The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations*

Harold Boley

DFKI GmbH
boley@dfki.de

Abstract. Shared declarative aspects of Prolog and XML are examined. An XML version of pure Prolog is shown to be at the center of the Rule Markup Language. The RuleML data model uses Order-Labeled trees, combining the RDF and XML models. As part of RuleML's hierarchy of sublanguages, the RuleML-Prolog DTD is developed into an XML Schema. XSLT (XSL Transformations) is employed for practical XML-to-XML and XML-to-(X)HTML transformation of Prolog on the Web.
.

1 Introduction

The original Web enabled the distributed development, usage and maintenance of HTML-based informal documents. The Semantic Web [BLHL01] now attempts to employ the same infrastructure for formal specifications or declarative programs. Besides description logic, Horn logic and Prolog have the potential to serve as a uniform semantic and pragmatic (implementational) foundation in this endeavor.

The potential is due to Web-based B2C and B2B business rules becoming an important application area of the Semantic Web. This can be illustrated by a merchant-customer exchange scenario. Suppose a customer (who can represent a business) has preselected some merchants who can all offer a desired product. The customer now wants to compare their discounts etc., which may be conditional on the customer as well as the product, hence are best formulated as (Horn) rules. While the merchants may use different internal formats for such business rules, they need to translate them into a standard format to be understandable to the customer (or to his agent). For example, some merchant may use the following business rule:

* I would like to thank Oskar Bartenstein, Osamu Yoshie, Ulrich Geske, and the program committee of INAP2001 for inviting me to give this presentation. Moreover, thanks go to this volume's referees for valuable suggestions. I also want to express my gratitude to Said Tabet, Benjamin Grosof, Gerd Wagner, and all other colleagues from the Rule Markup Initiative for joining their forces on this Web standards effort. This research was funded by the EU in the IST project Clockwork.

The discount for a customer buying a product is 5.0 percent if the customer is premium and the product is regular.

Internally, this merchant may express it in the Datalog subset of Prolog, thus:

```
discount(Customer,Product,"5.0 percent") :-
                                premium(Customer), regular(Product).
```

Another merchant may express similar discounting conditions as an SQL view. Yet another one could use some proprietary representation.

The syntactic exchange format, however, will most likely be a version of the Extensible Markup Language (XML) standard. Fortunately, translators between the XML and, say, Prolog syntaxes can be defined, which can also form the basis for semantic-pragmatic technology transfers [Bol02]. If two rule languages are already expressed in XML, translators between them can even be based on the standard Extensible Stylesheet Language Transformations (XSLT) technology. In the current paper these possibilities will be exemplified using the XML syntax of the Rule Markup Language (RuleML), Version 0.8 [BTW01], in the following just called 'RuleML'. The RuleML form of the above business rule is depicted in section 7. Similar RuleML rules are processed by the rule-applying comparison-shopping agent RACSA [http://www.dfki.de/racsa/].

The Rule Markup Initiative [http://www.dfki.de/ruleml/] constitutes an open network of various groups from industry and academia. Our first goal has thus been to provide a modular system of RuleML sublanguages based on XML and RDF. This led to the development of a novel XML-RDF-integrating Web data model. On its basis, RuleML was initially defined with a hierarchy of DTDs, which has been partially redefined with XML Schema. Java-based user interfaces and engines for this RuleML definition have been developed. To facilitate rule exchange, XSLT-based translators between RuleML and other rule languages have been specified. For details see the initiative's website above.

After this introduction, XML elements and attributes will be shown to correspond to Prolog ground structures (section 2). Next, the RuleML data model will be presented and illustrated via Order-Labeled trees (section 3). This then permits to generically represent Herbrand terms as RuleML XML elements (section 4). Horn clauses will be similarly represented as XML elements (section 5). Prolog will subsequently be defined within the DTD/Schema hierarchy of RuleML sublanguages (section 6). XSLT translators will be exemplified by a bidirectional RuleML-RFML transformer and an associated HTML renderer (section 7), followed by some conclusions.

2 XML Elements and Attributes as Ground Structures

From a Prolog perspective the expressive power of XML looks quite limited, as it is on the level of ground structures, needing no variables. This section will show the direct correspondence between XML elements and Prolog ground structures, and a somewhat indirect correspondence between XML attributes and Prolog ground structures.

XML documents consist of (nested) elements. Each *element* is a sequence of the form <*tag*> c_1 . . . c_m </*tag*>, i.e. some ordered *content* "c_1 . . . c_m" enclosed by *tag*-'colored' start- and end-brackets. This content may consist of text or again of elements. Prolog terms can represent XML elements as variablefree or *ground* structures of the form *tag*(c_1, . . ., c_m).

