Revised from Version in: Proc. 14th Workshop Logische Programmierung, W"urzburg, Jan. 2000

Relationships between Logic Programming and XML*

Harold Boley
Deutsches **F**orschungszentrum für **K**ünstliche **I**ntelligenz GmbH
Email: boley@dfki.de

Abstract

Mutual relationships between logic programming and XML are examined. XML documents are introduced as linearized derivation trees and mapped to Prolog structures. Conversly, a representation of Herbrand terms and Horn clauses in XML leads to a pure XML-based Prolog (XmlLog). The XML elements employed for this are complemented by uses of XML attributes like id/idref for extended logics. XML's document type definitions are introduced on the basis of ELEMENT declarations for XmlLog and ATTLIST declarations for id/idref. Finally, queries in languages like XQL are treated functional-logically and inferences on the basis of XmlLog are presented. All concepts are explained via knowledge-representation examples.

1 Introduction

The simplicity of Web-based data exchange is beneficial for nonformal, semiformal and formal documents. For formal specifications and programs the Web permits a distributed development, usage and maintenance. Logic programming (LP) has the potential to serve as a standard language for this. However, the World Wide Web Consortium (W3C) [http://www.w3.org/] has enhanced HTML (for nonformal and semiformal documents) into the Extensible Markup Language (XML) [Har99] for semiformal and formal documents.

Thus the issue of the relationships between XML and LP arises. Will logic programming have the chance, despite, or perhaps, precisely because of XML, to become a "Web technology" for formal documents? Could the HTML-like syntax of XML forthis be replaced by a Prolog-like syntax, or could it be edited or presented - over a standardized style sheet - in such a Prolog syntax? Is SLD resolution a suitable starting point for the interpreter semantics of an XML query language like XQL? [http://www.w3.org/TandS/QL/QL98/pp/xql.html] or should an LP-oriented, inferential, query language be developed in the form of an XML-based Prolog ? In the following text, such questions will be dealt with, and possible interplays between XML and LP, in both directions, will be discussed.

The already foreseeable success of XML as a 'universal' interchange format for commercial practice (incl. E-commerce) can also be viewed as a success of the declarative representation technique; as it was proposed, in a somewhat different form by logic programming. Similarities

*I would like to thank Ulrich Geske and the programm committee of the 14th Workshop on logic programming for the invitation. For the proof-reading of a first version my thanks go to Andreas Kl"uter and Michael Sintek. Preliminary translation to English by Christy Anderson, Brice Kamga, and Kushalappa Paleyanda..

and differences between these declarative paradigms will be later elaborated upon. Let us here state two success factors of XML: (1) A sufficient compatibility of XML with the already widely-used HTML, e.g. through XHTML. (2) The XML standardization through the W3C, and, consequently, the building in of XML tools into current browsers like Internet Explorer and Netscape/Mozilla, and, hence, the increase of the number of XML documents in the Web. The ISO standard of Prolog [DEDC96] can hardly attain the same broad effect because of its missing Web/HTML orientation, but it has a more strongly focused semantics. Altogether, therefore, it would be interesting to contrast the two standards in a more detailed manner than the beginning,that can be made in the following. Through that, the path for (partial) Prolog-XML translators, interfaces, etc. would be paved in both directions, with (essential) advantages of both paradigms being combined.

Beyong all startup euphoria, XML offers new general possibilities from which logic programming can also benefit: (1) Definition of self-describing data in an internationally standardized, non-proprietary format (2) Structured exchange of data and Knowledge between very diverse enterprises (3) Integration of information from various sources into unified documents. Furthermore, as already discussed in [Bol97] (for the XML predecessor SGML), XML is the most suitable language for logical knowledge bases in the Web. Additional LP uses of XML would be the exchange of knowledge bases between different logic languages as well as between LP-knowledge bases and databases, application systems, etc. Even the transformation or compilation of logic source programs could be done on the basis of XML markups and annotations. These and similar possibilities will be expanded in the following.

