⇒currency].   % Signature of currency method
currency[exchange@(currency)⇒numeric].
germany:context[currency→*euro*[exchange@(*dollar*)→0.90]].
usa:context[currency→*dollar*[exchange@(*dollar*)→1]].

ptcruiser:(*usa*.car)[price→30000]; *smart*:(*germany*.car)[price→16000];

With this, a currency exchange method can be defined generically for all contexts:

context[currency_xc@(context,numeric)⇒numeric].
From_c[currency_xc(To_c,VFrom)→VTo] :-
From_c:context.currency[exchange@(*dollar*)→VX],
To_c:context.currency[exchange@(*dollar*)→VY], VTo = VFrom/VY * VX,

X[price@(C2)→Q] :- X:(C1.car)[price→P], C1[currency_xc(C2,P)→Q].

Then, we can, e.g., derive smart[price@(usa)→14400].

In course of the integration of overlapping sources, the objects of the target model are constructed by fusing objects of the individual sources, thereby accumulating their properties:

Example 13 *Assume two data sources* mondial *and* cia *which describe countries. Then, the countries can be integrated as follows (equijoin on C):*

C1=C2, C1:country[name→N; code→C; area→A; gdp→G] :-
C1:(mondial.country)[car_code@(mondial)→C; area@(mondial)→A],
C2:(cia.country)[name@(cia)→N; code@(cia)→C; gdp@(cia)→G].

For integration, ontology-based reasoning in combination with signature metareasoning can be applied. Also algorithms for finding overlapping fragments of different sources can be employed for automatical integration.

Integration of SGML and XML Sources. F-Logic rules allow for deriving and adding new facts to the database. Then, information can be cleaned and restructured, and information from several HTML and XML sources can be integrated. In contrast to XML sources which are directly mapped to object-oriented representations of their application domains, the SGML documents are only repesented by their parse-trees which first have to be transformed into a semantical model. Depending on how much information has to be extracted from the SGML sources, there are two alternatives:

- a wrapper for the SGML document is specified which translates the SGML parse-tree into a standalone object-oriented model. Then, this model is integrated with the object-oriented models of the XML sources.
- some rules are specified which extract the required information from the SGML parse-tree and directly add it to one of the object-oriented models of the XML sources.

In both cases, our deductive framework allows for a declarative specification of rules. Here, the full functionality of the FLORID system, including generic wrapping rules for tables, lists, etc. as presented in [MHLL99], can be used.

7 Overview of Implemented Transformations

An important issue when accessing and interpreting documents is if the source should be regarded as a document with its parse-tree or as a database. As already shown, for XML sources there are again different possibilities how to represent the document in the internal model, depending on what is provided with the XML source, and what the user intends to do with the XML source(s). The F-Logic data model covers both the DTD or XMLSchema (by signature atoms) and the XML instance; it even holds more explicit semantics since the target classes of reference attributes are known. Additionally, it allows to extend the data with a class hierarchy and inheritance. FLORID provides the following functionality:

- "document" view: the source is interpreted as a sequentially structured parse-tree where the ordering of items is relevant and references merely are cross-references as known from documents,
- "database" view: the source is seen as an (object-oriented) database where objects and their properties are described. Here, references are seen as object-valued properties,
- automatical resolving of references inside the source (only possible when a DTD for distinguishing `ID` and `IDREF` attributes is available),
- if a DTD to the document is provided (an declared in the XML prolog), it can be used,
- accessing several documents using several contexts.

The following Web access methods which implement the features described above are provided by the current Web interface:

u**.get:** The ASCII representation of a Web page is fetched into the system.

u**.parse@(sgml, *references*, *context*):** the document accessible via u is regarded as an SGML document, i.e., the parse-tree is generated. If *references* is true, references are resolved automatically. If a *context* is given (nil otherwise), all atoms are augmented with the context (cf. Section 6).

u**.parse@(xml, *references*, *ordering*, *schema*, *context*):** the document accessible via u is regarded as an XML source from which an object-oriented representation is generated. If *references* is true, then references are resolved automatically. If *ordering* is true, the parse-tree is embedded into the object-oriented model (cf. Example 10). The *schema* parameter selects if scalar and multivalued properties should be distinguished by the DTD given in the XML source or by another DTD or XMLSchema file (given by an url), or if a resident F-Logic signature should be used. If a *context* is given (nil otherwise), all atoms are augmented with the context (cf. Section 6).

8 Related Work and Conclusion

Several XML querying and transformation languages have already been mentioned in the introduction: XQL is a restricted pure querying language based on path expressions which also provides the basis for the transformation language XSL(T) [XSL99]. XML-QL [DFF+99] is another querying/transformation language based on matching of XML-style patterns. Both directly produce XML output, lacking typical database functionality like view definitions and updates. QUILT [CRF00] is a recent proposal for a comprehensive XML query language, but it is not yet implemented. There are also several XML database-like systems, e.g., TAMINO [Sof00] and eXcelon [eC00] which provide comfortable user-interfaces for querying and design of XML databases. The WIDL framework [All97] provides a language for specifying mappings from HTML and XML sources into XML. [BGL+99] presents an XML environment which is especially tailored to the integration of XML sources.

