How schema-validity is different from being married

C. M. **Sperberg-McQueen**

Abstract

Validation with some schema languages (e.g. XML 1.0 and SGML DTDs) is a black-or-white question: either the document is (wholly) valid or it's not valid (at all). There are no gray areas: a document cannot be “mostly valid” any more than one can be “a little bit married”. The theoretical information content of a validation result in these systems is thus exactly one bit: yes/valid or no/invalid. In practice good validators try to provide a little more information about errors, but quality of error diagnostics varies and error handling is not standardized. In other languages (e.g. XML Schema), validity assessment is designed to provide more than one binary digit of useful information. XML Schema allows various forms of partial validation: Validation can start at an element other than the root. Wildcards can specify that elements they match should not be validated but skipped (aka “black-box processing” — the data must be well-formed XML, but don't look inside). And wildcards can specify 'lax' validation (matching elements must be well-formed XML, and if the schema has declarations for them, they'll be validated, but the absence of declarations doesn't make the container invalid).

In XML Schema, schema-validity is captured by three properties: The first is `[validation attempted]`: Did we try to validate this item? Its values are `full`, `none`, `partial`. The second is `[validity]`: Is the element valid? It takes the values `valid`, `invalid`, `notKnown`. The third is `[schema error code]`: If the item is not valid, then a list of error codes (references to XML Schema validation rules) explaining why.

This paper will talk about why it can be dangerous and unhelpful to reduce validity to a single bit of information, and how it can be helpful to take a richer view of validity as a property with several values, a property not just of the document as a whole but of each element and attribute in the document.

Table of Contents

By providing a formal, machine-checkable definition of document validity, SGML and XML make a significant advance over other methods of representing documents. Within the limits of the expressive power of the schema language used, a document type designer can capture formally many of the constraints which should be imposed on documents of that type. A document type definition using the DTD notation of XML 1.0 or of SGML can require that a purchase order have both a shipping address and a billing address; or that an article have a title; we may require that bullets lists always have at least two items; and so on. XML Schema 1.0 allows us to go further: U.S. zip codes may be required to consist of five numeric digits, or (for ZIP+4) of five digits, a hyphen, and four more digits. We may require that dates fall into a certain range (are we really likely to acquire any new customers born before 1865? probably not), or that URIs in a bibliography be absolute.

XML Schema 1.0 allows us to go beyond SGML and XML DTDs in another way, too. For users of XML 1.0, as for SGML, DTD-validity is a lot like being married. For users of XML Schema, schema-validity can be rather different. This paper is about the difference.

1. All or nothing?

For our purposes, the first thing to note about marriage is that, oversimplifying only slightly, it's an all-or-nothing proposition. Until the presiding judge, priest, minister, or rabbi pronounces the marriage complete using either the traditional formula “I now pronounce you man and wife” or some approximate equivalent, you're not married. After those words are pronounced, you are. There is no in-between state of being partially married, or sort of married, or a little bit married. (I speak, needless to say, of the legal state of marriage, not of the complex psychological states of those getting married, staying married, or not staying married. I'm also not going to address the issues raised by institutions like civil unions, domestic partnership laws, religious traditions other than Judaism and mainstream Christianity, or common-law marriage.)

The state of being married can thus be represented without serious distortion as a pure Boolean variable. You are (`true`), or you're not (`false`), and we are extremely unlikely ever to wish we had a fuzzy category or a percentage scale for the concept, the way we may wish for a fuzzy category for being tall, or being smart, or being rich or poor.

DTDs in SGML and XML 1.0 treat document validity in the same way, as an all-or-nothing Boolean property. The document is valid, or it's not valid. And if it's not valid, many processors will decline to process it, even if nothing they do depends in any way on the document being valid.

Some examples may help make this clearer.

The following document is a valid instance of a schema for simple purchase orders, in which up to ten items may be ordered, each costing no more than $200.

```
<po:purchaseOrder orderDate="2005-11-16"
                  xmlns:po="http://www.example.com/PO1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/PO1 po.xsd"
>
<shipTo>
<name>V. I. Warshawski</name>
<street>7001 N. Mesa</street>
<city>Palo Alto</city>
<state>CA</state>
<zip>94305</zip>
</shipTo>
<billTo>
<name>V. I. Warshawski</name>
<street>7001 N. Mesa</street>
```

```
<city>Palo Alto</city>
<state>CA</state>
<zip>94305</zip>
</billTo>
<items>

<item partNum="07138766000">
<productName>Craftsman 4.5 hp, 22 in. Deck Side Discharge Lawn Mower
</productName>
<quantity>1</quantity>
<USPrice>139.88</USPrice>
</item>

<item partNum="07133048000">
<productName>Black &amp; Decker Grass Catcher</productName>
<quantity>1</quantity>
<USPrice>34.99</USPrice>
</item>

<item partNum="07133117000">
<productName>Classic Accessories Lawn Mower Cover</productName>
<quantity>1</quantity>
<USPrice>24.99</USPrice>
</item>

<item partNum="07165027000">
<productName>Suncast 20 cu. ft. Garden Shed, Vertical</productName>
<quantity>1</quantity>
<USPrice>99.99</USPrice>
</item>

</items>
</po:purchaseOrder>
```