The current text uses concrete comparisons of LP and XML examples in order to work out (syntactic) differences as well as (semantic) similarities. The text is directed primarily to readers who wish to understand basics of XML via LP and deductive or relational database. But also in the converse direction aspects of LP appear in a new light via XML. The sections after this one (1.) are constructed as follows: 2. Firstly we introduce 'elementary' XML and show an element representation by means of Prolog structures; 3. Thereafter we look into the reverse direction, demonstrating how Herbrand terms are representable as XML elements; 4. Building on that we deal with an XML representation of Horn clauses (pure XmlLog). 5. Subsequently, XML attributes like `id/idref`are employed for extended logic. 6. After this, XML's document type definition are used for logic language like XmlLog. 7. Then, XML query languages like XQL and inferential logic programs are compared. 8. In conclusion, we discuss some similar approaches and give an outlook.

2 XML Elements as Ground Structures

XML documents consist of (nested) elements. Each element is a sequence of the form `<`*tag*`>` . . . `</`*tag*`>`, i.e. the content is enclosed by a *tag*-'colored' start- and end-brackets ". . .". This content may consist of text or again of elements. Consequently, a 'start-tag' and 'end-tag' can be used for the total markup of textual content, which may contain any desired well-detailed markup. With the increase of XML markup in a document its 'formality' will also increase in the sense of the spectrum mentioned at the beginning of section 1 .

Let us next consider an **informal** minimal markup with a single tag pair `<sentence>` . . . `</sentence>`:

```
<sentence> Onoffbook sold 12417 copies of XML4You online </sentence>
```

Through rough parsing one obtains a **semiformal** <triple> element, whose <predicate> and <object> subelements contain unanalyzed text:

```
<triple>
  <subject> Onoffbook </subject>
  <predicate> sold online </predicate>
  <object> 12417 copies of XML4You </object>
</triple>
```

Finally, a complete, context-free parsing yields a (syntactically) **formal** document with the following <s>-rooted element nesting:

```
<s>
  <np>
    <noun> Onoffbook </noun>
  </np>
  <vp>
    <vgroup> <verb> sold </verb> <adverb> online </adverb> </vgroup>
    <np>
      <ngroup>
        <card> 12417 </card>
        <noun> copies </noun>
      </ngroup>
      <pp>
        <prep> of </prep>
        <np>
          <noun> XML4You </noun>
        </np>
      </pp>
    </np>
  </vp>
</s>
```

A document, therefore, is structured through XML in the form of a parsing tree or a derivation tree, whereby each element <*tag*> . . . </*tag*> is a *tag*-marked node (a nonterminal) which has ordered, directed edges with direct subelements, appearing in ‘‘. . .’’ (shown here through indent). The texts appearing in‘‘. . .’’ are the leaves of the derivation tree. In the normal case,each XML element contains either only subelements (corresponding to nonterminals) or only text (corresponding to a ‘token’ of terminals).

Apparently, such an XML document is - in Prolog - comprehendable as (a variable-free,i.e) ground structure *tag*(., ., .) , provided that texts on the leaves can be represented as individual constants. Thereby *tag* becomes constructor, and the contents ‘‘. . .’’ becomes correspondingly transformed arguments ‘‘., ., .’’. Without going into the exact ‘‘character escape’’-conventions of XML and Prolog, we represent XML texts as Prolog strings ; after complete analysis we use Prolog individual constant (with lower case letters).

Three steps of parsing example in Prolog therefore lead to following ground structures:

```
sentence("Onoffbook sold 12417 copies of XML4You online")

triple(
  subject("Onoffbook"),
  predicate("sold online"),
  object("12417 copies of XML4You")
      )

s(
  np(noun(onoffbook)),
  vp(
    vgroup(verb(sold),adverb(online)),
    np(ngroup(card(12417),noun(copies)),
       pp(prep(of),np(noun(xml4you))))
    )
 )
```

Our linguistic analysis proceed from this general syntax, to special semantics, e.g. for discourse domain ‘‘E-commerce’’. Analysts of specific discourse domains often jump directly to such semantics, without the earlier discussed natural language (XML-) part.

Thus in XML use a (semantically) formal document with four <sales> subelements, corresponding to domains of a relational <sales> database table:

```
<sales>
  <channel> online </channel>
  <company> Onoffbook </company>
  <item> XML4You </item>
  <quantity> 12417 <quantity>
</sales>
```

This then in Prolog leads to the following ground structure:

```
sales(
  channel(online),
  company(onoffbook),
  item(xml4you),
  quantity(12417)
     )
```

3 Herbrand Terms as XML Elements

For the representation of Herbrand terms (individual constants, logic variables and structures) in XML, we use, in essence, <ind>, <var> and <struc> elements.