We presented a transformation approach from SGML and XML into an integrated object-oriented data model. Depending on the source, the Web interface transforms the document into a parse-tree representation or directly into an object-oriented representation. The transformation of an XML document into a graph-like F-Logic database can be done in a generic way, retaining all of the relations between elements (which are turned into objects). The F-Logic model "contains" the hierarchical model of the XML document and makes it accessible to the F-Logic querying and data manipulation language. SGML sources are represented by their parse-tree and can be wrapped by F-Logic rules. Additionally, the whole functionality as a general system for data extraction and integration from the Web can be used. Providing a powerful data manipulation and querying language, our system can also compete with XML-only approaches.

Providing a declarative language at the user level (in contrast to most other systems which are Java-based), the system can easily be extended to XML extensions such as XMLSchema and XPointer: extensions can be implemented and investigate prototypically by F-Logic rules which are later implemented as built-ins into the system. E.g., the DTD and XML "wrappers" have first been developed and evaluated as F-Logic programs solely based on the original HTML parse-tree before they were "hard-coded" on the C++ level.

References

[All97] C. Allen. WIDL, Application Integration with XML. *World Wide Web Journal*, 2(4):229–248, 1997.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *Intl. Journal on Digital Libraries (JODL)*, 1(1):68–88, 1997.

[ASL89] A. Alashqur, S. Su, and H. Lam. OQL: A query Language for Manipulating Object-Oriented Databases. In P. M. Apers and G. Wiederhold, editors, *Intl. Conference on Very Large Data Bases (VLDB)*, pp. 433–442, 1989.

[BGL+99] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-Based Information Mediation with MIX. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 1999.

[CB00] R. Cattell and D. Barry. *The Object Database Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

[CDSS99] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your Mediators need Data Conversion. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 1999.

[Cla] J. Clark. SP: An SGML System Conforming to International Standard ISO 8879 – Standard Generalized Markup Language. `http://www.jclark.com/sp`.

[CRF00] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *WebDB 2000*, pp. 53–62, 2000.

[DFF+99] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. In *8th. WWW Conference*. W3C, 1999. World

Wide Web Consortium Technical Report, NOTE-xml-ql-19980819, `www.w3.org/TR/NOTE-xml-ql`.

[eC00] eXcelon Corporation. XML Application Development using eXcelon. In *Intl. Conference on Extending Database Technology*, Springer LNCS 1777, 2000. `http://www.odi.com/`.

[FHK+97] J. Frohn, R. Himmeröder, P.-T. Kandzia, G. Lausen, and C. Schlepphorst. FLORID: A Prototype for F–Logic. In *Intl. Conference on Data Engineering (ICDE)*, 1997.

[GBMS98] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Transactions on Information Systems (TOIS)*, 1998.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2), 1997.

[KIF98] Knowledge Interchange Format (KIF); also ANSI Draft NCITS.T2/98-004. `http://logic.stanford.edu/kif/kif.html`, 1998.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.

[LHL+98] B. Ludäscher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective. *Information Systems*, 23(8):589–612, 1998.

[Man98] F. Manola. Towards a Web Object Model. `http://www.objs.com/OSA/wom.htm`, 1998.

[May99a] W. May. Information Extraction and Integration with FLORID: The MONDIAL Case Study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999. `http://www.informatik.uni-freiburg.de/ may/Mondial/`.

[May99b] W. May. Modeling and Querying Structure and Contents of the Web. In *Workshop on Internet Data Modeling (IDM'99), Proc. DEXA 99 Workshops*, pp. 721–725. IEEE Computer Science Press, 1999.

[MHLL99] W. May, R. Himmeröder, G. Lausen, and B. Ludäscher. A Unified Framework for Wrapping, Mediating and Restructuring Information from the Web. In *International Workshop on the World-Wide Web and Conceptual Modeling (WWWCM)*, Springer LNCS 1727, pp. 307–320, 1999.

[Rob99] J. Robie. XQL (XML Query Language). `http://www.metalab.unc.edu/xql/xql-proposal.html`, 1999.

[Sof00] Software AG. Tamino – An Internet Database System. In *Intl. Conference on Extending Database Technology*, Springer LNCS 1777, pp. 383–387, 2000. `http://www.tamino.com/`.

[XML99] XML Schema Part 2: Datatypes. `http://www.w3.org/TR/xmlschema-2`, 1999.

[XSL99] XSL Transformations (XSLT). `http://www.w3.org/TR/xslt`, 1999.