SWAD-Europe: WP5.2 Extracting Semantics from XML Structure

Project name:
Semantic Web Advanced Development for Europe (SWAD-Europe)
Project Number:
IST-2001-34732
Workpackage name:
Workpackage description: 5: Integration with XML Technology
Workpackage description:
☞http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-5.html
Deliverable title:
5.2 Extracting Semantics from XML Structure
URI:
Authors:
☞Stephen Buswell, Stilo Technology
Abstract:
"Is it possible to get the RDF out of the XML ?"
Demonstrator RDF Extractor System
Status:
Complated report 2003-10-17.
Comments on this document should be sent to ☞Stephen Buswell.

Contents

Introduction

This paper is part of SWAD-E Workpackage 5.2. It describes a demonstrator system for extracting RDF-type data from XML document instances.

It further looks at the various uses and interpretations of choices of XML structure in XML encodings, and the implications for tools and techniques aiming to extract meaning from XML syntax and structure. This question could be characterised as "Is it possible to get the RDF out of the XML ?", although in fact the issue is rather more general than that.

We reach the conclusion that RDF-type data can be extracted, but only where certain assumptions are made about the intention of the original author of the schema/DTD, and that in general these cannot alwaysbe justified without some explicit supporting indication from the document/schema creator. We make some recommendations for further work pursuing these ideas

The Problem

The problem targetted was to demonstrate an open-source system for extracting RDF-style data from schema-based XML input documents. There are many different paradigms for expressing semantic information in a schema-based XML encoding - this issue is discussed later in this paper. As our test case, we selected the 'Layered Normal Form', one of the XML Normal Forms [XNF]. This model is also called the Alternating Normal Form.

The Layered Normal Form implements a striped syntax. The outermost element represents an object, the first level child elements represent properties, the second level children again represent objects and so on. Leaf nodes represent properties and their values. In this Normal Form, XML attributes are not used

Sample Input XML Document in Normal Form Structure

```
<PurchaseOrder>
      <orderDate>1999-10-20</orderDate>
      <shipTo>
       <Address>
```

```
      <country>US</country>
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <city>Mill Valley</city>
      <state>CA</state>
      <zip>90952</zip>
     </Address>
    </shipTo>
    <billTo>
     <Address>
      <country>US</country>
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <city>Old Town</city>
      <state>PA</state>
      <zip>19144</zip>
     </Address>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <item>
     <Item>
      <partNum>872-AA</partNum>
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</comment>
     </Item>
    </item>
    <item>
     <Item>
      <partNum>926-AA</partNum>
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
     </Item>
    </item>
   </PurchaseOrder>
```

Solution Architecture

The demonstrator solution uses a combination of ☞schematron and ☞XSLT. We use ☞XT as the underlying XSLT engine.

We decided to implement a two-stage process. The first stage is a schematron processor to convert the data in the XML source document into a source-schema-neutral format. The second stage is an XSLT stylesheet which converts the neutral format into the RDF-XML format.

There were a number of reasons for this approach:

- We gained experience with different technologies for performing this kind of data extraction
- The architecture permits the re-use of the second stage with different first stage processors from different schema paradigms
- Processing the RDF syntax in schematron turned out to be more difficult than in XSLT, particularly the handling of whitespace

Stage 1 Schematron Processor

Essentially the schematron processor implements the underlying logic of the Alternate Normal Form by analysing the input document XML tree layer and emitting a sequence of statements in the intermediate language.

Leaf nodes are detected by counting child elements; unique Ids are generated to cater for repeated instances of a given element. Then, depending on the leaf-node-ness and the parity of the nesting level, the processor will emit one of:

- `decl` - an object/class declaration. `decl` has a `subject` (class name) and the autogenerated `subjectID`.
- `decl-p` - a property declaration. `decl-p` has a `predicate` (the property name)
- `prop` - a proposition (a statement that a given object has a particular property). `prop` comes in two forms. For an object property, `prop` has a `subject/subjectID` pair, a `predicate` and an `object/objectID` pair. For a datatype property, `prop` has a `subject/subjectID` pair, a `slot` (datatype property name) and a `value`.

It can be seen that there is no need to reference any of the element names in the input XML, so this processor is completely independent of the schema of the input document document. In addition, the structure could be simply extended using the same model to process deeper levels of nesting if required.