Like all elements, an *individual constant* in XML must be explicitly labeled as such through markup, here an <ind> *element* of the form <ind> ... </ind>. Thus the Prolog individual constant channel-tunnel for the channel tunnel between Great Britain and France, becomes the XML element <ind> channel-tunnel </ind>.

In Prolog, instead of an individual symbol, a *(ground) structure* like undersea-connection(britain,france) can also be used for the denotation of an individual. One can also view this as a 'record' with components for the connected lands. In XML, a structure becomes a <struc>- *element* with an embedded element for constructor, followed by elements for its argument terms, here two individual constants:

```
<struc>
  <constructor> undersea-connection </constructor>
  <ind> britain </ind>
  <ind> france </ind>
</struc>
```

Because the XML markup makes the syntax of the original Prolog structure explicit, this can be regained from it as well as a Prolog structure can be constructed (as in paragraph 2), whose syntax is described as follows:

```
struc(
   constructor(undersea-connection),
  ind(britain),
  ind(france)
 )
```

The original structure is already nested as, for example,

```
service-tunnel(
  undersea-connection(
    britain,
    surrounded-country(belgium,luxembourg,germany,switzerland,italy,spain)))
```

then this leads to embedded <struc>-elements as in

```
<struc>
  <constructor> service-tunnel </constructor>
  <struc>
    <constructor> undersea-connection </constructor>
    <ind> britain </ind>
    <struc>
      <constructor> surrounded-country </constructor>
      <ind> belgium </ind>
      <ind> luxembourg </ind>
      <ind> germany </ind>
      <ind> switzerland </ind>
      <ind> italy </ind>
```

```
      <ind> spain </ind>
    </struc>
  </struc>
</struc>
```

Through the start-tag/end-tag bracketting, XML elements are always explicitly labeled by their type. Therefore, in Prolog, one does not intuitively see from the symbols channel-tunnel and service-tunnel that the first is individual constant and the second is constructor. In XML , as in strongly typed LP languages ,such distinctions are made explicit - however at every occurence of a symbol. Example above gives impression of self-description advantage - but ‘space requirement’ - of this generous application of 'syntactic sugar'. Instead of writing reduntantly every closing XML bracket as a full 'end-tag', the 'neutral' closing bracket </> could be used - approaching Prolog structures - as in XML-QL [http://www.w3.org/TR/NOTE-xml-ql/]. This, however, is not standardized in XML 1.0. Lists - as they are used in Prolog - for example, can also be reduced to nested cons-structures or be represented as n-tuples [Bol99] in XML directly. In XML, a *logic-variable* is marked through <var>*-element* of the form <var> ... </var>. Thus the Prolog variable Xyz becomes the XML element <var> xyz </var>, whereby, through the </var>-markup conventions like the writing of capital letters for beginning words, are rendered redundant.

Hence a (non-variable free) non-ground structure like Prologs undersea-connection(britain,Xyz) becomes XMLs non-ground element

```
<struc>
  <constructor> undersea-connection </constructor>
  <ind> britain </ind>
  <var> xyz </var>
</struc>
```

4 HORN CLAUSES as XML Elements

For the representation of Horn clauses (facts and rules), we introduce further elements of XML. A predicate or *relation symbol* will in XML be a <relator> *element*.
For example, the relation symbol travel, will be <relator> travel </relator>.

Relation symbol application to terms is labeled in XML with a <relationship> *element*.
A travel-application travel(john,channel-tunnel) on two individual constants will thus be

```
<relationship>
  <relator> travel </relator>
  <ind> john </ind>
  <ind> channel-tunnel </ind>
</relationship>
```

A Horn fact in XML can therefore be asserted as <hn>-*element* that possesses <relationship> element as subelement. In the example, the Prolog fact

```
travel(john,channel-tunnel).
```

becomes

```
<hn>
  <relationship>
    <relator> travel </relator>
    <ind> john </ind>
    <ind> channel-tunnel </ind>
</relationship>
</hn>
```

For the representation of rules, no further XML element types will now be required.