A document identical to the preceding, in which a typo has introduced a comma into one of the `partNum` attributes incorrect, might be detected by a DTD as invalid (or might not, depending on how `partNum` was declared). A document in which eleven items appeared, rather than only ten, would also be invalid. So also would a document in which all the prices erroneously bore dollar signs. And so would the following document, which as the reader may see has nothing whatever to do with purchase orders:

```

<!--* Richard Rodgers and Oscar Hammerstein, Jr., Oklahoma. *-->
<song>
<l>With me, it's all or nothing.</l>
<l>Is it all or nothing with you?</l>
<l>It can't be in between,</l>
<l>It can't be now and then.</l>
<l>No half and half romance will do.</l>
...
</song>
```

There is, in the world of Boolean validity, no formal distinction among these, even though some of them are much more nearly correct than the others. Validation with DTDs (and with most other schema languages) always starts with the outermost XML element, always validates every element and attribute in the document, and always assigns a single Boolean value to the document as a whole: valid or invalid. From a formal point of view, the process of validation is a function mapping an XML document into one bit of information.

In XML Schema, validation is not a Boolean all-or-nothing proposition. There are several ways in which schema-validity assessment using XML Schema differs from validation using DTDs or most other schema languages.

First of all, there is no requirement that validation start at the root. If you have a SOAP message and wish only to validate the payload, ignoring the SOAP envelope and header, you can request that validation, and your software can provide it, without being non-conforming. As a practical matter, identifying a specific element other than the root as the place to start validation will typically require some notation for referring to arbitrary elements, such as the `element()` schema of XPointer, or a pointer to an element in DOM or other programming-language representation of an XML document. Perhaps for this reason, few command line tools actually allow users to exploit this feature of the spec; the facility is more common in schema validators implemements as libraries providing APIs.

Second, the result of validation is not a Boolean value, but a pair of properties in the post-schema-validation information set (PSVI): the `[validation attempted]` property indicates whether the document was validated fully (value `full`), in part (value `partial`), or not at all (value `none`), and the `[validity]` property indicates whether it is valid or not. Possible values for `[validity]` are: `valid`, `invalid`, and `notKnown`. These two sets of values are not independent: the validity of the document must be `notKnown` if validation started somewhere other than the top-level element of the document, and if `[validation attempted]` is `none`, then `[validity]` must be `notKnown`; if the outermost element was not validated at all, we cannot possibly know that the document is `valid` or `invalid`.

Actually, XML Schema defines schema-validity assessment as providing validity information not just about the document as a whole but about each element in the document. The possible outcomes are summarized in the following table (adapted from [Thompson / Tobin / Sperberg-McQueen 2001]).

Validation attempted	**Validity**		
	`valid`	**`invalid`**	**`notKnown`**
`full` (this node and all descendants fully validated)	OK: entire subtree rooted at this element was checked and is valid.	OK: entire subtree was checked; there is an error either here or in some descendant.	*Not possible*: if the entire subtree was checked, we necessarily have a result of valid or invalid.
`partial` (either this node or some descendant was skipped)	OK: this node is locally valid and none of its attributes or children was invalid or missing a required declaration.	OK: this node was validated, and some descendant was skipped. Either this node is locally invalid or it has an invalid descendant.	OK: this node was not locally validated, but one of its descendants was.
`none` (neither this node nor any descendants were validated)	*Not possible*: if the subtree was skipped, we cannot know it's valid.	*Not possible*: if the subtree was skipped, we cannot know it's invalid.	OK: entire subtree was skipped; we don't know whether it would be valid against a declaration or not.

Table 1. Validation outcomes

With validity information attached to each element and attribute, it's possible to have islands of validity in an invalid context, or bits of invalid data intermixed with bits of valid data. Tools that allow the PSVI to be visualized or navigated can make it a little easier to find and fix errors in the document. Some tools have provided element-level validity information for a long time, relying on internal proprietary interfaces to commercial validators to get the information. By standardizing the information in the PSVI, the XML Schema spec makes it a little easier to get to and use for our own purposes.

So the first way that XML Schema validity is different from being married is that it's not a simple yes/no property. You can't be a little bit unmarried, but you can certainly be a little bit invalid.