Stage 1 Schematron Processor - Code

```
<schema xmlns="http://www.ascc.net/xml/schematron"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
      <pattern name="wrap1">
          <rule context="/*">
               <report test="count(*) &gt; 0">
               &lt;wrapper&gt;
               </report>
          </rule>
    </pattern>
```

```
     <pattern name="level1">
          <rule context="/*">
               <report test="count(*) &;gt; 0">
               &;lt;decl&;gt;
               &;lt;subject&;gt;<name/>&;lt;/subject&;gt;
               &;lt;subjectID&;gt;<value-of select="generate-id(.)"
/></subjectID>
               &;lt;/decl>
               </report>
          </rule>
    </pattern>

   <pattern>
        <rule context="/*/*">
               <report test="count(*) = 0">
               <prop>
                <subject><value-of select="name(..)"
/></subject>
                <subjectID><value-of
select="generate-id(..)" /></subjectID>
                <slot><name/></slot>
                <value><value-of select="."
/></value>
                </prop>
              </report>

               <report test="count(*) > 0">
               <decl-p>
               <predicate><name/></predicate>
                </decl-p>
              </report>
          </rule>
    </pattern>

     <pattern>
        <rule context="/*/*/*">
               <report test="count(*) > 0">
               <decl>
               <subject><name/></subject>
                <subjectID><value-of
select="generate-id(.)" /></subjectID>
               </decl>
              </report>

            <report test="count(*) > 0">
              <prop>
               <subject><value-of select="name(../..)"
/></subject>
               <subjectID><value-of
select="generate-id(../..)" /></subjectID>
               <predicate><value-of select="name(..)"
/></predicate>
               <object><name/></object>
               <objectID><value-of
select="generate-id(.)" /></objectID>
               </prop>
               </report>

        </rule>
    </pattern>

    <pattern>
        <rule context="/*/*/*/*">
               <report test="count(*) = 0">
              <prop>
               &lt;subject><value-of select="name(..)"
/></subject>
               <subjectID><value-of
select="generate-id(..)" /></subjectID>
               <slot><name/></slot>
              <!--  <value-of select="name(..)" /><value-of
select="generate-id(..)" /> has-<name/>  "<value-of select="." />" -->
               <value><value-of select="."
/>lt;/value>
              </prop>
               </report>
         </rule>
    </pattern>

      <pattern name="wrap2">
          <rule context="/*">
               <report test="count(*) > 0">
               </wrapper>
               </report>
          </rule>
    </pattern>

</schema>
```

Output File from Stage 1 Schematron Processor

The intermediate XML is a simple declarative statement-oriented format intended to represent the extracted semantics of the input document in a form which is both human-readable and readily translatable to RDF. The intermediate format conforms to the schema attached in ☞Appendix A.

```
<wrapper>

 <decl> <subject> PurchaseOrder </subject>
<subjectID> N1 </subjectID> </decl>

 <prop> <subject> PurchaseOrder </subject>
 <subjectID> N1 </subjectID> <slot> orderDate
</slot> <value> 1999-10-20 </value> </prop>

 <decl-p> <predicate> shipTo </predicate> </decl-p>

 <decl-p> <predicate> billTo </predicate> </decl-p>

 <prop> <subject> PurchaseOrder </subject>
 <subjectID> N1 </subjectID> <slot> comment
 </slot> <value> Hurry, my lawn is going wild! </value>
</prop>

 <decl-p> <predicate> item </predicate> </decl-p>