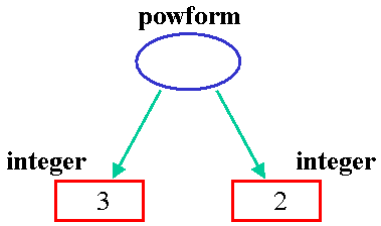
For example, suppose we want to mark up power formulas, in a typed manner, as integer bases raised to integer exponents. In XML we can use a positional representation, where a `power formula` consists of a base followed by an exponent; the base and exponent can each be marked up as an `integer`. Thus, the power formula 3^2 would be marked up as the following nested element:

```
<powform>
  <integer>3</integer>
  <integer>2</integer>
</powform>
```

In Prolog this corresponds to the following, more compact, ground structure (integers and lower-case words will directly be used as constants):

```
powform(integer(3),integer(2))
```

Graph-theoretically, we obtain a tree with left-to-right-ordered arcs and two kinds of nodes labeled by their types, here `powform` and `integer`, where oval nodes are XML-like (sub)elements or RDF-like resources while rectangular nodes are XML-like PCDATA or RDF-like literals, here 3 and 2:



In general, these simple XML elements and Prolog terms constitute two equivalent ways of linearizing left-to-right-ordered, node-labeled trees (with oval inner nodes and rectangular leaf nodes [http://www.dfki.de/~boley/xslt/xmlrdf/powformdivrootposi.gif]).

Besides elements, XML uses *attributes*: A start-tag can be enhanced by n non-positional attribute-value pairs of the form a_i=v_i, where v_i must be a string (XML's CDATA); in general, every element thus has the form <*tag* a_1=v_1...a_n=v_n> c_1 . . . c_m </*tag*>. However, this can be re-represented as the attributeless form <*tag*> v_1...v_n c_1 . . . c_m </*tag*>, where the n attribute values become positionalized as the first n subelements. This again corresponds to a Prolog ground structure. We can thus treat XML elements, with attributes included, as Prolog-like terms.

A data model that better – non-positionally – captures XML attributes, as a special case, will be developed in the following section.

3 The RuleML Data Model

Since XML-based serializations are quite verbose, a good underlying graph-theoretical data model will be essential – for semantic formalizations, translator and engine implementations, as well as user interfaces. This section will develop the RuleML data model by unifying existing Web data models.

The XML and RDF communities have developed W3C recommendations with different data models. XML is based on, possibly attributed, left-to-right ordered, node-labeled trees, reminiscent of parse (syntax) trees, except that their hierarchical structure permits an overlay of 'horizontal' `id/idref` links. RDF [LS99] is based on directed, arc-labeled (unordered) graphs with two kinds of nodes, resources and literals, where the latter do not allow outgoing arcs. While originally the intended uses of XML and RDF seemed to be sufficiently distinct to justify two different data models, it later turned out, e.g. with the advent of the Semantic Web [http://www.w3.org/2001/sw/], that a unified data model would be advantageous.

Consider the following problem with positional XML markup. While in section 2 `powform`'s binary-positional convention "First subelement is base, second subelement is exponent" seemed natural and easy to memorize, without some kind of 'signature declaration' this markup could instead mean 2^3 according to a "First subelement is exponent, second subelement is base" convention. Analogous conventions for N-ary operators ($N > 2$) need to disambiguate a combinatorially exploding number of possible interpretations. Without extra information the positions of the 'roles' (`powform`: base and exponent) of two or more subelements cannot in general be uniquely determined from some such markup.

In 'object-centered' modeling and 'feature'/'frame' logics the way out is representing powers and other operators in a non-positional manner, making them *objects* with explicitly indicated *roles* for their arguments. In the Frame/Horn-logic-integrating DOOD system F-Logic [KL89], developed for RDF in TRIPLE [SD01], our example could thus be represented as a fact:

```
powform[base->3:integer; exponent->2:integer].
```

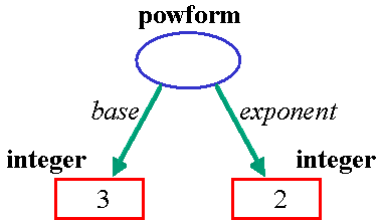
Paralleling such an approach in RuleML, we complement XML 'type tags' by explicitly distinguished, RDF-like 'role tags' [Bol01]. Generalizing XML attributes, roles will allow values that themselves contain markup. In our example we thus use `powform` with `_base` and `_exponent` subelements as follows, where a leading "`_`" distinguishes roles:

```
<powform>
  <_base><integer>3</integer></_base>
  <_exponent><integer>2</integer></_exponent>
</powform>
```