Relation call is written again as <relationship> *element*.
The carry call carry(eurostar,someone) with individual constant and logic variable becomes

```
<relationship>
  <relator> carry </relator>
  <ind> eurostar </ind>
  <var> someone </var>
</relationship>
```

Then, a Horn rule in XML is asserted as an <hn> *element*, that has - the head <relationship>-element followed by at least one body <relationship> element. The above example is consequently generalized to a Prolog rule

```
travel(someone,channel-tunnel) :- carry(eurostar,someone)
```

and rewritten in XML as

```
<hn>
  <relationship>
    <relator> travel </relator>
    <var> someone </var>
    <ind> channel-tunnel </ind>
  </relationship>
  <relationship>
    <relator> carry </relator>
    <ind> eurostar </ind>
    <var> someone </var>
  </relationship>


</hn>
```

Pure XmlLog - the pure Prolog in XML syntax up to now introduced via examples - will be precisely defined in paragraph 6.

5 Attributes for Extended Logic

Although nested elements -as tree - allow the representation of arbitrary information, in some situations they lead to unnecessarily deeply or broadly nested representations.In XML, remedies are often obtained through *attributes* start-tag which can be enhanced by n attribute value pairs of the form a_i=v_i. Therefore, every element is a sequence of the type <*tag* a_1=$v_1 \ldots a_n$=v_n> . . . </*tag*>.

Thus,for example the <sales> element at the end of section 2 written more compactly and intuitively through the conversion of the <channel> subelement into a channel attribute with the string value "online" :

```
<sales channel="online">
          <company> Onoffbook </company>
          <item> XML4You </item>
          <quantity> 12417 <quantity>
</sales>
```

While there exists no direct counterpart to this in Prolog, other Logic Programming languages allow a corresponding conversion of the original Prolog structure of section 2. In the functional-logical Relfun, [http://www.relfun.org/] one would obtain - apart from bracket type - the structure

```
sales(channel(online))(
          company(onoffbook),
          item(xml4you),
          quantity(12417)
                    )
```

with a constructor sales(channel(online)) parametrized by the attribute.

Additional information, however, is often best introduced as attributes instead of as subelements. The <noun> copies </noun> used in section 2, analogously, is extensible through attributes for number and gender to noun(genus(neutrum),numerus(plural))(copies) respectively to, and to <noun genus="neutrum" numerus="plural"> copies </noun>.

In general, grammars in XML and LP can be constructed with this feature.

Regardless of the grammar, XML attributes can be used in logic programming for the representation of feature-/Psi terms. Since the values in a_i=v_i are atomic in XML 1.0, only feature-/Psi-terms with corresponding atomic values are directly representable.This, however, is not a problem, for example, in descriptive logics [CDL99].

A helpful application of XML attributes for Prolog are in particular the "arity" labelings of relation symbols. Thus, in section 4, a relation symbol labeled as binary travel/2 could be written in XML with an arity attribute as <relator arity="2"> travel </relator>. The same goes for arity labelings of constructors and -in functional-logical languages- of (defined) function symbols.

Analogously, via attributes annotations are possible on arbitrary element levels- among others: mode declarations for logic variables, determinism specifications for clauses or procedures and context conditions for entire knowledge bases.

Finally, the annotations which are necessary for compilating logic source programs can be represented in a declarative intermediary language, e.g., in the form of ''classified clauses'' [Kra90]. These are now representable as XML attributed elements.

Aside from the previous used attributes with values of the general type CDATA (character data''), XML recognizes attributes of more specific type too. Thus there exists, in particular, the types ID and IDREF for naming and referencing of elements of a given XML document. A ID-typed value must uniquely identify an element and an IDREF-typed value must also occur as ID-value of an element.

With these XML means, many meta logical and modal logical circumstances can be expressed, since, for example, clauses can be named and their names can be used for the construction of further clauses. In this section, we first assume that, as in many XML tools, the attributes id and idref are of the type ID and IDREF, respectively. In such an extension of our pure XmlLog, we can, for example, name the Prolog fact of section 4 as follows, and moreover, save ''hypothetic Prolog facts'' in a partial knowledge base.

```
<hn id="john-channel">
 <relationship>
    <relator> travel </relator>
    <ind> john </ind>
    <ind> channel-tunnel </ind>
  </relationship>
</hn>
```

For the representation of the ''modal Prolog facts''

```
belief(mary,travel(john,channel-tunnel)).
```

we can access the declaration "john-channel" from out of another partial knowledge base, as follows:

```
<hn>
  <relationship>
    <relator> belief </relator>
    <ind> mary </ind>
    <prop idref="john-channel"/>
  </relationship>
</hn>
```

Here, the XML abbreviation of empty elements <*tag* ...> </*tag*> to <*tag* .../> is used in order to write the propositional argument of the belief operators as <prop idref="john-channel"/> in a compact manner. One should observe that the above Prolog-like notation for propositions as arguments, without the declaration of modal operators like belief or the separation of relation symbols and constructors are not distinguishable from the notation for structures as arguments. In the self-describing XML notation, the tags for prop and struc are clearly differentiated, totally regardless of the fact that we have represented propositions in XML only by their names.