 <decl-p> <predicate> item </predicate> </decl-p>

 <decl> <subject> Address </subject> <subjectID>
N8 </subjectID> </decl>

 <prop> <subject> PurchaseOrder </subject>
 <subjectID> N1 </subjectID> <predicate> shipTo
 </predicate> <object> Address </object>
<objectID> N8 </objectID> </prop>

 <decl> <subject> Address </subject> <subjectID>
N32 </subjectID> </decl>

 <prop> <subject> PurchaseOrder </subject>
 <subjectID> N1 </subjectID> <predicate> billTo
 </predicate> <object> Address </object>
<objectID> N32 </objectID> </prop>

 <decl> <subject> Item </subject> <subjectID> N59
</subjectID> </decl>

 <prop> <subject> PurchaseOrder </subject>
 <subjectID> N1 </subjectID> <predicate> item
 </predicate> <object> Item </object> <objectID>
N59 </objectID> </prop>

 <decl> <subject> Item </subject> <subjectID> N80
</subjectID> </decl>

 <prop> <subject> PurchaseOrder </subject>
 <subjectID> N1 </subjectID> <predicate> item
 </predicate> <object> Item </object> <objectID>
N80 </objectID> </prop>

 <prop> <subject> Address </subject> <subjectID>
 N8 </subjectID> <slot> country </slot><value> US
 </value> </prop>

 <prop> <subject> Address </subject> <subjectID>
 N8 </subjectID> <slot> name </slot><value> Alice
 Smith </value> </prop>

 <prop> <subject> Address </subject> <subjectID>
 N8 </subjectID> <slot> street </slot><value> 123
 Maple Street </value> </prop>

 <prop> <subject> Address </subject> <subjectID>
 N8 </subjectID> <slot> city </slot><value> Mill
 Valley </value> </prop>

 <prop> <subject> Address </subject> <subjectID>
 N8 </subjectID> <slot> state </slot><value> CA
 </value> </prop>

 <prop> <subject> Address </subject> <subjectID>
 N8 </subjectID> <slot> zip </slot><value> 90952
 </value> </prop>

 <prop> <subject> Address </subject> <subjectID>
 N32 </subjectID> <slot> country </slot><value>
 US </value> </prop>
```

```
<prop> <subject> Address </subject> <subjectID>
N32 </subjectID> <slot> name </slot><value>
Robert Smith </value> </prop>

<prop> <subject> Address </subject> <subjectID>
N32 </subjectID> <slot> street </slot><value> 8
Oak Avenue </value> </prop>

<prop> <subject> Address </subject> <subjectID>
N32 </subjectID> <slot> city </slot><value> Old
Town </value> </prop>

<prop> <subject> Address </subject> <subjectID>
N32 </subjectID> <slot> state </slot><value> PA
</value> </prop>

<prop> <subject> Address </subject> <subjectID>
N32 </subjectID> <slot> zip </slot><value> 19144
</value> </prop>

<prop> <subject> Item </subject> <subjectID> N59
</subjectID> <slot> partNum </slot><value>
872-AA </value> </prop>

<prop> <subject> Item </subject> <subjectID> N59
</subjectID> <slot> productName </slot><value>
Lawnmower </value> </prop>

<prop> <subject> Item </subject> <subjectID> N59
</subjectID> <slot> quantity </slot><value> 1
</value> </prop>

<prop> <subject> Item </subject> <subjectID> N59
</subjectID> <slot> USPrice </slot><value>
148.95 </value> </prop>

<prop> <subject> Item </subject> <subjectID> N59
</subjectID> <slot> comment </slot><value>
Confirm this is electric </value> </prop>

<prop> <subject> Item </subject> <subjectID> N80
</subjectID> <slot> partNum </slot><value>
926-AA </value> </prop>

<prop> <subject> Item </subject> <subjectID> N80
</subjectID> <slot> productName </slot><value>
Baby Monitor </value> </prop>

<prop> <subject> Item </subject> <subjectID> N80
</subjectID> <slot> quantity </slot><value> 1
</value> </prop>

<prop> <subject> Item </subject> <subjectID> N80
</subjectID> <slot> USPrice </slot><value> 39.98
</value> </prop>

<prop> <subject> Item </subject> <subjectID> N80
</subjectID> <slot> shipDate </slot><value>
1999-05-21 </value> </prop>

</wrapper>
```