This is algebraically equivalent to the following commutative markup using the other correctly prefixed `powform`-subelement permutation:

```
<powform>
  <_exponent><integer>2</integer></_exponent>
  <_base><integer>3</integer></_base>
</powform>
```

Graph-theoretically, the RuleML data model permits trees with unordered arcs labeled by their roles, here base and exponent, and with the nodes introduced earlier:



Generally, for trees the branching order of (explicitly labeled) arcs is immaterial and for markups the following equation holds:

<*type*>. . .<*_role*$_1$>...</*_role*$_1$>. . .<*_role*$_2$>...</*_role*$_2$>. . .</*type*>

=

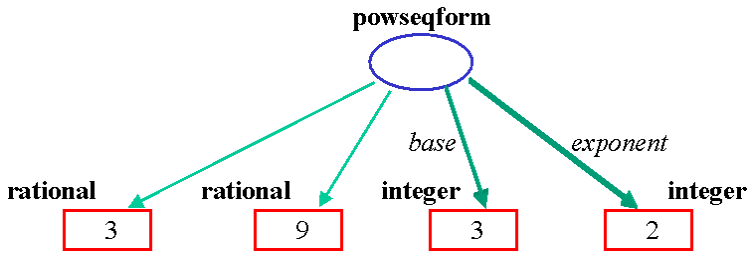
<*type*>. . .<*_role*$_2$>...</*_role*$_2$>. . .<*_role*$_1$>...</*_role*$_1$>. . .</*type*>

RDF-like role-prefixed and XML's positional children can be easily combined, obtaining our basic RDF-XML integration of *Order-Labeled (OrdLab) Trees*.

For example, suppose we want to combine the markup of a `power` `sequence` $a^1, \ldots, a^n$ with the marked up `formula` showing its base a and highest exponent (length) n. For the sequence $3, 9$ combined with the formula 3^2 we obtain the following markup (like RDF, we allow an object, e.g. `3`, to have multiple types, e.g. `rational` and `integer`):

```
<powseqform>
  <rational>3</rational>
  <rational>9</rational>
  <_base><integer>3</integer></_base>
  <_exponent><integer>2</integer></_exponent>
</powseqform>
```

Graph-theoretically, positional children become left-to-right-ordered arcs, here targeting the `rationals` `3` and `9`, while role children become labels on unordered arcs, here targeting the `integers` `3` and `2` (emphasized by different kinds of arrows for ordered and labeled arcs):



By the nature of unordered children/arcs, in the example they could also be internally permuted and/or put before the sequence of ordered children/arcs.

In a generalized F-Logic this could be represented as the fact

```
powseqform[3:rational, 9:rational;
           base->3:integer; exponent->2:integer].
```

where the 'ordered infix' "`,`" has precedence over the 'unordered infix' "`;`" and the ordered sequence is connected with "`->`" pairs via "`;`".

A *startrole (endrole) normal form* can be defined for OrdLab trees, their markups, and generalized F-Logic, where all role children are put before (after) the positional children and are ordered lexicographically according to their role names; for the empty sequence of positional children the startrole and endrole normal forms coincide. Thus, the previous OrdLab, markup, and F-Logic versions of the "3^2" example are both in startrole and endrole normal form; the above versions of the "$3, 9; 3^2$" example are in endrole normal form.

4 General Herbrand Terms as XML Elements

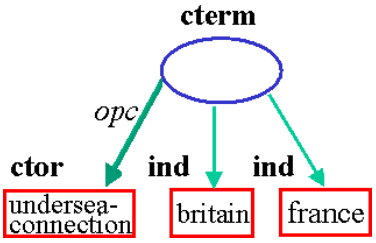
The RuleML data model can be used to represent general Herbrand terms by combining – in prefix or postfix form – one role child for the constructor with n positional children for the arguments. While section 2 re-represented XML elements as Prolog ground structures, the current section thus generically encodes Prolog's Herbrand terms as XML elements, using the RuleML data model of section 3.

The representation of Herbrand terms (individual constants, logic variables, and structures) in RuleML XML requires, in essence, the use of `ind`, `var`, and `cterm` elements, where the occurrence of a structure or complex `term` distinguishes Prolog from Datalog.

