

XML Schemas for Publishing Applications

Alan R. Houser <arh@groupwellesley.com>

Abstract

Many in the "document-centric" (as opposed to "data-centric") XML community have been skeptical about the value of XML Schema for publishing-oriented XML applications. Many in the community believe that XML Schema, given its complexity as compared to XML DTDs, is of questionable value for supporting XML-based publishing applications. For example, although XML Schema offers far richer data typing than do XML DTDs, many consider the data typing capabilities of DTDs to be sufficient for XML-based publishing applications.

Nonetheless, the XML Schema Recommendation has significant value for XML-based publishing applications. This presentation provides an overview of the benefits of XML Schema over XML DTDs for those applications, and addresses real and perceived disadvantages of XML Schema compared to XML DTDs.

1. Why Document People Like DTDs

Many of my colleagues in the publishing field have been skeptical of XML Schema. (This paper refers to the W3C May 2001 XML Schema Recommendation). XML DTDs (document type definitions) have well-served the needs of the publishing community. XML DTDs offer several real and perceived advantages to the publishing community.

- **Simplicity.** XML DTD syntax is reasonably easy to learn and reasonable easy to teach.
- **Familiarity.** Because XML DTD syntax was borrowed from SGML, DTDs were in use well before the arrival of XML.
- **Sufficiency:** Since publishing applications are typically concerned with pushing strings of characters to a page, the Web, or other publishing medium, the data typing and content modeling capabilities of XML DTDs are sufficient for publishing applications.

Likewise, many of my publishing colleagues have been skeptical of the XML Schema effort, for the following reasons:

- **Complexity.** XML Schema is difficult to learn, and certainly difficult to create by hand.
- **Verbosity.** Even if you could create an XML Schema by hand, the resulting schema is always far larger than its equivalent DTD.
- **Irrelevance.** XML Schema was obviously designed to support data-centric applications. Why would a publishing application need more than 30 pre-defined data types?

I was formerly of the opinion that XML DTDs meet the needs of publishing applications. However, as I've studied XML Schema, I've come to be intrigued by their capabilities. I propose that we examine whether XML DTDs are truly sufficient for modern publishing applications, and then consider the capabilities of XML Schema.

2. A Facetious Example

Imagine the ingredients list in a cookbook recipe. A typical ingredients list might look like this:

- 1 cup flour
- 2 cups sugar
- 1 teaspoon salt

We might represent this ingredients list as the following XML document instance:

```
<?xml version="1.0"?>
<ingredientslist>
  <item>
    <amount>1</amount>
    <unit>cup</unit>
    <ingredient>flour</ingredient>
  </item>
  <item>
    <amount>2</amount>
    <unit>cup</unit>
    <ingredient>sugar</ingredient>
  </item>
  <item>
    <amount>1</amount>
    <unit>teaspoon</unit>
    <ingredient>salt</ingredient>
  </item>
</ingredientslist>
```

Consider an XML DTD that describes our ingredients list. This DTD would look something like the following:

```
<!ELEMENT ingredientslist (item+)>
<!ELEMENT item (amount, unit, ingredient)>
<!ELEMENT amount (#PCDATA)>
<!ELEMENT unit (#PCDATA)>
<!ELEMENT ingredient (#PCDATA)>
```

Nothing is particularly wrong with this DTD. However, it does not specify that the contents of `<amount>` must be a number, nor does it specify any legal values for `<unit>` or `<ingredient>`. The following document would validate without errors against this DTD:

```
<?xml version="1.0"?>
<ingredientslist>
  <item>
```

```
<amount>fizzblop</amount>
<unit>pooldish</unit>
<ingredient>sludge</ingredient>
</item>
</ingredientslist>
```

We would hope that human editors would catch this sort of error. Why can we not, however, use a validation mechanism that automatically ensures the validity of the `<amount>`, `<unit>`, and `<ingredient>` elements? This is exactly the kind of richer validation that XML Schema can provide.

3. Advantages of XML Schema

Let's consider several of the major advantages of XML Schema over XML DTDs, particularly with respect to publishing-oriented applications. Here we will consider four relevant areas: data typing (including regular expression support), content modeling, namespace support, and extensibility mechanisms.

3.1. Stronger Data Typing

XML DTDs allow element content to be constrained to a string value (`PCDATA`), child elements, or a mix of both. The only data type supported by XML DTDs for element content is a character string.

XML Schema permits far greater flexibility in constraining the data type of element content. The XML Schema Recommendation specifies more than 30 data types that can be used to constrain element (and attribute) content. These data types include not only string types, but numeric, time, and boolean types.