Over ID/IDREF, the tree structure of XML can be generalized to acyclic constructed graphs (DAGs). Thus a disbelief fact for fred with idref can access the same "john-channel" declaration, i.e. ID/IDREF enable sharing. Beyond that, one also gains access to general directed graphs, with which, e.g., regular trees can be represented in Prolog II+ [Col84].

6 Document Type Definition for Logic Languages

In sections 3 and 4, using examples, we introduced Horn logic language as pure XmlLog. XML also permits their general definition.

In order to define languages syntactically, XML uses *Document Type Definitions (DTDs)*, which we will firstly consider only as ELEMENT *declarations* for **non-attributed elements**. On the level of the non terminals, a DTD corresponds to an ordinary, context-free grammar in a varied ???? (EBNF)-notation. On the level of terminals, however, usually, any permutations of the alphabet taken are used (so called. Parsed Character Data, or ‘‘PCDATA’’), instead of specifying definite terminal sequences. The abbreviation #PCDATA used in DTDs can also be considered as a nonterminal, which (like Kleenian conclusion) expands to all ‘Tokens’ that can be built with the alphabet.

With a DTD grammar, therefore, a context-free word pattern with any ‘Tokens’ can be derived from the sheets. This however is sensible only when one thereby constructs the derivation-tree in a form linearized by brackets as the generation result. In XML this takes place by letting the well known brackets <*tag*> . . . </*tag*> remain for every nonterminal *tag*.

A given XML element is characterized as *valid* wrt a DTD, if it can be generated from the DTD as a linearized derivational tree in this way.

As an example of a DTD we consider the syntactic ELEMENT declarations for our
pure XmlLog as a knowledge base (kb) of zero or more Horn clauses (hn*):

```
<!ELEMENT kb             (hn*) >
<!ELEMENT hn             (relationship, relationship*) >
<!ELEMENT relationship   (relator, (ind | var | struc)*) >
<!ELEMENT struc          (constructor, (ind | var | struc)*) >
<!ELEMENT relator        (#PCDATA) >
<!ELEMENT constructor    (#PCDATA) >
<!ELEMENT ind            (#PCDATA) >
```

```
<!ELEMENT var          (#PCDATA) >
```

With this DTD we show a generation of the XML-linearized derivation tree for the kb-embedded example rule in section 4 which, like an XML validator, one can read from bottom to top as a parsing. One should observe how, for example, the start symbol kb encloses the clauses which it introduces later, as part of the linearized start-tag/end-tag tree representation <kb> . . . </kb>.The middle steps which had been left out are easily completed:

```
kb

<kb> hn* </kb>

<kb> hn </kb>

<kb> <hn> relationship relationship* </hn> </kb>

<kb> <hn> relationship relationship </hn> </kb>

.    .    .

<kb>
  <hn>
    <relationship>
      <relator> #PCDATA </relator>
      <var> #PCDATA </var>
      <ind> #PCDATA </ind>
    </relationship>
    <relationship>
     <relator> #PCDATA </relator>
     <ind> #PCDATA </ind>
     <var> #PCDATA </var>
    </relationship>
  </hn>
</kb>

<kb>
  <hn>
    <relationship>
      <relator> travel </relator>
      <var> someone </var>
      <ind> channel-tunnel </ind>
    </relationship>
    <relationship>
      <relator> carry </relator>
      <ind> eurostar </ind>
      <var> someone </var>
    </relationship>
```

```
  </hn>
</kb>
```

We now consider DTDs for **attributed elements**. For that we use ATTLIST *Declarations*, which gives to an element name an attribute name with its attibute type and possible occurences.