Proceeding by example here, the Prolog *ground structure* `undersea-connection(britain,france)` in RuleML becomes a `cterm` *element* with a role (unordered) subelement `_opc` for the `functor`/`constructor` of type `ctor` and two sequential (ordered) subelements for its argument terms, here two `ind`ividual constants of type `ind`:

```
<cterm>
  <_opc><ctor>undersea-connection</ctor></_opc>
  <ind>britain</ind>
  <ind>france</ind>
</cterm>
```

This RuleML `cterm` markup can be visualized as an OrdLab tree as introduced in section 3:



The single unordered arc is no problem, of course; it could have equivalently be drawn on the right-hand side of the two ordered arcs.

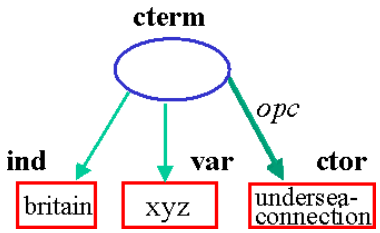
Lists – as they are used, e.g, in Prolog – could be reduced to nested (`cons`-)structures, but are directly represented as `n-tuples` in RuleML [http://www.dfki.de/ruleml/dtd/0.8/ruleml-hornlog.dtd].

In RuleML, a *logic variable* is marked through a `var` *element* of the form `<var>` ... `</var>`. Thus, the Prolog variable `Xyz` becomes the XML element `<var>xyz</var>`, whereby the `var` markup makes conventions like first-letter capitalizing superfluous.

Hence, a (not variablefree, i.e.) *non-ground structure* such as Prolog's `undersea-connection(britain,Xyz)` can be written in RuleML as the following element (we now put the constructor-role subelement after the two argument-sequence subelements):

```
<cterm>
  <ind>britain</ind>
  <var>xyz</var>
  <_opc><ctor>undersea-connection</ctor></_opc>
</cterm>
```

Visualizing this as an OrdLab tree gives us (with the unordered arc now drawn on the right-hand side):



Note that the ground OrdLab and markup example is in startrole normal form; the non-ground example is in endrole normal form.

The unification of Prolog terms (in RuleML) can be rendered as the unification of OrdLab trees, their markups, or generalized F-Logic facts, which can retain 'linearity' via a possible 1-step swap into either startrole or endrole normal form. For the above sample pair of ground and non-ground structures this binds `<var>xyz</var>` to `<ind>france</ind>`.

5 Horn Clauses as XML Elements

The RuleML data model can also be used to represent Horn clauses by combining – in either order – a role child for the clause head with an optional role child for the clause body. This section will develop corresponding XML elements for Horn facts and rules with a Datalog example.

The representation of Horn clauses (facts and rules) in XML calls for further elements. A predicate or *relation symbol* will in RuleML be a `rel` *element*. The *application* of an `_opr`-role-embedded relation symbol to a sequence of terms is marked by an `atom` *element* in RuleML. For example, a `travel` application `travel(john,channel-tunnel)` to two individual constants will thus be

```
<atom>
  <_opr><rel>travel</rel></_opr>
  <ind>john</ind>
  <ind>channel-tunnel</ind>
</atom>
```

Moreover, a Horn fact in RuleML is asserted as a `fact` *element* that possesses exactly one subelement – the `_head`-role-embedded `atom` element. In the example, the Prolog fact

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

becomes

```
<fact>
  <_head>
    <atom>
      <_opr><rel>travel</rel></_opr>
      <ind>john</ind>
      <ind>channel-tunnel</ind>
    </atom>
  </_head>
</fact>
```

Finally, a RuleML `imp` *element* is required to represent Horn rules. A *relation call* is again written as an `atom` *element*. The `carry` call `carry(eurostar,Someone)`, with an individual constant and a logic variable, becomes

```
<atom>
  <_opr><rel>carry</rel></_opr>
  <ind>eurostar</ind>
  <var>someone</var>
</atom>
```

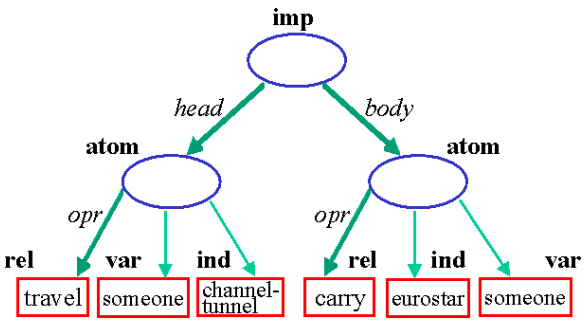
A Horn rule, then, in RuleML is asserted as an `imp` *element* that has two subelements – a `_head`-role `atom` element augmented (either to its right or left) by a `_body`-role `atom` or `and` element. The above fact example is thus generalized to the Datalog rule

```
travel(Someone,channel-tunnel) :- carry(eurostar,Someone).
```

and rewritten in RuleML as

```
<imp>
  <_head>
    <atom>
      <_opr><rel>travel</rel></_opr>
      <var>someone</var>
      <ind>channel-tunnel</ind>
    </atom>
  </_head>
  <_body>
    <atom>
      <_opr><rel>carry</rel></_opr>
      <ind>eurostar</ind>
      <var>someone</var>
    </atom>
  </_body>
</imp>
```