Stage 2 XSLT Translator XML to RDF

Each `decl`, `decl-p` and `prop` complex element in the intermediate form translates to a single RDF statement. The XSLT stylesheet eliminates superfluous whitespace from the input and builds unique instance names from the class/property names and generated IDs. A statement is then emitted in RDF XML syntax expressing the logical content of the input element.

Stage 2 XSLT Translator - Code

```
<xsl:transform version="1.0"
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns:xdr="urn:schemas-microsoft-com:tensing-data"
               xmlns:type="urn:schemas-microsoft-com:datatypes"
               xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
               xmlns="http://www.w3.org/TR/xhtml1/strict"
               xmlns:msxsl="urn:schemas-microsoft-com:xslt"
               xmlns:po="http://www.example.org/PO"
               >


<!-- Output method ... -->

<xsl:output method="html"  />
```

```
<!-- Global PARAMs -->

<!-- Template rules -->

<xsl:template match="/">
<xsl:apply-templates select="//wrapper" />
</xsl:template>



<!-- Root rule -->
<xsl:template match="wrapper">
<rdf:RDF xml:base = "http://www.example.org/PO">
 <xsl:apply-templates/>
</rdf:RDF>
</xsl:template>

<xsl:template match="decl">
        <xsl:variable name = "su1"><xsl:value-of
 select="subject"/></xsl:variable>
        <xsl:variable name = "su2"><xsl:value-of select=
"translate($su1, ' ', '')"/></xsl:variable>
        <xsl:variable name = "si1"><xsl:value-of
select="subjectID"/></xsl:variable>
        <xsl:variable name = "si2"><xsl:value-of select=
"translate($si1, ' ', '')"/></xsl:variable>
  <xsl:element name="rdf:Description">
     <xsl:attribute name="rdf:type">po:<xsl:value-of select =
"$su2"/></xsl:attribute>
     <xsl:attribute name="rdf:about">po:<xsl:value-of select =
"$su2"/><xsl:value-of select = "$si2"/></xsl:attribute>
  </xsl:element>
</xsl:template>

<xsl:template match="decl-p">
        <xsl:variable name = "su1"><xsl:value-of
select="predicate"/></xsl:variable>
        <xsl:variable name = "su2"><xsl:value-of select=
"translate($su1, ' ', '')"/></xsl:variable>
  <xsl:element name="rdf:Description">
     <xsl:attribute name="rdf:about">po:<xsl:value-of select =
"$su2"/></xsl:attribute>
     <xsl:attribute name="rdf:type">rdfs:Property</xsl:attribute>
  </xsl:element>
</xsl:template>


<xsl:template match="prop">
 <xsl:variable name = "su1"><xsl:value-of
select="subject"/></xsl:variable>
 <xsl:variable name = "su2"><xsl:value-of select=
"translate($su1, ' ', '')"/></xsl:variable>
 <xsl:variable name = "si1"><xsl:value-of
select="subjectID"/></xsl:variable>
 <xsl:variable name = "si2"><xsl:value-of select=
"translate($si1, ' ', '')"/></xsl:variable>
 <xsl:variable name = "sup">po:<xsl:value-of select=
"$su2"/><xsl:value-of select= "$si2"/></xsl:variable>

<!-- <xsl:value-of select = "$su2"/><xsl:value-of select = "$si2"/> -->

<!-- -->
<xsl:choose>

  <xsl:when test = "slot">
          <xsl:variable name = "sl1"><xsl:value-of
select="slot"/></xsl:variable>
          <xsl:variable name = "sl2">po:<xsl:value-of
select="normalize-space($sl1)"/></xsl:variable>
          <xsl:variable name = "vl1"><xsl:value-of
select="value"/></xsl:variable>
          <xsl:variable name = "vl2"><xsl:value-of
select="normalize-space($vl1)"/></xsl:variable>

          <xsl:element name="rdf:Description">
            <xsl:attribute name="rdf:about"><xsl:value-of select =
"$sup"/></xsl:attribute>
            <xsl:element name = "{$sl2}">
            <!--  <xsl:attribute name =
"rdf:resource"><xsl:value-of select = "normalize-space($vl1)"/></xsl:attribute> -->
              <xsl:attribute name = "rdf:resource"><xsl:value-of
select = "$vl2"/></xsl:attribute>
            </xsl:element>
          </xsl:element>
  </xsl:when>

  <xsl:when test = "predicate">
          <xsl:variable name = "pr1"><xsl:value-of
select="predicate"/></xsl:variable>
```

```
          <xsl:variable name = "pr2">po:<xsl:value-of
select="normalize-space($pr1)"/></xsl:variable>

        <xsl:variable name = "ob1"><xsl:value-of
select="object"/></xsl:variable>
        <xsl:variable name = "ob2"><xsl:value-of select=
"translate($ob1, ' ', '')"/></xsl:variable>
        <xsl:variable name = "oi1"><xsl:value-of
select="objectID"/></xsl:variable>
        <xsl:variable name = "oi2"><xsl:value-of select=
"translate($oi1, ' ', '')"/></xsl:variable>
        <xsl:variable name = "obp">po:<xsl:value-of select=
"$ob2"/><xsl:value-of select= "$oi2"/></xsl:variable>

          <xsl:element name="rdf:Description">
            <xsl:attribute name="rdf:about"><xsl:value-of select = "$sup"/></xsl:attribute>
            <xsl:element name = "{$pr2}">
              <xsl:attribute name = "rdf:resource"><xsl:value-of select = "$obp"/></xsl:attribute>
            </xsl:element>
          </xsl:element>
  </xsl:when>


</xsl:choose>
<!-- -->

</xsl:template>

</xsl:transform>
```

RDF Output from Stage 2 XSLT Translator

The generated RDF has been tested using the online RDF validator at W3C (☞http://www.w3.org/RDF/Validator).
A generated SVG image showing the graph structure of the RDF is attached in Appendix B.