2. The central registry

A second important aspect of marriage is that it's not just about the couple getting married. Society is involved (if only through the legal system's role in dealing with inheritance and division of property if the marriage is dissolved), a license is required to get married, the marriage can only be performed by licensed individuals, and the event is recorded in a publicly available registry.

Some people behave as if schema-based validation were similarly dependent on a central registry of some kind, in which the authoritative schemas for various application domains might be found. Schema repositories can certainly be useful, and I don't want to disparage them, but it is a fundamental and important fact about XML validation that you get to write your own schema, expressing your own requirements. You may obligate yourself contractually to conform to a particular schema when exchanging data with a particular interchange partner, but in principle decentralization — the freedom to define your own document types — is a fundamental part of XML's appeal.

Decentralization in the use of XML often takes the form of assigning elements and attributes to namespaces. One of the big differences between XML Schema and DTDs is that XML Schema was designed, from the beginning, to support namespaces, while the relationship between DTDs and namespaces has for historical reasons always been somewhat more fraught. The usual way of defining a schema for use with XML Schema software is to write one or more schema documents defining elements, attributes, and types in one or more namespaces. Each schema document defines schema components for just one namespace, but each schema document can refer to components in other namespaces as well.

One reason for the rule that a single schema document can only define schema components for a single namespace is the pragmatic observation that different namespaces frequently have different owners; this way, owners can maintain their own namespaces by maintaining the schema documents that define the namespaces, without getting into each others' hair. Even if two namespaces have the same owner, the fact that two distinct namespaces have been defined is likely in practice to reflect different maintenance policies or some other difference that makes it convenient to have the relevant declarations in different schema documents.

If the owner of a namespace has made relevant schema documents available at the URI which serves as the namespace name, then those documents can be retrieved at validation time.

But the process of schema validity assessment does not assign any more formal authority to the owner of the namespace than to the invoker of the validation process. If you choose to do so, you can supply a locally modified version of a particular schema document and validate the document against that. (Say you want your HTML always to use `div` elements for sections, and you want to flag headings as errors if they appear anywhere except the beginning of a `div`. The W3C schema for XHTML doesn't require this, but you can require it by writing your own schema document for the XHTML namespace. Since you are restricting the namespace rather than extending it, your documents will still be interpretable and processable using the standard schema. Formally, you could extend the namespace as well, but that would lead to interoperability problems and is bad practice.)

As the existence of multiple incompatible formal definitions for the HTML namespace illustrates, a namespace is associated with the assignment of a particular core set of semantics to an XML vocabulary, not necessarily with a unique language in the formal sense, and still less with a unique formal definition of such a language.

Because XML Schema does not require the assumption that a particular namespace has a single schema, or that “the” schema document for a particular namespace can be found at a particular address, namespace owners can provide multiple definitions of their namespace in different schema documents.

In summary: The second important difference between validity and marriage is that it doesn't need to involve other people or centralized authority. It can be between you and your interchange partners. Or you can do it without even involving them, on your own.

3. Forsaking all others?

In one traditional form of the marriage vow, the couple marry each other, explicitly “forsaking all others”.

Some forms of validation involve a similar level of commitment. They make it hard to play around, or to validate a document promiscuously against different schemas.

In relational databases, the schema is a crucial definitional part of the database. No schema, no tables. No tables, no data. You can't build a database without a schema, because without a schema there is no place to put the data.

In SGML, the document type declaration is formally part of the document. If you don't have a document type declaration, you don't have an SGML document. In XML 1.0 and 1.1, the document type declaration is optional, but when it's present it's formally part of the document; there is no convenient way, short of editing the document, to associate it with a different set of declarations. In these systems, the document instance is *structurally* bound to a particular formal definition of the vocabulary.

In practice, some users put the document type declaration (the `<!DOCTYPE ...>`) into one file and the document instance into another, so that they could mix and match at validation time by concatenating the document with one DTD or another at the last possible moment. Users who put the declaration into the same file as the data, on the other hand, were out of luck: neither XML nor SGML foresee the possibility of associating a different set of declarations with the document at validation time.

In XML Schema (and also in some other recent schema languages), the binding between the document and the declarations which are to be used to validate it is much looser. The specification explicitly allows conforming processors to seek and find schema components in any way that seems useful. In practice, this means that if we need a program to validate just one particular kind of document again and again, we can hard-code the schema into it for speed. For a more general-purpose processor, it will typically be more useful to allow the user to specify one or more sources of schema components at validation time. A system may have a local repository of components stored in some convenient binary representation, the user may wish to validate elements and attributes in a particular URI against the schema document found at the the namespace URI, or against a locally cached copy, or against a locally modified version of the schema for that namespace.