Graph-theoretically, RuleML clauses are again OrdLab trees, as illustrated for this rule:



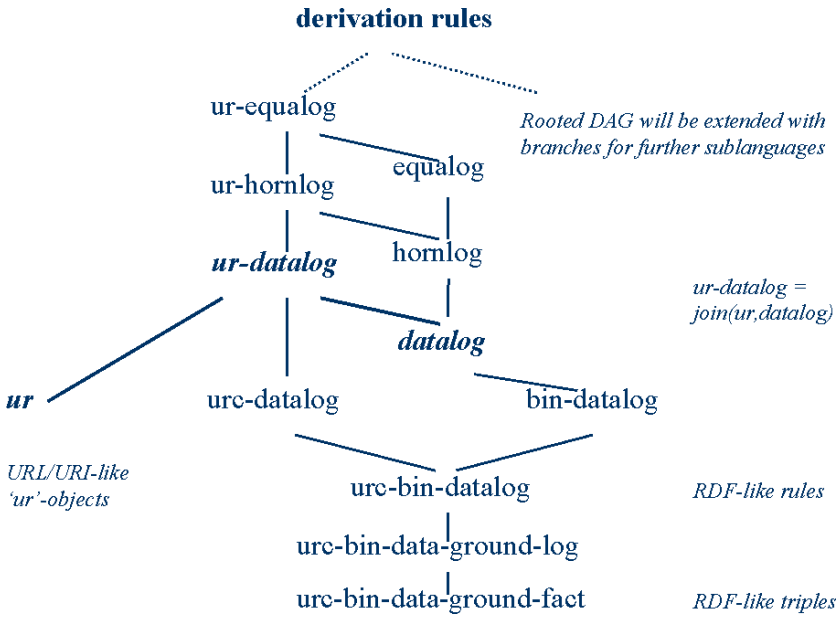
We could proceed to a full Prolog rule – both in the RuleML markup and in the OrdLab tree – by replacing the individual `channel-tunnel` with the ground `undersea-connection` `cterm` from section 4.

A Horn logic subset of RuleML – the pure Prolog in our XML syntax that was up to now introduced by examples – will be precisely defined in section 6.

6 Prolog in the DTD/Schema Hierarchy of RuleML

The XML DTD/Schema definition of RuleML can be viewed as syntactically characterizing certain semantic expressiveness subclasses of the language. This section will focus on the subclasses of Datalog and – on top of it – Prolog (actually, Horn logic).

RuleML has been modularized through a hierarchy of rule sublanguages, encompassing derivation rules (Prolog-like clauses), transformation rules (function-defining equations), and reaction rules (Event-Condition-Action rules). The following sublanguage hierarchy focusses on the 12 sublanguages of derivation rules that together constitute the modularized basic RuleML definition. All sublanguages except the 'UR' (URI) group correspond to well-known rule systems where each sublanguage has an associated semantic (model- and proof-theoretic) characterization [BTW01] (more expressive sublanguages are closer to the root):



In sections 4 and 5 we introduced RuleML's 'hornlog' sublanguage based on examples. XML also permits the general definition of this language via *Document Type Definitions (DTDs)* or, more expressively, *XML Schemas* [Fal01].

Actually, each node in the above hierarchy, e.g. 'hornlog', corresponds to a DTD/Schema that defines the syntax of this sublanguage: instance documents

– knowledge bases – can then refer to the most specific DTD/Schema to maximize interoperability. Non-leaf nodes are composed of the – possibly modified – node(s) reachable via their outgoing link(s) below plus possibly some extra definition parts. For example, 'hornlog' is composed of a – slightly modified – 'datalog' plus complex terms; 'datalog' itself contains 'bin-datalog', etc.