In a first example ,we declare the relator attribute arity used in section 5 as CDATA-typified (analogous to #PCDATA) and optionally appearing as (#IMPLIED):

```
<!ATTLIST relator       arity CDATA #IMPLIED >
```

As a second example we define the extended XmlLog introduced in section 5 with named hn-clauses and embedded propositions.To accomplish this we first change the above ELEMENT-declaration of relationship to prop and declare the latter as an empty element :

```
<!ELEMENT relationship  (relator, (ind | var | struc | prop)*) >
<!ELEMENT prop          EMPTY >
```

Then we add an ATTLIST-declaration, which specifies the attribute id as optional ID type for the idref as mandatory IDREF type: for the

```
<!ATTLIST hn            id    ID    #IMPLIED >
<!ATTLIST prop          idref IDREF #REQUIRED >
```

The "john-channel"-known fact from 5 and the fact accessing it can now be generated/parsed with the whole DTD.

7 XML Query Languages and Logic Programming

The W3C standardization of XML query languages is not yet as far advanced as the W3C standardization of XML itself. Nevertheless, a provisional comparison with logic programming is already possible. This comparison will begin examplarily on the basis of XQL [http://www.w3.org/TandS/QL/QL98/pp/xql.html] and XmlLog. We will consider the relationships in both directions.

For an **LP-treatment of XQL-queries** we will take as the basis, a document with the <s>-rooted element-nesting from section 2. This document is extended to attributes for gender and number corresponding to section 5. We assume that the document is accessible via a variable called Root.

Through the XQL-query //*tag*, all elements of the form <*tag* ...> . . . </*tag*> of the document are returned. In the following example, //np delivers the following three elements, wherein the third is contained in the second:

```
<np>
  <noun genus="neutrum" numerus="singular"> Onoffbook </noun>
</np>

<np>
```

```
  <ngroup>
    <card> 12417 </card>
    <noun genus="neutrum" numerus="plural"> copies </noun>
  </ngroup>
  <pp>
    <prep> of </prep>
    <np>
      <noun genus="neutrum" numerus="singular"> XML4You </noun>
    </np>
  </pp>
</np>

<np>
  <noun genus="neutrum" numerus="singular"> XML4You </noun>
</np>
```

In pure logic programming, this behavior would be realized through a relation-call descendant(Root,np,Res), which binds a result-variable nondeterministically. Closer to the value-oriented thinking of XQL, and somewhat as in a functional-logical language, would be the use of a function-call descendant(Root,np). This function-call returns the following Prolog-structures as non-deterministically enumerated objects:

```
np(noun(genus(neutrum),numerus(singular))(onoffbook))

np(ngroup(card(12417),noun(genus(neutrum),numerus(plural))(copies)),
   pp(prep(of),np(noun(genus(neutrum),numerus(singular))(xml4you))))

np(noun(genus(neutrum),numerus(singular))(xml4you))
```

Through the XQL-query
//*tag*[@*a*=*v*] all elements of the form <*tag* ...*a*=*v*...> . . . </*tag*> of the document are returned which contain the attribute value pair *a*=*v* .

In the example //noun[@numerus="plural"] delivers only the following element:

```
<noun genus="neutrum" numerus="plural"> copies </noun>
```

The corresponding function call descendant filter(numerus(plural))(Root,noun) parametrized through a filter, returns the following Prolog-structure:

```
noun(genus(neutrum),numerus(plural))(copies)
```

The definition of query functions like descendant and descendant-filter( ) can, naturally, not consist only of (deterministic!) Prolog unifications. All Prolog substructures with the given constructors must be formed through recursive transversing, and filters must be applied to the constructor parameter.

Conversly, the unification permits logical variables not only in the query, but also in the document, somewhat like in the element for the non-ground structure undersea-connection(britain,Xyz) from section 3.

Thus, truly unifying queries that correspond to undersea-connection(Same,Same),

and which connect Same to britain, could also be made on the document.

The non-ground element, saved and universally readable, documents, among others, the claim that a British island is connected to another island undersea. The non-ground documents are like forms with presently identical fields that are to be filled out. Experience will reveal the extent to which these non-ground documents will be used in practise.

We will now turn to **queries and inferences on the basis of the pure XmlLog** defined in sections 4 and 6.

