Experiences with the Design of the W3C XML Schema Definition Language

(Position paper for the W3C RDF Validation Workshop – 10 & 11 September 2013)

Noah Mendelsohn
Tufts University

XML and RDF are different in many ways, but it is possible that some of the challenges that arose in designing a validation system for XML will have equivalents in the Semantic Web world. The purpose of this paper is to suggest a few such areas for consideration. In some cases, the nature of RDF itself will suggest design approaches different from those used for XML Schema, e.g. that schema information be represented as RDF triples. Indeed, there is no intention here to imply that the solutions adopted in the XML Schema Definition Language (XSD) [Schemas] were in all cases good ones even for XML. Rather, the hope is to point out some general areas that might be considered in building a validation system for the Semantic Web. These suggestions are based on my experiences as a member of the W3C XML Schema working group and co-editor of the schema structures specification, but the opinions in this position paper are mine alone.

Use cases

For XML, we found a strikingly broad range of perspectives on what validation was for and who would use it. Some believed that validation should be a simple binary test to be applied to questionable data: is this XML OK? Others needed to use schemas to drive various software tooling, to guide the import of XML data in to applications such as spreadsheets, to configure prompts for XML forms and to facilitate data binding to languages like C, Java and SQL. There was also a requirement that any DTD be convertible into a schema and a hope, ultimately realized, that XML Schema could provide the type system for the XML Query language.

All of these needs were real at least to some extent. People did build software systems that use XSD for all these purposes. Nonetheless, meeting so many requirements seriously constrained the language design. For example, data binding requires a deterministic parse and type assignment, and DTD emulation requires providing default values for attributes. The resulting language was large and complex.

RELAX NG [RNGCompact] targeted a much smaller set of use cases and created a smaller, simpler language. Experience thus suggests that careful attention be paid to focusing on high value use cases *before* any particular validation technology is chosen.

What gets validated and why?

The most obvious object to be validated in the case XML is an XML document. When the working group dug deeper, we found that the question was more subtle. Defining validation only for entire documents

precluded separate validation use document fragments, which in the case of XML are often constructed and maintained separately.

We thus decided to define *element* validity as the fundamental building block for XSD validation. Because of this, it is possible for a tool like an XML editor to validate pieces of documents as they are input or updated. Nonetheless, this choice causes complications: XML DTDs (and thus XSD) support validation of IDREF attributes that, to be valid, must contain the identifier of some other element *in the document.* An element with such an IDREF may be invalid by itself, but valid in the context of the larger document. Difficult also questions regarding context arose when XSD 1.1 extended XSD to include Schematron-style path-based validation: should a validation path be allowed to "inspect" the parents or siblings of an element? The answer we eventually chose was "no", so path-based validation is mostly context-free. Except for ID/IDREF checking, the validity of an element can usually be checked independent of the context in which it appears*.[1] The choice to define validity provide types for individual elements also facilitated use of XSD as the type system for XML Query.

Furthermore, XML documents are sequences of characters, but the need was seen also to validate information stored in memory as part of XML databases or query systems. The choice was therefore made to define validity as a property of an abstract XML Infoset tree [Infoset]. This choice made XSD more directly applicable to a broader range of use cases, but it also proved very confusing to users who clearly understood what an XML document was but who were less comfortable with the notion of an abstract Infoset.

The RDF community is likely more comfortable than the XML community with the focus on abstraction (triples) as fundamental and the use of serializations (RDF/XML, N3, etc.) as secondary. Nonetheless, it is important to choose carefully the nature of the information to be validated, and the boundaries on that information. For example, is validity a property of a bounded RDF subgraph or of the unbounded graph of information transitively available through hyperlinks? Does the choice matter?

The form of schemas

It's obvious that some choice will have to be made regarding the form of RDF validation schemas. For RDF, the most obvious form is a set of triples expressing the constraints, and from that follows the possibility of writing out schemas in any of the standard RDF serializations such as N3, RDF/XML etc.