That means, in turn, that it's easy to validate a document against more than one schema. If we are about to migrate from version 2.3 of a particular schema to version 2.5, and we have a few tens of thousands of documents, we may wish to find all the documents valid against version 2.3 but not valid against version 2.5, so we can route them to a special upgrad process. It's easy enough, by giving the right runtime parameters to the processor, to validate each document against each of the two schemas and flag the ones which are valid against one but not the other.

Validating the same document now with one, now with a different schema is perfectly legitimate. No adultery is involved.

For convenience, the XML Schema specification provides a `schemaLocation` attribute by means of which the document can indicate where to find suitable schema documents for the namespaces in use. But the meaning of the `schemaLocation` attribute is purely declarative, not imperative. It says that there are schema documents for the relevant namespaces at the locations indicated. It emphatically does *not* constitute a request to validate the document using those schema documents (or any schema documents). Some schema validators do assume by default that they should fetch the schema documents indicated. This is often quite useful. But normally those validators should have a switch to turn off this behavior, because ultimately the control over which schema documents should be used should lie with the user who invokes schema validation, not with the document.

Documents are regularly validated when they cross trust boundaries. In some cases, the trust boundary is minor or even purely notional: I revalidate a document after editing because I don't trust myself not to have unwittingly introduced changes that render the document invalid. (Sometimes in the past I have trusted myself that way, but I've learned better. If a byte of the file changes, it's wise to revalidate.) In other cases, the trust boundary is much more important: the document is transmitted from one organization to another, and the recipient needs to be able to rely on the document's being valid, on the sender's having complied with the contract negotiated between the organizations.

In this case, it is a good idea to validate the document, as part of normal practice, and it would be a grave mistake to rely (as XML 1.0 DTDs and SGML do) solely on information in the document. The recipient is validating the document in part because the document is not trusted: if the processing software relies on the input being valid, handing it invalid data can have exciting and undesirable results. The sender, by implication, is also not fully trusted: if the recipient trusts the sender without any reservations, why is the recipient not taking the sender's word for it that the document is valid? If the document is not trusted to be valid, and the sender is not trusted to have made it valid, then why would we wish to trust the schema documents pointed to by the document as the final arbiters of validity? Surely an adversarial intruder smart enough to intercept the data stream and modify it in some nefarious way is likely also to be smart enough to provide a schema document which says that the modified data stream is valid.

If we are validating because we mistrust the document, it may be essential that we also mistrust the schema documents pointed to by the document. When we do, we *must* have the ability to override the `schemaLocation` attributes in the document being validated and specify exactly which copies of the relevant schema documents we trust. The XML Schema spec gives us this ability by defining `schemaLocation` formally as a hint.

Validating a document against different schemas at different times can also make it easier to check the intermediate results of multi-stage XML processing pipelines.

So the third way that schema validity is different from being married is that the document is not expected to be faithful to a single schema for the duration of its life.

To everything there is a season. There are surely situations in which crisp Boolean distinctions, public registries, and unswerving fidelity to a single person, or idea, or definition of a concept, are all appropriate. Validation is not necessarily one of them; it's a great advantage to have a definition of validity which allows at least a few different shades of gray to be recognized and described, which allows the user who invokes validation to determine what definitions are to apply, and which does not require marrying a document to a single schema for its entire life.

Bibliography

[Biron/Malhotra 2001] Biron, Paul V. Malhotra, Ashok (Eds.), XML Schema Part 2: Datatypes W3C Recommendation, 2 May 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/2001/xmlschema-2/

[Fallside 2001] Fallside, David (Ed.), XML Schema Part 0: Primer W3C Recommendation, 2 May 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/2001/xmlschema-0/

[Grosso et al. 2003] Grosso, Paul et al., ed., XPointer element() Scheme. W3C Recommendation, 25 March 2003. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/xptr-element/

[Thompson et al. 2001] Thompson, Henry S. et al., ed., XML Schema Part 1: Structures W3C Recommendation, 2 May 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/2001/xmlschema-1/

[Thompson / Tobin / Sperberg-McQueen 2001] Henry S. Thompson Richard Tobin C. M. Sperberg-McQueen, XML Schema validation outcomes, 29 May 2001. Available at http://www.w3.org/XML/2001/06/validity-outcomes.html

Biography

C. M. **Sperberg-McQueen**
W3C [http://www.w3.org]
Española
New Mexico
United States of America

C. M. Sperberg-McQueen is a member of the technical staff of the World Wide Web Consortium, an international membership organization responsible for developing Web standards. He co-edited the XML 1.0 specification and the "Guidelines" of the Text Encoding Initiative; he also chairs the W3C XML Coordination Group and serves on the W3C XML Schema Working Group and the W3C XSL Working Group. See also http://www.w3.org/People/cms-mcq/.