Furthermore, XML Schema provides extensibility mechanisms, including the ability to extend and restrict built-in types, for defining customized data types.

3.2. Extensible Data Typing through Regular Expressions

XML Schema supports regular expressions for defining the type of element content. This capability is particularly useful and powerful for defining and validating customized data types. For example, perhaps your document instances include part numbers, possibly of the form `xxx-yyy` (three digits, followed by a dash, followed by four digits). You can create an XML Schema data type to validate the contents of elements that contain your organization's part numbers. The following XML Schema fragment defines a simple type `PartNumber` that accomplishes this:

```
<xsd:simpleType name="PartNumber">
  <xsd:restriction base="xsd:string">
    <xsd:length value="8"/>
    <xsd:pattern value="\d{3}-\d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

As our documents become the basis for an increasing number of document-based applications – applications that, for example, might allow us to order a part by clicking on a part number in a document – this sort of validation becomes critical.

3.3. Better Content Modeling

XML DTDs provide very limited content modeling capabilities. Elements can occur in sequence or within OR'ed groups. Elements or groups may be either required or optional, and repeatable or singular.

Content modeling provided by XML DTDs is particularly limited for mixed content models – elements which contain a combination of child elements and character data. In mixed content models, the order and number of character data and elements cannot be specified.

XML Schema surpasses the content modeling capabilities of XML DTDs. The capabilities of XML Schema content modeling include the following:

- Ability to constrain the order and number of child elements in mixed content models.
- Ability to define a minimum and/or maximum number of consecutive instances of an element (via `minOccurs` and `maxOccurs`).
- Ability to define groups of elements, of which all must appear but in unspecified order.
- Ability to define groups of elements, of which any one element must appear.

3.4. Namespace Support

As XML publishing-related vocabularies such as scalable vector graphics (SVG), mathematics (MathML), and multimedia (SMIL) mature, document instances are ever more likely to include elements from more than one XML namespace. Although it is possible to modify an XML DTD to validate documents that contain multiple vocabularies, this is essentially a hack. XML Schema provides namespace support and mechanisms for validating document instances that include elements from multiple namespaces.

3.5. Extensibility Mechanisms

XML DTDs support extensibility through a parameter entity mechanism – essentially a string substitution. XML Schema provides the following extensibility mechanisms, for extending not only definitions within a schema, but also for extending entire schema instances:

- Ability to derive type definitions by extension and restriction.
- Ability for the schema author to control (i.e. constrain or forbid) derivations on particular types.

- Ability to include external schema documents in an XML Schema.
- Ability for an XML instance document to reference multiple schema.

4. What XML Schema does not Provide

It's a common misconception that XML Schema is intended to serve as a replacement for XML DTDs. This is not true. XML Schema only provides a formalization of the structure of an XML document. XML Schema does not provide a mechanism for defining any type of entities (internal, external parsed, unparsed), or notations. Although XInclude and XLink provide some of the functionality provided by DTDs in the space of external entities and notations, no other mechanism exists for defining internal and parameter entities. These items must be defined in a DTD or in the internal subset section of the XML declaration in each instance document.

5. Revisiting our Example

Let's again consider our simple recipe DTD. Let's create an XML Schema instance to more tightly constrain the contents of our ingredients list. Here we will look at the process for creating a simple XML Schema.

We start by declaring our simple element types - elements that have no child elements or attributes, and whose content is one of the simple types defined by the XML Schema Recommendation. From our DTD, three elements fit the bill: `<amount>`, `<unit>`, and `<ingredient>`. `<amount>` will be a decimal number, while `<unit>` and `<ingredient>` are strings. We can define these elements as follows:

```
<xsd:element name="amount" type="xsd:decimal">
<xsl:element name="unit" type="xsd:string">
<xsl:element name="ingredient" type="xsd:string">
```

Here we have defined `<unit>` and `<ingredient>` as a simple element of type string. But because there are only a handful of possible measurement units, we wish to constrain the value of `<unit>` to be a legal measure. Just assume that all measures in our ingredients list will be `cup`, `teaspoon`, `tablespoon`, or `none`. (We will use `none` for a pure number of items, with no unit; for example, `5 carrots`.) Likewise, we may wish to constrain the contents of `<ingredient>` to a fixed set of ingredients. We can do this as follows:

```
<xsd:simpleType name="unit">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="cup" />
    <xsd:enumeration value="teaspoon" />
    <xsd:enumeration value="tablespoon" />
    <xsd:enumeration value="none" />
  </xsd:restriction>
</xsd:simpleType>
```

This schema fragment uses the `<xsd:restriction>` constraint to specify that element `<unit>` is of type `xsd:string`. The value of `<unit>` is further constrained by the facets (an XML Schema term) defined by the `<xsd:enumeration/>` elements. Each of these elements contains an attribute named `value`, which lists one of the legal string values for the `<unit>` element.

We have defined three of the five elements in your recipe list fragment. Because the remaining two elements, `<item>` and `<ingredientslist>`, have child elements as content, these are, by definition, complex elements. Let's start with `<item>`. This element must contain one or more of the triplet `<amount>`, `<unit>`, `<ingredient>`, in that order. We can set up the order of elements by declaring an element `<item>` of complex type, whose content is constrained by the `<xsd:sequence>` element.

```
<xsd:element name="item">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="amount"/>
      <xsd:element name="unit"/>
      <xsd:element name="ingredient"/>
    </xsd:sequence></xsd:complexType>
</xsd:element>
```

That leaves only one remaining element – `<ingredientslist>`. This element consists simply of a sequence of `<item>` elements. We can define it as follows:

```
<xsd:element name="ingredientslist">
  <xsd:complexType>
    <xsd:element name="item" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:complexType>
</xsd:element>
```

This schema fragment specifies that our `<ingredientslist>` element has only one child, `<item>`. The attribute/value pair `minOccurs="1"` specifies that at least one `<item>` instance must exist; while the attribute/value pair `maxOccurs="unbounded"` specifies that any number of `<item>` instances are allowed.

Here we have our equivalent XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="amount" type="xsd:integer"/>
  <xsd:element name="ingredient">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="flour"/>
        <xsd:enumeration value="salt"/>
        <xsd:enumeration value="sugar"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="ingredientslist">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="item" type="itemType" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

    </xsd:complexType>
</xsd:element>
<xsd:complexType name="itemType">
  <xsd:sequence>
    <xsd:element ref="amount" />
    <xsd:element ref="unit" />
    <xsd:element ref="ingredient" />
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="unit">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="cup" />
      <xsd:enumeration value="cups" />
      <xsd:enumeration value="teaspoon" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
</xsd:schema>

```

Is this XML Schema more verbose than the DTD fragment we considered earlier? Yes. Is it more complex? Arguably. However, remember that this schema is not simply replacing the DTD. This schema constrains the content of the `<amount>`, `<unit>`, and `<ingredient>` elements. This schema provides far better validation capabilities for our instance documents than does the "equivalent" DTD.

6. Issues for Acceptance

Despite the intriguing capabilities presented here, XML Schema may take some time to become popular in the publishing community. In part because the publishing community is dependent on relatively complex, off-the-shelf software, it may be some time before XML Schema support is widely incorporated into tools typically used for XML-based publishing. Until then, maintainers of XML-based publishing solutions must incorporate XML Schema as an add-on to the publishing workflow.

The publishing community is also relatively traditional. SGML DTD-based solutions pre-date XML by more than 10 years. Even with better tools support, the perception is likely to continue that XML Schema were developed for the "data-centric" XML community and are not relevant to the document-centric community. Educating the XML publishing community in the capabilities of XML Schema will speed the acceptance of XML Schema within the community.

7. Conclusion

The XML Schema Recommendation provides many capabilities over XML DTDs, particularly in the areas of stronger data typing, better content modeling capabilities, and improved extensibility mechanisms. Although many perceive that these capabilities are only relevant of data-centric XML applications, XML Schema in fact provides appealing and important capabilities for publishing applications. Barriers to deployment include educating the XML publishing

community on the value of XML Schema, and introduction of XML Schema support in XML publishing tools.

Biography

Alan R. Houser

Principal

Group Wellesley

Pittsburgh

U.S.A.

Email: arh@groupwellesley.com

Alan Houser is a principal partner in Group Wellesley, a Pittsburgh PA-based company that provides documentation and content management consulting services. Houser holds degrees in electrical engineering and professional writing from Carnegie Mellon University, Pittsburgh, PA. He has more than eleven years of computer industry experience.

Houser presents seminars and corporate training courses in SGML and XML-related topics, and is co-author of "XML Weekend Crash Course", published in 2001 by Hungry Minds, Inc. He has designed both SGML and XML-based publishing solutions to support both single-source publishing for multiple audiences and personalized content for specific reader profiles.