

Thinking XML: Enrich Schema definitions with SKOS

Learn how to attach concepts from the business problem domain to XML applications

Skill Level: Intermediate

[Uche Ogbuji \(uche@ogbuji.net\)](mailto:uche@ogbuji.net)

Partner

Zepheira, LLC

11 Nov 2008

The things in schemata (people, places and things) are inextricably tied to how people describe them, and this is the key to alignment of business with technology. One of the most important things an XML schema designer can do is express this connection clearly. SKOS, a language well known as a component of DITA, is a very useful means for such expression. Learn how to enrich schema definitions with SKOS definitions.

The main theme in this *Thinking XML* column has been semantic transparency, to clarify the meaning of constructs in XML schemata and instances. I've talked about bottom-up approaches to semantic transparency, where you define terms and concepts at the discrete level (data dictionaries, in effect), independently of their documents and schemata, and then you can apply these broadly. Industries have built up many initiatives for bottom-up semantic transparency, some of which have survived better than others. In this column I've discussed ISO Basic Semantics Register (BSR), RosettaNet Dictionaries, ebXML Core Components, Universal Data Element Framework (UDEF), ISO 15022 in the financial services space, and more. But no major industry initiatives tackle terminology to drive schema development in most applications of XML. More often you need to define your own specialized data dictionaries.

Frequently used acronyms

- DITA: Darwin Information Typing Architecture
- URI: Uniform Resource Identifier
- W3C: World Wide Web Consortium
- XML: Extensible Markup Language
- XSL: Extensible Stylesheet Language
- XSLT: XSL Transformations

More and more, architects realize that data dictionaries alone are not enough to support richer information integration. In XML documents you need to refer to people, places, and things with inter-relationships ranging from general to more specific, part and kind, and synonym and antonym. You need to describe connections to geographical points, to key times and dates, to policies, and to business rules. Sometimes you expand your scope beyond your specialized information space toward larger industry conventions. These details are why Semantic Web technology is such a good fit for supporting XML development, and it makes sense to start with the most modest, simplest Semantic Web technologies. Simple Knowledge Organization System (SKOS) is just such a technology, presently in the last call stage of the Working Draft process, but already well understood, implemented, and discussed. SKOS has unfortunately lost some of its simplicity in the latest drafts, as its committee ties it to the far more complex Web Ontology Language (OWL), but it's still quite useful if you ignore some of the more arcane flourishes. It at least provides the word-relationship aspect of connecting basic meaning relationships of terms, and this is a great first step for enriching XML schemata.

From concept to expression

It's best to capture concepts as early as you can in the development process. During the analysis for a new application or integration of existing ones, you should write down the main concepts, preserving as much of the original context as possible. The Turtle syntax for Resource Description Framework (RDF) is a useful way to capture this information in a format that non-technical users can review. Such close cooperation between the developers and the business interest is key to bolster the value of information managed in XML. In this article, I'll use the scenario of a snowboard manufacturer, Fluffy Boards, developing a format to capture marketing information about a new model of board called the Cumulus. Listing 1 is a subset of a SKOS definition of relevant concepts in Turtle syntax.

Listing 1. SKOS definition of key concepts for snowboard marketing information in Turtle format

```

@prefix skos: <http://www.w3.org/2004/02/skos/core#>.
@prefix f: <http://www.fluffyboards.com/vocabulary#>.

f:product
  a skos:Concept;
  skos:prefLabel "product";
  skos:altLabel "merchandise item";
  skos:definition "Item developed for sale by Fluffy Boards.".

f:snowboard
  a skos:Concept;
  skos:prefLabel "snowboard";
  skos:altLabel "deck";
  skos:definition "Deck to be mounted with bindings for riding on snow.";
  skos:broader f:product.

f:endorsement
  a skos:Concept;
  skos:prefLabel "endorsement";
  skos:altLabel "formal thumbs-up";
  skos:definition "Formal statement of approval of the product.";
  skos:broader f:review.

f:review
  a skos:Concept;
  skos:prefLabel "review";
  skos:altLabel "product opinion";
  skos:definition "Statement of opinion of a product.".

f:customer
  a skos:Concept;
  skos:prefLabel "customer";
  skos:definition "Person or group engaged by Fluffy Boards in the purchase process.".