Let us consider the fact carry(eurostar,fred). It can be used in its XmlLog version

```
<hn>
  <relationship>
    <relator> carry </relator>
    <ind> eurostar </ind>
    <ind> fred </ind>
  </relationship>
</hn>
```

In order to answer the XmlLog-version of the query carry(eurostar,someone) shown in section 4, the XML logic variable connects

<var> someone </var> to <ind> fred </ind>

With such carry basis facts, a rule is now applicable for dynamically deriving travel declarations whenever neccesary. This obviates having to globally and statically save them: travel(Someone,channel-tunnel): carry(eurostar,Someone). Their XML version from section 4 is usable for inferential queries such as travel(fred,Where) in XML versions like:

```
<relationship>
  <relator> travel </relator>
  <ind> fred </ind>
  <var> where </var>
</relationship>
```

This query unifies

<var> someone </var>= <ind> fred </ind>

and

<var> where </var> = <ind>channel-tunnel </ind>

with the head of the rule.

The internal XML query corresponding to carry(eurostar,fred), and resulting out of the first connection from the body of the rule, is succesful with the above XML-fact. Hence, the second connection, i.e. Where = channel-tunnel is furnished as a result of the external XML query.

In all, therefore, the inference is carried out as an SLD-resolution step [Llo87].

In general, the procedural semantics of *SLD Resolution* can be carried over to XmlLog. If, for this purpose, we assume that the clauses are distributed over various documents in the Web, then we find ourselves in the situation described in [Bol97]: In this open world, in particular, hardly any logical completeness can be claimed. The (Java-) realization of the semantics in the form of an XmlLog-interpreter could occur on the client side as a standard extension or Plug-in of the Web browser.

8 Related Works and Conclusions

The pure XmlLog introduced here is a sub-language of **RFML** (Relational-Functional Markup Language) [Bol99], which admits, among others, user-defined functions and higher-order operators [http://www.relfun.org/rfml/].

SHOE (Simple HTML Ontology Extensions) [HHL99] is the first language providing Horn clauses in the Web to have become well-known. Before XML was clearly defined, SHOE was specified first in HTML, and only later on in XML [http://www.cs.umd.edu/projects/plus/SHOE/].

An interesting aspect of SHOE is that individual constants are represented through the (official) URL/URI for the individual. In our Horn rule example in section 4, two individuals appear, which in this way, can be representative of eurostar = http://www.eurostar.com/ and channel-tunnel = http://www.eurotunnel.com/. Hence this rule in SHOE becomes

```
<DEF-INFERENCE DESCRIPTION="travel(?someone,http://www.eurotunnel.com/) if
                           carry(http://www.eurostar.com/,?someone)">
        <INF-IF>
                <RELATION NAME="carry">
                        <ARG POS=1 VALUE="http://www.eurostar.com/">
                        <ARG POS=2 VALUE="someone" USAGE=VAR>
                </RELATION>
        </INF-IF>
        <INF-THEN>
                <RELATION NAME="travel">
                        <ARG POS=1 VALUE="someone" USAGE=VAR>
                        <ARG POS=2 VALUE="http://www.eurotunnel.com/">
                </RELATION>
        </INF-THEN>
</DEF-INFERENCE>
```

"Courteous Logic Programs" [GLC99] for prioritized treatment of agent-communication conflicts in E-Commerce (e.g. in supply-orders) were studied. For the exchange of knowledge, an XML-based Business Rules Markup Language (**BRML**) is developed, [http://www.oasis-open.org/cover/brml.html], which also codes the clause subset of the Knowledge Interchange Format (**KIF**) [http://logic.stanford.edu/kif/dpans.html] in XML.

Of the many suggested XML-query languages, [http://www.w3.org/TandS/QL/QL98/] we have until now mentioned only **XQL** [http://www.w3.org/TandS/QL/QL98/pp/xql.html] and **XML-QL** [http://www.w3.org/TR/NOTE-xml-ql/].

A more user-oriented alternative is **EquiX**. In **EquiX**, Result-documents with their DTDs are automatically constructed from queries over a formula-based Interface. [CYK$^+$99].

The **Algebra for XML-query languages** [http://www.cs.bell-labs.com/who/wadler/papers/xquery-algebra/xquery-algebra.html] is implemented in the functional Haskell. Used with the Haskell generalizing functional-logical Curry, [Han97] it is well transferred to Logic-programming.

More closely related to term replacement systems than to logic programming is XSL (Extensible Stylesheet Language): For limited term-replacements, the XSL-part **XSLT** (XSL Transformations) [http://www.w3.org/TR/xslt] is preferred to XML documents. For this, XSLT again uses an XQL-like muster language. An exact characterization of the XSLT term replacement system subclass would be interesting, particularly after our representation of XML elements as 'ground terms' in this work. The corresponding processing of 'non-ground' terms with 'narrowing' will be examined in functional-logical languages like Curry and Relfun.