RDF Output from Stage 2 – Code

```
 <rdf:RDF xml:base="http://www.example.org/PO"
 xmlns:msxsl="urn:schemas-microsoft-com:xslt"
 xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:type="urn:schemas-microsoft-com:datatypes"
 xmlns:xdr="urn:schemas-microsoft-com:tensing-data"
 xmlns="http://www.w3.org/TR/xhtml1/strict">

 <rdf:Description rdf:type="po:PurchaseOrder"
rdf:about="po:PurchaseOrderN1" xmlns=""></rdf:Description>

 <rdf:Description rdf:about="po:PurchaseOrderN1"
 xmlns=""><po:orderDate rdf:resource="1999-10-20"
 xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:orderDate></rdf:Description>

 <rdf:Description rdf:about="po:shipTo" rdf:type="rdfs:Property"
xmlns=""></rdf:Description>

 <rdf:Description rdf:about="po:billTo" rdf:type="rdfs:Property"
xmlns=""></rdf:Description>

 <rdf:Description rdf:about="po:PurchaseOrderN1"
 xmlns=""><po:comment rdf:resource="Hurry, my lawn is going wild!"
 xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:comment></rdf:Description>

 <rdf:Description rdf:about="po:item" rdf:type="rdfs:Property"
 xmlns=""></rdf:Description>

 <rdf:Description rdf:about="po:item" rdf:type="rdfs:Property"
xmlns=""></rdf:Description>

 <rdf:Description rdf:type="po:Address" rdf:about="po:AddressN8"
xmlns=""></rdf:Description>

 <rdf:Description rdf:about="po:PurchaseOrderN1"
 xmlns=""><po:shipTo rdf:resource="po:AddressN8"
 xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:shipTo></rdf:Description>

 <rdf:Description rdf:type="po:Address" rdf:about="po:AddressN32"
xmlns=""></rdf:Description>

 <rdf:Description rdf:about="po:PurchaseOrderN1"
 xmlns=""><po:billTo rdf:resource="po:AddressN32"
 xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:billTo></rdf:Description>
```

```
<rdf:Description rdf:type="po:Item" rdf:about="po:ItemN59"
xmlns=""></rdf:Description>

<rdf:Description rdf:about="po:PurchaseOrderN1"
xmlns=""><po:item rdf:resource="po:ItemN59"
xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:item></rdf:Description>

<rdf:Description rdf:type="po:Item" rdf:about="po:ItemN80"
xmlns=""></rdf:Description>

<rdf:Description rdf:about="po:PurchaseOrderN1"
xmlns=""><po:item rdf:resource="po:ItemN80"
xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:item></rdf:Description>

<rdf:Description rdf:about="po:AddressN8" xmlns=""><po:country
rdf:resource="US" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:country></rdf:Description>

<rdf:Description rdf:about="po:AddressN8" xmlns=""><po:name
rdf:resource="Alice Smith" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:name></rdf:Description>

<rdf:Description rdf:about="po:AddressN8" xmlns=""><po:street
rdf:resource="123 Maple Street" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:street></rdf:Description>

<rdf:Description rdf:about="po:AddressN8" xmlns=""><po:city
rdf:resource="Mill Valley" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:city></rdf:Description>

<rdf:Description rdf:about="po:AddressN8" xmlns=""><po:state
rdf:resource="CA" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:state></rdf:Description>

<rdf:Description rdf:about="po:AddressN8" xmlns=""><po:zip
rdf:resource="90952" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:zip></rdf:Description>

<rdf:Description rdf:about="po:AddressN32" xmlns=""><po:country
rdf:resource="US" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:country></rdf:Description>

<rdf:Description rdf:about="po:AddressN32" xmlns=""><po:name
rdf:resource="Robert Smith" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:name></rdf:Description>

<rdf:Description rdf:about="po:AddressN32" xmlns=""><po:street
rdf:resource="8 Oak Avenue" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:street></rdf:Description>

<rdf:Description rdf:about="po:AddressN32" xmlns=""><po:city
rdf:resource="Old Town" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:city></rdf:Description>

<rdf:Description rdf:about="po:AddressN32" xmlns=""><po:state
rdf:resource="PA" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:state></rdf:Description>

<rdf:Description rdf:about="po:AddressN32" xmlns=""><po:zip
rdf:resource="19144" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:zip></rdf:Description>

<rdf:Description rdf:about="po:ItemN59" xmlns=""><po:partNum
rdf:resource="872-AA" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:partNum></rdf:Description>

<rdf:Description rdf:about="po:ItemN59"
xmlns=""><po:productName rdf:resource="Lawnmower"
xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:productName></rdf:Description>

<rdf:Description rdf:about="po:ItemN59" xmlns=""><po:quantity
rdf:resource="1" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:quantity></rdf:Description>

<rdf:Description rdf:about="po:ItemN59" xmlns=""><po:USPrice
rdf:resource="148.95" xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:USPrice></rdf:Description>

<rdf:Description rdf:about="po:ItemN59" xmlns=""><po:comment
rdf:resource="Confirm this is electric"
xmlns:po="http://www.example.org/PO"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:comment></rdf:Description>

<rdf:Description rdf:about="po:ItemN80" xmlns=""><po:partNum
rdf:resource="926-AA" xmlns:po="http://www.example.org/PO"
```

```
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:partNum></rdf:Description>

 <rdf:Description rdf:about="po:ItemN80"
 xmlns=""><po:productName rdf:resource="Baby Monitor"
 xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:productName></rdf:Description>

 <rdf:Description rdf:about="po:ItemN80" xmlns=""><po:quantity
 rdf:resource="1" xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:quantity></rdf:Description>

 <rdf:Description rdf:about="po:ItemN80" xmlns=""><po:USPrice
 rdf:resource="39.98" xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:USPrice></rdf:Description>

 <rdf:Description rdf:about="po:ItemN80" xmlns=""><po:shipDate
 rdf:resource="1999-05-21" xmlns:po="http://www.example.org/PO"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"></po:shipDate></rdf:Description>

</rdf:RDF>
```