```

See [Resources](#) for full introductions to SKOS, but briefly, the first two lines are Turtle versions of namespaces declarations. The `f` namespace is proprietary to Fluffy Boards. The next block defines the concept `f:product`, specifying a preferred label as well as optional alternates. `skos:definition` is a brief but precise description of the concept. The `skos:broader` property is a way to organize related concepts. "Person" is a broader concept than "boy", and "boy" a narrower concept than "person". You will generally deal with XML versions of SKOS vocabularies in schemata. Once you have a Turtle version, you can use tools to automatically convert it to RDF/XML (and vice versa). Listing 2 is the XML form of [Listing 1](#).

Listing 2. SKOS definition of key concepts for snowboard marketing information in RDF/XML form

```

<?xml version="1.0"?>
<rdf:RDF xmlns:f="http://www.fluffyboards.com/vocabulary#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#">
  <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#snowboard">
    <skos:prefLabel>snowboard</skos:prefLabel>
    <skos:altLabel>deck</skos:altLabel>
    <skos:definition>Deck to be mounted with bindings for riding on snow.
  </skos:definition>
  <skos:broader>
    <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#product">

```

```

    <skos:prefLabel>product</skos:prefLabel>
    <skos:altLabel>merchandise item</skos:altLabel>
    <skos:definition>Item developed for sale by Fluffy Boards.</skos:definition>
  </skos:Concept>
</skos:broader>
</skos:Concept>
<skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#endorsement">
  <skos:prefLabel>endorsement</skos:prefLabel>
  <skos:altLabel>formal thumbs-up</skos:altLabel>
  <skos:definition>Formal statement of approval of the product.</skos:definition>
  <skos:broader>
    <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#review">
      <skos:prefLabel>review</skos:prefLabel>
      <skos:altLabel>product opinion</skos:altLabel>
      <skos:definition>Statement of opinion of a product.</skos:definition>
    </skos:Concept>
  </skos:broader>
</skos:Concept>
<skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#customer">
  <skos:prefLabel>customer</skos:prefLabel>
  <skos:definition>Person or group engaged by Fluffy Boards in the purchase process.
  </skos:definition>
</skos:Concept>
</rdf:RDF>

```

Into the schema

Once you have the conceptual base expressed in a semi-formal language such as SKOS, it's a matter of your own good XML design sense and creativity to incorporate it into a schema. Suppose one use of XML in the Fluffy Boards marketing department is to store reviews for each board. They might come up with a W3C XML Schema (WXS) definition as in Listing 3.

Listing 3. The W3C XML Schema definition

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="http://www.fluffyboards.com/marketing/"
  xmlns:f="http://www.fluffyboards.com/vocabulary#"
  xmlns:m="http://www.fluffyboards.com/marketing/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#">
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2000/10/xml.xsd"/>
  <xs:element name="snowboard">
    <xs:annotation>
      <xs:appinfo>
        <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#snowboard">
          <skos:prefLabel>snowboard</skos:prefLabel>
          <skos:altLabel>deck</skos:altLabel>
          <skos:broader rdf:resource="http://www.fluffyboards.com/vocabulary#product"/>
        </skos:Concept>
      </xs:appinfo>
      <!-- from skos:Concept/skos:definition -->
      <xs:documentation>Deck to be mounted with bindings for riding on snow.
      </xs:documentation>
    </xs:annotation>
  </xs:element>

```