Experience with XML schemas suggests that some thought should be given to the choice of serialization. XSD made essentially the analogous choice: I.e., schemas themselves should be serialized as XML documents. Furthermore, the choice was made that explicit markup should be emphasized over simplicity for users. Thus, to declare an element as an integer with the range 1-20 one writes:

---

[1] So-called local-element declarations do provide another context-sensitive declaration mechanism for elements. The point here is independent of such details: deciding what gets validated is an important design decision. In the case of XSD, the decision was to validate elements as opposed to whole documents, and with XML being a tree-structured system, to focus in many but not all cases on context-independent validation of elements.

```
<xsd:element  name="someElement">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="20"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

In fact, the underlying mechanisms of XSD would work equally well with a syntax like:

```
<xsd:element name="someElement" type="xsd:positiveInteger"
             maxInclusive="20" />
```

or even with a non-XML syntax. Indeed, one of the major attractions of RELAX NG over XSD is the availability of the RELAX NG Compact Syntax [RNGCompact], which is perceived as significantly more convenient than its RELAX NG XML format counterpart.

When defining a validation system for RDF, some thought should be given to the balance between use of generic RDF tooling and formats, and creation of formats that will be easy for users to author and maintain.

Combining Schemas

In both XML and RDF, information maintained by different groups tends to be combined for use. Therefore, it can be valuable to provide facilities by which validation schemas can be composed from pieces authored separately. Sometimes such modularity is needed just so that an individual company or organization can manage work on a large project. More challenging is the ability to define rules for dynamically combining validation schemas that may come from a mix of standards organizations, tool vendors, application builders and individual users. At one level, RDF provides straightforward answers: if constraints are expressed as triples, then combining multiple schemas would, most obviously, lead to the union of such constraints.

Maybe or maybe not such straightforward mechanical rules will meet the needs of actual users, but experience suggests that a variety of use cases should be crafted and that any design should be validated against those.

One very controversial question that arose in the design of XML schemas was whether to allow for what was informally called: *action at a distance.* Roughly, this relates to the orthogonality of the specifications for validity. That is, as two or more fragments of validation schemas are combined, can the effect of the specifications in one be altered by the others? One such feature of XSD is the so-called “substitution group”. An element can allow as its children not just particular elements, but also elements known to be in a common “substitution group”. One might in XHTML put in to a substitution group all the elements that are allowed as children of `<td>`, for example. The controversial choice was to allow elements to be added to groups dynamically as schemas are composed, and without changing either the declaration of the group or its explicit uses. Thus, inspection of an element and the declaration used to validate it is not sufficient to determine validity: one must search all the pieces of validation schema that

are composed for the entire document validation to see whether any elements have been added to a group. Details of this mechanism aren't important here: what is useful is the lesson that composition mechanisms are tricky. It can be difficult to strike a balance between mechanisms that are simple and clean architecturally, and that do what users need.

Versioning

Perhaps one of the most important lessons of the XSD experience is that the systems users build will communicate with versions that are both newer and older than the one in hand. Defining an "all or nothing" validity function may be too rigid.

Consider a system that transfers purchase orders, I.e. information about the purchase of one or more items. Each such order includes information about where the items are to be sent, what is to be purchased, how much things cost, and (we'll assume) a phone number to be called in case of trouble. Let's further assume that the first version of the application was built only for use in the United States, but that later versions are upgraded for international use. As part of the upgrade, two changes are made: a country code field is added to the contact phone number, and field is added in which a shipping carrier can be suggested.

When an old copy of the application sends an order to a newer one, the system must tolerate the old style phone numbers and assume a country code of 1 (US). The lack of a suggested shipper can be ignored. More challenging is the question of how an old version of the application receives a new form order: the decision may be made to accept the order and even to record the phone number, but presumably the old version of the application has no way of dialing the country code. *Therefore, it's essential that the entire phone number be marked as suspect (perhaps invalid?) and not for dialing.*

The point of this example is that *validity is not an all or nothing question when different versions of an application communicate.* One of the major enhancements to XSD 1.1 was to allow for content models that would specify in detail information that is truly required, but allow for some additional information to be tolerated, under user control. Furthermore, XSD does more than define validity: a suitably written validator can indicate which pieces of inbound information were validated by the parts of the schema that allow for future growth vs. those defining expected fields. These hints can be useful to an application, e.g. in identifying whether an unexpected country code has been provided.

Again, the details of the requirements for RDF are likely to be different, and details of the likely solutions will almost surely be different. Nonetheless, experience strongly suggests that anticipating versioning is both important and challenging.

Bibliography

[Infoset} *XML Information Set,* http://www.w3.org/TR/xml-infoset/

[RNGCompact] Clark, James, *RELAX NG Compact Syntax Tutorial,* Working Draft 26 March 2003, http://relaxng.org/compact-tutorial-20030326.html

[Schemas] W3C XML Schema Definition Language: Part 1 Structures (http://www.w3.org/TR/xmlschema11-1/) and Part 2 Datatypes (http://www.w3.org/TR/xmlschema11-2/)