Let us first exemplify the DTD-to-Schema transition via the two initial DTD lines of 'datalog' and their – quite a bit longer – Schema transcriptions. A `rulebase` root element with `imp` rules and `fact` assertions as subelements, or the (EBNF-like) DTD line

```
<!ELEMENT rulebase ((imp | fact)*)>
```

along with `imp` rules consisting of – in any order – a conclusion role `_head` and a premise role `_body`, or the DTD line

```
<!ELEMENT imp ((_head, _body) | (_body, _head))>
```

become the following Schema, with the DTD's `_head`/`_body`-sequence permutation becoming the Schema's `xsd:all` set:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="rulebase">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="imp" type="impType"/>
        <xsd:element name="fact" type="factType"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="impType">
    <xsd:all>
      <xsd:element name="_head" type="_headType"
                   minOccurs="1" maxOccurs="1"/>
      <xsd:element name="_body" type="_bodyType"
                   minOccurs="1" maxOccurs="1"/>
    </xsd:all>
  </xsd:complexType>

. . .

</xsd:schema>
```

The remaining essential lines of the 'datalog' DTD,

```
<!ELEMENT _head (atom)>
<!ELEMENT _body (atom | and)>
<!ELEMENT and (atom*)>
<!ELEMENT atom ((_opr, (ind | var)*) | ((ind | var)+, _opr))>
```

can likewise be quite straightforwardly transcribed to XML Schema [http://www.dfki.de/ruleml/inxsd0.8.html]. For example, the `atom` line in XML Schema becomes

```
<xsd:complexType name="atomType">
  <xsd:choice>
    <xsd:sequence>
      <xsd:element name="_opr" type="_oprType"/>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="ind" type="indType"/>
        <xsd:element name="var" type="varType"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:sequence>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="ind" type="indType"/>
        <xsd:element name="var" type="varType"/>
      </xsd:choice>
      <xsd:element name="_opr" type="_oprType"/>
    </xsd:sequence>
  </xsd:choice>
</xsd:complexType>
```

The 'hornlog' DTD can now be composed from the 'datalog' DTD using conditional DTD parts via `INCLUDE`/`IGNORE` switches. Essentially, we thus obtain a choice-extended `atom` line plus an extra `cterm` (cf. section 4) line, as still employed for RuleML [http://www.dfki.de/ruleml/dtd/0.8/ruleml-hornlog.dtd] (here without `tup` lists etc.):

```
<!ELEMENT atom  ((_opr, (ind|var|cterm)*) | ((ind|var|cterm)+, _opr))>
<!ELEMENT cterm ((_opc, (ind|var|cterm)*) | ((ind|var|cterm)+, _opc))>
```

But we are currently investigating in XML Schema the issue of modularization/composition mechanisms better than conditional DTD/Schema parts, possibly involving `xsd:redefine`. In the 'datalog' Schema, if `xsd:all` groups would permit complex content (unfortunately, they currently do not), then the above sequence-permuting `xsd:choice` group for `atom`s could be rewritten thus:

```
<xsd:complexType name="atomType">
  <xsd:all>
    <xsd:element name="_opr" type="_oprType"
                 minOccurs="1" maxOccurs="1"/>
    <xsd:sequence minOccurs="1" maxOccurs="1">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="ind" type="indType"/>
        <xsd:element name="var" type="varType"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:all>
</xsd:complexType>
```

We would then require that `xsd:redefine` permit (which it currently does not) `choice`-extending the embedded sequence to