```

</xs:annotation>
<xs:complexType>
  <xs:sequence>
    <xs:element name="name" />
    <xs:element ref="m:description" />
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="m:review" />
    </xs:choice>
  </xs:sequence>
  <xs:attribute ref="xml:base" />
  <xs:attribute name="id" use="required" type="xs:anyURI" />
</xs:complexType>
</xs:element>
<xs:element name="review">
  <xs:annotation>
    <xs:appinfo>
      <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#review">
        <skos:prefLabel>review</skos:prefLabel>
        <skos:altLabel>product opinion</skos:altLabel>
      </skos:Concept>
    </xs:appinfo>
    <xs:documentation>Statement of opinion of a product.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="m:source" />
      <xs:element name="content" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="source">
  <xs:annotation>
    <xs:appinfo>
      <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#customer">
        <skos:prefLabel>customer</skos:prefLabel>
      </skos:Concept>
    </xs:appinfo>
    <xs:documentation>Person or group engaged by Fluffy Boards in the purchase process
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="description">
  <xs:complexType>
    <xs:attribute name="purpose" use="required" type="xs:anyURI" />
  </xs:complexType>
</xs:element>
</xs:schema>

```

The constructs that come directly from the SKOS are in bold font. Notice that I import `xml.xsd` so that I can use the `xml:base` attribute. Besides SKOS annotations, another useful Semantic Web practice in XML schema design is to use URIs for IDs. XML Base allows you to unambiguously abbreviate such URIs. The `snowboard` element is attached to the abstract concept of the same name through expression within the schema annotation. I also lift the SKOS concept description into the `xs:documentation` element, which helps smooth integration with WXS-aware tools. Notice that there isn't always a one-to-one correlation between a SKOS concept and a construct in the XML vocabulary. Some of the schema elements have no attached concepts, and some of the concepts don't appear in the schema. The `review` element attaches to the concept of the same name, but its `source` child attaches to the `customer` concept, and in this way you have an

implicit rule that sources of reviews are customers. I recommend also explicitly expressing such rules from concept bases, but you get a sense from this example how expressive it can be to just attach concepts to schemata.

Sample instance

Listing 4 is a sample instance of the schema in [Listing 3](#).

Listing 4. Sample instance based on schema in Listing 3

```
<?xml version="1.0" encoding="UTF-8"?>
<snowboard xmlns="http://www.fluffyboards.com/marketing/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.fluffyboards.com/marketing/ file:listing3.xsd"
  xml:base="http://www.fluffyboards.com/marketing/"
  id="boards/cumulus">
  <name>Cumulus</name>
  <description purpose="unsolicited-blurb"/>
  <review>
    <source>Uche Ogbuji</source>
    <content>The Cumulus floats in powder like a dream. My favorite snowboard.
    </content>
  </review>
</snowboard>
```

Automating the connection

One problem with the approach in the [previous section](#) is that the conceptual information is duplicated in schemata, which can lead to skew as one or the other is updated. A better way is to use links from the schema to the conceptual information. You can always automate merging this information whenever that's useful. Listing 5 is a variation on [Listing 3](#) using links to the SKOS rather than copying constructs.

Listing 5. WXS that uses links into a SKOS document

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="http://www.fluffyboards.com/marketing/"
  xmlns:f="http://www.fluffyboards.com/vocabulary#"
  xmlns:m="http://www.fluffyboards.com/marketing/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#">

  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2000/10/xml.xsd"/>

  <xs:annotation>
    <xs:appinfo>
      <owl:imports rdf:resource="fluffy-marketing.skos"/>
    </xs:appinfo>
  </xs:annotation>
```

```

<xs:element name="snowboard">
  <xs:annotation>
    <xs:appinfo>
      <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#snowboard"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name"/>
      <xs:element ref="m:description"/>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="m:review"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute ref="xml:base"/>
    <xs:attribute name="id" use="required" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>

<xs:element name="review">
  <xs:annotation>
    <xs:appinfo>
      <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#review"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="m:source"/>
      <xs:element name="content" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="source">
  <xs:annotation>
    <xs:appinfo>
      <skos:Concept rdf:about="http://www.fluffyboards.com/vocabulary#customer"/>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

<xs:element name="description">
  <xs:complexType>
    <xs:attribute name="purpose" use="required" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>
</xs:schema>

```

Notice my improvised use of `owl:imports` to formalize the incorporation of the SKOS concepts into the schema. To illustrate how easy it is to traverse these links and automate generating the fuller form in [Listing 5](#) from the shorter form in [Listing 3](#), I wrote a little XSLT to do the job. Listing 6 isn't meant to be robust (it won't handle arcane RDF constructs, nor even simple ones such as `rdf:ID` and `xml:base`), and if you do end up using SKOS a lot in XML schema design, you'll want to have some proper RDF tools handy. But Listing 6 illustrates the idea, and handles the examples in this article.