Interoperability

Interoperable computing solutions imply the existence of a sharable ontology, or common set of object semantics. Implementors will still be able to use localized and otherwise customized XML markup languages if they choose, but it should be possible to express and validate the semantics of the design as well as the raw XML syntax. Participants in a partner or network relationship need assurance that transactions will be negotiated meaningfully and correctly because all participants mean the same thing in the use of the interchange markup constructs. But shared ontologies are not the domain of XML. (taken from ☞[XMLSem])

In Search of the Semantics of Structure

In trying to extract RDF information from 'non-semantic' XML structures, we are making an assumption that this information is there to be extracted, ie. that something more than the structural relations between elements can be discovered. In order dec ide whether this assumption is justifyable, we studied in turn the various constructs used in XML to express information about objects, their properties and the relationships between them.

The principal constructs available in XML for expressing objects, properties and relationships are elements, attributes, containment (hierarchy), position with respect to sibling elements, and cross-reference. Schemas and DTDs (although not instances) can have concepts of 'OR'-groups ar alternatives in structural form. We will discuss the various ways in which these constructs can be used or interpreted below.

Containment with the semantics of Property Values and Relations

It seems somehow natural to interpret elements as representing objects, attributes as representing property values, and element containment as stating relationships between the parent and child elements, and in many cases this is reasonable, as below:< /p>

```
<book ISBN = "0-932633-60-9">
  <title>Waltzing with Bears</title>
  <author>Tom De Marco</author>
  <author>Timothy Lister</author>
  <publisher>
    <name>Dorset House Publishing</name>
    <address>
      <street>353 West 12 th Street</street>
      <city>New York</city>
      <zipcode>NY 10014</zipcode>
```

The semantics here are the intuitive interpretation: there is a 'book' object with datatype-properties ISBN (value 0-932633-60-9), title (value Waltzing with Bears), and two instances of the author datatype-property. Furthermore, 'book' has an object-p roperty relationship with the 'publisher' object, which itself has a datatype-property 'name', and so on.

Note also that the ISBN property could have been encoded as a child element rather than attribute without changing the meaning, illustrating the fact that the element/attribute distinction is entirely syntactic and gives us no additional semantic infor mation.

Containment with the semantics of Typing

However, the 'natural' interpretation above does not apply always. Consider the following construct (common in schemas for Purchase Orders, Invoices etc):

```
<Items>
  <pen code = "SKU001" price = "12.75" units = "1" />
  <pencil code = "SKU002" price = "2.75" units = "10" />
</Items>
```

There is no "Items" object in our universe here having a relationship with the 'pen' and 'pencil' objects - if this construction tells us anything, it is something like:

```
pen isA Items-type
pencil isA Items-type
```

An equally common construction is the following:

```
<Items>
```

```
  <item name = "pen" code = "SKU001" price = "12.75" units = "1" />
  <item name = "pencil" code = "SKU002" price = "2.75" units = "10" />
</Items>
```

in which the type-implication is even clearer. (Alternatively, one might interpret this construct as a grouping for processing convenience, with no real semantics whatsoever.)

Containment with the semantics of Grouping

A variation on the above is:

```
<SoldOn date = "20030922">
  <pen code = "SKU001" price = "12.75" units = "1" />
  <pencil code = "SKU002" price = "2.75" units = "10" />
</<SoldOn>
<<SoldOn date = "20030923"> >
  <pen code = "SKU011" price = "15.75" units = "3" />
  <pencil code = "SKU022" price = "4.75" units = "90" />
</<SoldOn>
```

Here "SoldOn" is not an object in our universe, or the type of "pen" and "pencil". It is a grouping construct based on a shared property value of the 'pen' and 'pencil' objects, eg. DateSold = "20030923".

Note that this and the previous example may seem strange from the point of view of information modelling, but they would make perfect sense when looked from the viewpoint of a processing application.

Containment with the constructive semantics

Consider the following construct for building representations of mathematical objects - the first child of the container is the mathematical operator, the remainder are arguments to the . This formalism is used in Content MathML ☞[M athML], and also in OpenMath ☞[OM] (and also lisp, although not with XML syntax).

```
<apply>
  <sin/>
  <ci>x</ci>
</apply>
```

The object represented here is "sin (x)". There is no "apply" object about which we are stating a relationship with the operator "sin" object - unless we regard it as the result of the application of 'sin' to 'x', in which case its relationship to the 'sin' object is purely syntactic (semantically trivial).

The Conventional semantics of position

The example above illustrates another complexity of interpreting semantics from structure - the users of the MathML and OpenMath schemas know (from the textual parts of the standards documents) that convention defines the first child element to be of t ype operator, and the remainder its arguments. However, this important piece of information does not appear anywhere in the schema or instance.