Because of place restrictions, the **XML-name spaces** cannot be discussed here, although there exist interesting relationships with the Module-systems of LP-languages. The latter were, as is generally accepted, the more difficult part in the ISO-Standardization of Prolog [http://www.logic-programming.org/prolog_std.html]. On the other hand, they are not totally specific for Logic programming.

Nor can we go more deeply into the following XML extensions which are built up on that: (1) **RDF** for the description of metadata, and (2) **RDF Schema** for the corresponding Schema/Ontologie-specification. For RDF queries and RDF inferences, however, technologies from the field of logic programming come into question; technologies such as the application of F-Logic [DBSA97]. As an RDF-editor, a variation of Protégé-2000 would be suitable. [http://smi-web.stanford.edu/projects/protege/protege-rdf/protege-rdf.html].

In this contex it should be remarked, that XML 1.0 does not offer the usual built-in types like integer, float, etc.. Therefore, in section 6 for example, we had to typify the attribute arity with unspecific CDATA strings, although only the natural numbers come into question there. Here, **XML Schema** comes to the rescue with the differentiated type system in part 2 [http://www.w3.org/TR/xmlschema-2/]. Further examinations could, therefore, be dedicated to the relationships of type-/sorts- (the signature systems of (feature-/Psi-term-)LP-languages, respectively), to those of XML Schema and RDF schema.

Finally, XML opens new areas of analysis and application for logic-programming. This is in extension of Internet-Workshops into LP-conferences since 1996. Both areas can profit from the interplays discussed. Many of the XML-Extensions as preliminary drafts are not yet ready for (W3C)standardization. Therefore, even now, one can still influence the final form of these XML-Extensions.

References

[Bol97] Harold Boley. Wissensbasen im World Wide Web: Eine Herausforderung für die Logische Programmierung. In François Bry, Burkhard Freitag, and Dietmar Seipel, editors, *Twelfth Workshop on Logic Programming (WLP'97)*. Institut für Informatik, LMU München, September 1997.

[Bol99] Harold Boley. The Relational-Functional Markup Language RFML. Technical report, Stanford Medical Informatics, December 1999.

[CDL99] D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and Reasoning on XML Documents: A Description Logic Approach. *JLC: Journal of Logic and Computation*, 9, 1999.

[Col84] A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *Proc. of the International Conference on Fifth Generation Computer Systems (FGCS-84)*, pages 85–99, Tokyo, Japan, November 1984. ICOT.

[CYK+99] S. Cohen, Y.Kanza, Y. Kogan, W. Nutt, A. Serebrenik, and Y. Sagiv. EquiX – Easy Querying in XML Databases. In *Proc. of the ACM SIGMOD Workshop on the Web and Databases (WebDB'99)*, Philadelphia, Penn., June 1999.

[DBSA97] Stefan Decker, Dan Brickley, Janne Saarela, and Jürgen Angele. A Query and Inference Service for RDF. In *QL'98 - The Query Languages Workshop, http://www.w3.org/TandS/QL/QL98/*. World Wide Web Consortium, 1997.

[DEDC96] P. Deransart, A. Ed-Dbali, and L. Cervoni. *Prolog: The Standard*. Springer Verlag, 1996.

[GLC99] Benjamin N. Grosof, Yannis Labrou, and Hoi Y. Chan. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In *Proc. 1st ACM Conference on Electronic Commerce (EC-99), Denver, Colorado*, 1999.

[Han97] Michael Hanus. A Unified Computation Model for Functional and Logic Programming. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 80–93, Paris, France, January 1997.

[Har99] Elliotte Rusty Harold. *XML Bible*. IDG Books Worldwide Inc., San Mateo, CA, USA, 1999.

[HHL99] Jeff Heflin, James Hendler, and Sean Luke. SHOE: A Knowledge Representation Language for Internet Applications. Technical Report CS-TR-4078, University of Maryland, College Park, October 1999.

[Kra90] Thomas Krause. Klassifizierte relational/funktionale Klauseln: Eine deklarative Zwischensprache zur Generierung von Register-optimierten WAM-Instruktionen. Technical Report SWP–90–04, University of Kaiserslautern, Department of Computer Science, May 1990.

[Llo87] John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, Heidelberg, New York, 1987.