Listing 6. XSLT to expand simple SKOS references in WXS

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
>

  <xsl:variable name="skosfile"
    select="document(/xs:schema/xs:annotation/xs:appinfo/owl:imports/@rdf:resource)"/>

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="skos:Concept">
    <xsl:variable name="uri" select="@rdf:about"/>
    <xsl:copy>
      <xsl:apply-templates select="$skosfile//skos:Concept[@rdf:about=$uri]/*"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="xs:annotation">
    <xsl:variable name="concept" select="xs:appinfo/skos:Concept/@rdf:about"/>
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
    <xsl:if test="$concept">
      <xs:documentation>
        <xsl:value-of
          select="$skosfile//skos:Concept[@rdf:about=$concept]/skos:definition"/>
        </xs:documentation>
      </xsl:if>
    </xsl:template>

    <xsl:template match="skos:definition"/>

  </xsl:stylesheet>

```

Wrap up

Attaching SKOS concepts to constructs in richer schema languages such as RELAX NG and Schematron (both of which I personally prefer) is even easier than for WXS. In such cases you can put the SKOS elements in-line wherever it makes sense, thanks to its separate namespace. What you gain, regardless of schema language, is not a magic wand that suddenly makes all XML documents transparent to every person and application. What you get is what I call an anchor in this column—a hand-hold that gives people the clues they need to direct integration and to improve data quality. Schema annotations connected to overall information sharing tools such as wikis make it possible for all the people involved in an interest to collaborate and contribute regardless of their technical ability. SKOS is a good language to express the technical substance of such interchange. In this article, you learned how easy it is to attach domain concepts to XML schema definitions.

Resources

Learn

- [The W3C SKOS Simple Knowledge Organization System Primer](#): Learn about SKOS from the main source.
- [SKOS](#): Keep in touch with developments in knowledge organization systems (KOS) such as thesauri, classification schemes, subject heading systems and taxonomies within the framework of the Semantic Web.
- [Subject classification with DITA and SKOS](#) (Erik Hennum, Robert Anderson, and Colin Bird; developerWorks; October 2005): Learn about SKOS in the context of DITA, which can be used for organization of large content libraries. Be careful because this article uses a slightly out of date version of SKOS.
- [Use data dictionary links for XML and Web services schemata](#) (Uche Ogbuji, developerWorks May 2004): Learn more about the basic technique elaborated in this article, including its usage in RELAX NG and Schematron in the author's tip.
- [Create a maintainable extensible XML format](#) (Eric de Jonge and Sally Slack, developerWorks, August 2008): Learn the basic use of WXS schema annotations.
- [New to XML](#) page: Check out the XML zone's updated resource central for XML. Readers of this column may be too advanced for this page, but it's a great place to get your colleagues started. All XML developers can benefit from the XML zone's [coverage of many XML standards](#).
- Review related, earlier installments of this column by Uche Ogbuji:
 - [State of the art in XML modeling](#) (developerWorks, March 2005): Discover what developers need to know about the various approaches to semantic transparency.
 - [Schema standardization for top-down semantic transparency](#) (developerWorks, April 2005): Explore the state of the art in XML modeling that includes reuse of models designed by others.
 - [Schema annotation for bottom-up semantic transparency](#) (developerWorks, May 2005): Push schemata beyond syntax into semantics.
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [XML technical library](#): See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks, including

[previous installments](#) of the *Thinking XML* column.

- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- The [technology bookstore](#): Browse for books on these and other technical topics.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.

Get products and technologies

- [RDF Validator and Converter](#): Use Joshua Tauberer's tool for quick interchange between RDF formats.
- [IBM trial software for product evaluation](#): Build your next project with trial software available for download directly from developerWorks, including application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- [Participate in the discussion forum for this content.](#)
- [the *Thinking XML* forum](#): Post your comments on this article, or any others in this column.
- [XML zone discussion forums](#): Participate in any of several XML-related discussions.
- [developerWorks XML zone: Share your thoughts](#): After you read this article, post your comments and thoughts in this forum. The XML zone editors moderate the forum and welcome your input.
- [developerWorks blogs](#): Check out these blogs and get involved in the [developerWorks community](#).

About the author

Uche Ogbuji

Uche Ogbuji is a [partner](#) at [Zepheira, LLC](#), a solutions firm specializing in the next generation of Web technologies. Mr. Ogbuji is lead developer of [4Suite](#), an open source platform for XML, RDF, and knowledge-management applications, and its successor [Akara](#). He is also lead on the [Jacqard](#) agile methodology for team Web development, and the [Versa](#) RDF query language. He is a Computer Engineer and writer born in Nigeria, living and working in Boulder, Colorado, USA. You can find more about Mr. Ogbuji at his Weblog [Copia](#).

Trademarks

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, WebSphere, and pureXML are trademarks of IBM Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.