This type of conventional positional significance is common - consider the football score "Oxford 3, Cambridge 0". If this is a football (soccer) score, I can probably deduce that the home team has won. If it is a football (gridiron) score, I can prob ably deduce that the away team has won. The point is that conventionally, the home team score is given first in the UK and second in the USA.

This issue might even be called the 'social semantics' of position and can in fact appear explicitly in schema encodings such as {first name, last name} for personal names. In the UK and the USA, this construct typically has the semantics of {given nam e, family name}. However, in many cultures, such as China and, traditionally, Belgium, the family name is given first.

Explicit References in XML

The explicit referencing mechanism within XML uses the ID/IDREF attribute combination, for example :

```
<meeting>
  <attendees>
    <person id = "mp" name = "Martin Pike" affiliation = "Stilo" />
    <person id = "sb" name = "Stephen Buswell" affiliation = "Stilo" />
  </attendees>

[…]

<actionitems>
    <actionitem idref = "mp" deadline = "20030930" action = "write minutes" />
```

Use of ID establishes uniqueness for references to persons, and the ID/IDREF establishes a relationship between the 'actionitem' object and the 'person' object (although there is no information as to what this relationship represents).

However, probably the most common usage in practice is for bibliographical references:

```
<para> A.A.Milne writes <xref idref = "AAM"/> about Eeyore, a
```

and then at the end of the article:

```
<biblio>
  <bibentry id = "AAM">The House at Pooh Corner, …</bibentry>
```

☞[HPC] The actual semantics of this link are somewhat complex. The relationship established does not involve the 'xref' object - this is a purely syntactic construct; rather the relationship is between the 'bibentry' and the print in the 'para' where the 'xref' was inserted. Note that the relationship is not simply with the 'para' itself, as there could be multiple 'xref's and their individual position would then be significant.

Note that syntax of XML does not allow us to use URIs (used in the OWL/RDF world for unique identifiers) as the values of these ID/IDEFs.

Implicit Reference by Shared Value

An XML fragment extracted from a university admissions database might look something this:

```
<student name = "James Smith">
  <courses-followed>
     <course>101</course>
     <course>103</course>
     <course>107</course>
  </courses-followed>
</student>

[…]

<department name = "Mathematical Sciences">
  <courses-offered>
     <course code = "101" name = "basic algebra"/>
     <course code = "102" name = "number theory"/>
     <course code = "103" name = "elementary quantum dynamics"/>
  </courses- offered>
  <number-of-staff>17</number-of-staff>
  <number-of-students>107</number-of-students>
</department>
```

The '101' in <course>101</course> matches the '101' in <course code = "101" > and is in fact an opaque cross-reference. However, the '107' in <course>107</course> has nothing to do with the '107' in <number-of-students>107</number-of- students>. A human reader would almost certainly realise this immediately. However, given that numerical course identifiers were used, the ID/IDREF mechanism is not available (XML defines these to be of type NMTOKEN ☞[XML]), and there is nothing in the schema or instance languages offering a mechanism to indicate that one match is a cross-reference and the other is a red herring.

The semantics of OR-groups

One construct which occurs in schemas and DTDs (but cannot be inferred directly from looking at the document instances) is the OR-group. From a structural point of view this is clear: exactly one of the options must be selected. Looking at the semantics, there are two quite different usages:

A schema fragment for an *address* element for UK/US addresses might have a choice between *(zipcode | postcode)* child elements. We would expect any address in the US/UK to have exactly one of these, and see them as mutually exclusive properties.

By contrast, a person opening a bank account has to show identification, and a schema for the registration document might have an *Identificiation* element with a choice of *(passport | drivinglicence)* child elements. Here the selection gives which one was shown, but we would probably not see possession of passports and driving licences as mutually exclusive properties (indeed, we might expect most adults to have both).

This construct gives us some members of the value-range of a property, but no additional information about exclusivity.

The Question of Identity

One key issue in interpreting the semantics of a document is deciding whether two items with the same value refer to the same object. Consider the following:

```
<rdfs:Resource rdf:about="http://www.openmath.org/CD#CDType" rdf:type = "rdfs:Class">
  <rdfs:label xml:lang="en">CDType</rdfs:label>
</rdfs:Resource>

<rdfs:Resource rdf:about="http://www.openmath.org/CD#CDType" rdf:type = "rdfs:Class">
  <rdfs:comment>
    Class declaration for use by RDF representations of OpenMath CDs.
  </rdfs:comment>
</rdfs:Resource>
```

We know from the RDF specification that the value of *rdf:about* is a URI, and we know from the URI RFC 2396 ☞[URIR] that if two URI values are the same then they represent the same thing. An RDF reasoner could draw the valid conclusion that the label and the comment relate to the same resource.