```
<xsd:sequence minOccurs="1" maxOccurs="1">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="ind" type="indType"/>
    <xsd:element name="var" type="varType"/>
    <xsd:element name="cterm" type="ctermType"/>
  </xsd:choice>
</xsd:sequence>
```

for maximum inheritance in the 'hornlog' Schema.

7 XSL Transformations from and to RuleML

It is well-known that intertranslating between every pair of languages leads to growth of the number of translators quadratic in the number of languages, while selecting one as the canonical language keeps growth linear. Applying this to XML-based rule languages, this section proposes RuleML as the canonical one and presents the first pair of translators.

XSLT (XSL Transformations) [Cla99] has been employed for practical XML-to-XML and XML-to-(X)HTML transformation of RuleML Prolog on the Web.

The example discussed here consists of XSLT translators between the Horn-logic subsets of RuleML and RFML (Relational-Functional Markup Language) [http://www.relfun.org/rfml/]. We have specified a pair of 'inverse' translators ('stylesheets') exporting/importing markup between RuleML and RFML, where only the logic part is needed for Prolog [http://www.relfun.org/ruleml/rfml-ruleml.html]:

`ruleml2rfml.xsl`: Stylesheet translates a Hornlog RuleML rulebase to a corresponding RFML program: its RuleML input analysis can be transferred to other "RuleML $\longrightarrow$ *XML-rule-language*" translators.

`rfml2ruleml.xsl`: Stylesheet translates a Hornlog RFML program to a corresponding RuleML rulebase: its RuleML output generation can be transferred to other "*XML-rule-language* $\longrightarrow$ RuleML" translators.

The two XML-to-XML stylesheets are quite different, since RFML – like normal Prolog – is positional while RuleML is role-based.

As an example let us consider a transformational roundtrip (and an HTML digression) starting and ending at the sample RuleML Datalog business rulebase for discounting [http://www.dfki.de/ruleml/exa/0.8/discount.ruleml]. We focus on its first rule, discussed in the introduction (section 1). This is marked up in RuleML as follows (an individual constant can syntactically be an entire phrase like "5.0 percent"):

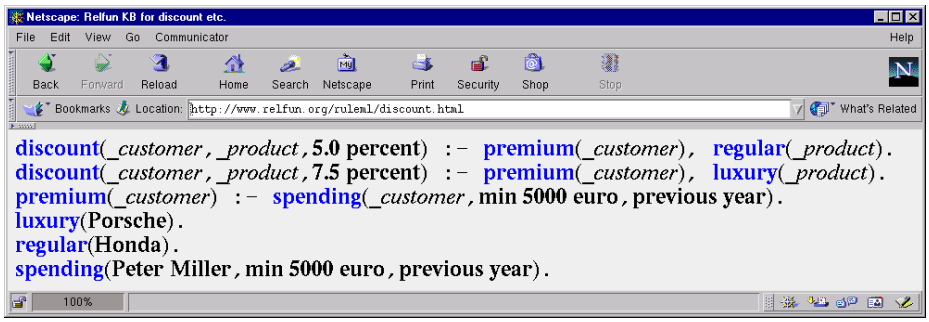
```
<rulebase>
  <imp>
    <_head>
      <atom>
        <_opr><rel>discount</rel></_opr>
        <var>customer</var>
        <var>product</var>
        <ind>5.0 percent</ind>
      </atom>
    </_head>
    <_body>
      <and>
        <atom>
          <_opr><rel>premium</rel></_opr>
          <var>customer</var>
        </atom>
        <atom>
          <_opr><rel>regular</rel></_opr>
          <var>product</var>
        </atom>
      </and>
    </_body>
  </imp>
. . .
</rulebase>
```

Via `ruleml2rfml`, this markup is transformed to the following RFML markup (except that additional whitespace has been inserted here):

```
<rfml>
  <hn>
    <pattop>
      <con>discount</con>
      <var>customer</var>
      <var>product</var>
      <con>5.0 percent</con>
    </pattop>
    <callop>
      <con>premium</con>
      <var>customer</var>
    </callop>
    <callop>
      <con>regular</con>
      <var>product</var>
    </callop>
  </hn>
. . .
</rfml>
```

Via `rfml2ruleml`, this result is transformed back to the original RuleML version (again except for whitespace handling).

As a digression from the roundtrip, the RFML version, via the XML-to-HTML stylesheet `rfmlsp` [http://www.relfun.org/rfml/rfmlsp.xsl], leads to a font-and-color-coded HTML Prolog version, rendered in a browser as follows (the first line is our sample rule):



Using something like "`File | Save As... | Text`", an ASCII Prolog version can be obtained, which permits space-separated phrases as constant symbols.

The inverse transformation of (ordinary) ASCII Prolog to RFML markup, hence to RuleML, requires a separate parser/generator as (Lisp-)implemented in Relfun [http://www.relfun.org/]: there are no ASCII-to-XML stylesheets.

Exporting from RuleML: With the example in mind, let us now inspect our XML-to-XML stylesheets in some more detail, starting with the two initial transformation rules of `ruleml2rfml`.

The first stylesheet rule ('template') matches the document root "/" followed by a `rulebase` element; it 'recursively' maps ('applies') all matching templates to all `rulebase` subelements and generates an `rfml` element from the result:

```
<!-- process rulebase and position fact/imp transformers -->
<xsl:template match="/rulebase">
  <rfml>
    <xsl:apply-templates/>
  </rfml>
</xsl:template>
```

The second template matches a `fact` element, does `apply-templates` over its `_head` in `mode pattop` (for RFML's `operator` `patterns`), and generates an `hn` element from the result:

```
<!-- process fact, transforming it to hn clause without premises -->
<xsl:template match="fact">
  <hn>
    <xsl:apply-templates select="_head" mode="pattop"/>
  </hn>
</xsl:template>
```

Importing to RuleML: We now also look at the two initial transformation rules of `rfml2ruleml`.

The first template is the exact 'inverse' of the first `ruleml2rfml` template:

```
<!-- process rfml program and position hn transformer -->
<xsl:template match="/rfml">
  <rulebase>
    <xsl:apply-templates/>
  </rulebase>
</xsl:template>
```

The second template matches an `hn` element, and, for the fact case, expects one child; a `fact` element with a `_head` role is generated from the result of an `apply-templates` over the `pattop`:

```
<!-- process hn clause, that is a fact, ... -->
<xsl:template match="hn">
  <xsl:choose>
    <xsl:when test="count(child::*)=1">
      <fact>
        <_head>
          <xsl:apply-templates select="pattop"/>
        </_head>
      </fact>
    </xsl:when>
      . . .
  </xsl:choose>
</xsl:template>
```

XML comments also explain the other important XSLT-template rules in both stylesheets. The XSLT-stylesheet engines used for the translations were Xalan [http://xml.apache.org/xalan-j/] as well as Cocoon [http://xml.apache.org/cocoon/].

8 Conclusions

This paper has prepared semantic-pragmatic XML-Prolog transfers using the XML syntax. In particular, an XML-RDF-integrating data model was used as the basis for DTD and XML Schema definitions of Prolog on the Web. Meanwhile, a new foundation for the WWW and the Semantic Web has been proposed on the basis of a related XML-RDF-integrating data model [PSS02].

The current paper has also started these semantic-pragmatic transfers in the XML-to-Prolog direction by specifying (semantics-preserving) XSLT-based translators between RuleML and RFML. It is now possible to build and exchange Prolog knowledge bases on the Web (using Web technologies). The online RACSA application [http://www.dfki.de/racsa/] can be viewed as a start.

We have thus also demonstrated the practical use of W3C technologies such as DTDs and XML Schema as well as XSLT. On the other hand we have hinted at a few problems with the current version of XML Schema (1.0), hoping that a more expressive version will be available in the future (maybe aligned with RELAX NG [http://www.oasis-open.org/committees/relax-ng/]). Functional and logic programmers may also have wondered whether the XSLT-based translators could be specified more concisely in a future version of XSLT. Perhaps XQuery [http://www.w3.org/TR/query-semantics/] can act as an incentive towards formal semantics of XML queries and transformations.

Corresponding transfers in the Prolog-to-XML direction have already started with Mandarax/Oryx, j-DREW, and other RuleML engines based on Prolog technology [http://www.dfki.de/ruleml/#Engines]. It may also be useful to implement (orthogonalized versions of) XML Schema, XSLT, etc. in (RuleML and) Prolog to explore the limits of expressiveness that can be realized formally and efficiently. A side-effect of this work could be translators between the XML syntax and a more concise Prolog syntax for W3C languages such as XML Schema and XSLT.

References

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web: A New Form of Web Content that is Meaningful to Computers Will Unleash a Revolution of New Possibilities. *Scientific American*, 284(5):34–43, May 2001.

[Bol01] Harold Boley. A Web Data Model Unifying XML and RDF. Draft http://www.dfki.de/~boley/xmlrdf.html, September 2001.

[Bol02] Harold Boley. Cross-Fertilizing Logic Programming and XML for Knowledge Representation. In Rolf Grütter, editor, *Knowledge Media in Healthcare: Opportunities and Challenges*, pages 38–56. Idea Group Publishing, Hershey, London, Melbourne, Singapore, Beijing, 2002.

[BTW01] Harold Boley, Said Tabet, and Gerd Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proc. Semantic Web Working Symposium (SWWS'01)*, pages 381–401. Stanford University, July/August 2001.

[Cla99] James Clark. XSL Transformations (XSLT) Version 1.0. Recommendation REC-xslt-19991116, W3C, November 1999.

[Fal01] David C. Fallside. XML Schema Part 0: Primer. Recommendation REC-xmlschema-0-20010502, W3C, May 2001.

[KL89] Michael Kifer and Georg Lausen. F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 134–146, Portland, Oregon, 31 May–2 June 1989.

[LS99] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation REC-rdf-syntax-19990222, W3C, February 1999.

[PSS02] Peter F. Patel-Schneider and Jérôme Siméon. Building the Semantic Web on XML. In Ian Horrocks and James A. Hendler, editors, *The Semantic Web – ISWC 2002, First International Semantic Web Conference*, pages 147–161, June 2002.

[SD01] Michael Sintek and Stefan Decker. TRIPLE—An RDF Query, Inference, and Transformation Language. In Dietmar Seipel, editor, *Deductive Databases and Knowledge Management (DDLP'2001)*, October 2001. Workshop in the Stream "Content Management" of INAP2001.