Note that even so, this idea should be treated with some caution, as it does not necessarily imply identity in the intuitive sense: for example, the concept denoted could be "The President of the French Republic" which is not a constant value over long timescales.

Comparing string values

If we do not know the string is a URI, or the value of an ID/IDREF cross-reference as discussed above, we can be much less certain.

The question is compounded when considering values representing numerical data, where an interpretation may need to consider any datatype information. The values decimal="10" and octal="10" have the same value as strings, probably the same type (eg. xs i:integer) but have different numerical interpretations. By contrast, (a real example) in <film name="10">, the datatype is (probably) xsd:string.

The values centigrade = "100" and fahrenheit = "100" have the same value as strings, the same numerical interpretation, but different interpretations as temperature values.

Conclusions

Schemas and DTDs model the structure of the documents they define rather than the universe from which the data to populate those documents is taken. As a consequence, structures in an XML document do not necessarily correspond directly (or even at all) to objects and properties in the universe of discourse. Furthermore, the same XML construct (for example containment) may have multiple possible semantic interpretations, and there is nothing in the DTD/Schema languages which allows us to distinguish between these possibilities.

Therefore, in order to infer the semantics from the structure (or extract the RDF from the XML) we have to make the assumptions that the original author of the schema

- thought about his universe in an 'ontological' way
- implemented this view in the schema/DTD.

There are a number of XML formats which implement this approach, for example the XML Normal Forms ☞[XNF] work. We have implemented an RDF extractor for one of these, the Layered Normal Form. This work is described earlier in this paper. When such an XML format is used, the RDF can be successfully extracted with confidence that the semantics are being maintained.

However, if we cannot make such an assumption (and this is the general case), then the results of this mechanical extraction of RDF-style data are harder to interpret from a semantic viewpoint.

Further Work

We considered whether any of the schema annotation mechanisms such as the Schema Adjunct Framework ☞[SAF] would provide the necessary additional information. For two reasons, we decided that this approach was inadequate

- as noted above, constructs in the XML document do not necessarily correspond directly to objects in our universe of discourse
- a schema based approach tells us only about the meaning of documents satisfying this particular schema.

What occurs in practice is that there may be multiple schemas describing the object (there are many schemas for Purchase Orders, for example). Similarly, schemas for different objects have content which relates to common objects in the universe (Purcha se Orders and Invoices, although different documents, share information abouts prices, order numbers and so on).

For these reasons we concluded that the best way forward is to investigate systems and languages external to the schemas and document which support the mapping of objects in an ontology (written for example in RDF/OWL) to information items extracted fr om the XML documents. This activity will be undertaken as part of WP6.

References

- [XMLSem]☞XML and Semantics http://www.oasis-open.org/cover/xmlAndSemantics.html
- [MathML]☞MathML http://www.w3.org/TR/MathML2/
- [OM]☞OpenMath http://www.openmath.org/
- [XNF]☞XML Normal Forms http://www.ltg.ed.ac.uk/~ht/normalForms.html
- [SAF]☞Schema Adjunct Framework http://www.extensibility.com/saf/spec/
- [URIR]☞RFC 2396 http://www.ietf.org/rfc/rfc2396.txt
- [XML]☞XML http://www.w3.org/XML/
- [HPC]☞The House at Pooh Corner, A. A. Milne, Methuen & Co, London 1928
- [SCHT]☞Schematron http://www.ascc.net/xml/resource/schematron/
- [XT]☞XT http://www.blnz.com/xt/index.html
- [XSLT]☞XSLT http://www.w3.org/Style/XSL/
- [XNF]☞XML Normal Forms http://www.ltg.ed.ac.uk/~ht/normalForms.html

Appendices

Schema for the Intermediate XML Language

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by stephen buswell (Math WG) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="wrapper">
        <xs:annotation>
                <xs:documentation>outermost container element</xs:documentation>
        </xs:annotation>
        <xs:complexType>
                <xs:choice>
                        <xs:element name="decl">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="subject"/>
                                                <xs:element name="subjectID"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                        <xs:element name="decl-p">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="predicate"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                        <xs:element name="prop">
                                <xs:complexType>
                                <xs:sequence>
                                <xs:element name="subject"/>
                                 <xs:element name="subjectID"/>
                                 <xs:choice>
                                <xs:sequence>
                                 <xs:element name="slot"/>
                                <xs:element name="value"/>
                                </xs:sequence>
                                <xs:sequence>
                                <xs:element name="predicate"/>
                                <xs:element name="object"/>
                                <xs:element name="objectID"/>
```

```
                                        </xs:sequence>
                                        </xs:choice>
                                        </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:choice>
                            </xs:complexType>
                        </xs:element>
                    </xs:schema>
```

SVG Graph for the Sample RDF Output

☞SVG Graph for the Sample RDF Output