XML schema languages

How's a User to choose?

B. Tommie **Usdin**

Abstract

XML schemas specify what tagging is allowed in a set of XML documents. Originally XML had only one way to express these rules; now there are many. What are they? What are the differences among them? When is one more appropriate than another? How should a user (or a project) choose which to use?

Table of Contents

1. Introduction

Schemas are a major component of virtually all XML applications. A schema can be considered:

- the model for one type or class of information — reference books, back transfers, journal articles, credit transactions, drug monographs
- a set of rules describing how documents of that type can be marked up
- an agreement on a common vocabulary (tag set) for an application.

All XML schemas are written in a formal syntax, sometimes called a "constraint language" because they define constraints on what a set of XML documents may contain. Examples of XML schema languages include: DTDs, W3C XML Schema, and RELAX NG.

XML schemas (XML document models) describe rules about elements, attributes, and (in the case of DTDs) entities. The schema may include rules about what may be inside each of these structures, how they relate to each other, and may specify datatypes (valid units of data).

Schemas, and choice of schema language, seems to generate more heat (emotion) than any other aspect of XML application design. There are "experts" who sneer at anyone who still uses DTDs (that's old-fashioned). There are people who know that the only "serious" XML schema language is W3C's XML Schema; you know that because it's called "Schema" and the others are all called something else. There are people who sing the praises of RELAX NG, and suggest that all projects should be based on it because it is based on a clean underlying model. There are people who think it doesn't matter which of those models you use, as long as you use Schematron to make up for the shortcomings they all have. There are people who are making up new constraint languages for particular environments. There are even people who suggest that while DTDs may be unfashionable, they meet the needs of some applications, and thus should be used when possible because they are so my less expensive (in terms of labor and learning) than the newer options.

The only absolute about XML schema selection I want to leave you with, and thus I start with it, is: *Disregard any advice on schema language selection that is not based on a good understanding of your needs, your project, and your application environment.* Just as one-size-fits-all socks don't fit all, one schema language does *NOT* meet all needs.

2. Functions of schemas

Schemas serve several purposes in XML applications. We use schemas to:

- Provide "handles" for data
- Enhance data
- Drive tools
- Protect applications
- Identify unlikely content

2.1. Provide 'handles for data'

By "Provide 'handles' for data" I mean provide the basic infrastructure of XML. XML Schemas allow us to:

- Name the things you can identify (in other words, name the elements and attributes)

- Provides names for these things (the tags)

In XML it is far easier to manipulate data that has "handles" than data that doesn't. So, for example, it may be possible to find the year in an element than contains day, month, and year all mixed together, but it is far easier and more reliable to find the year if it is either identified as an element or an attribute; that is, if there is a handle on it.

2.2. Enhance data

Schemas are sometimes used to add information to XML documents. This happens in two ways:

- Default values. DTDs can provide default values for attributes
- The "Post Schema Validation Infoset", produced by XSD validation software, includes information about data types and validation status. XSD validators can also provide default values for elements and attributes.

2.3. Drive Tools

Many schemas are used to drive tools:

- Context sensitive editors
- Documentation tools
- Databases
- Content management

Many of these tools work best (or work at all) with only one of the XML schema languages. And, in fact, many of then have documentation that is so biased toward one schema language that they often don't even acknowledge that other languages exist.

I find that it is not unusual for organizations, or projects, to made product decisions. especially selecting editing tools, very early in the development process. Sometimes these product work only, or by preference, with one schema language or another. It is not uncommon to find entire project designs, including but not limited to selection of schema language, falling out of one (often poorly informed) tool selection.

2.4. Protect applications

The most important function of schemas in many applications is to protect the system from data that might do harm. There are many ways in which data can harm systems; I have seen typesetting systems crash when presented with a footnote that contained a table which had a footnote. There are other environments where inappropriate content could cause destructive database updates. A schema can ensure that needed content is present and protect from situations that cause harm or crashes.

If schema validation is part of your security system, if one of the functions of the schema is to protect your systems or your database, it is important the you have no surprises in your validation. (This is a warning: some schema validation tools can provide "partial" validation. This is a circumstance in which you don't want partial validation.)

2.5. Identify unlikely content

Schemas are a key tool in identifying and correcting data errors. It is important to distinguish between data that is always an error and data that is highly unlikely but could be acceptable under some circumstances. In my opinion, most of the users who chafe at XML systems, and most failures of XML systems that include end users creating XML content, are caused by schemas that prohibit the unlikely, but occasionally correct content. This is because many system admin-

istrators confuse the "protect the system" function and the "alert QA to unlikely content" function. You want your system to reject harmful content, so data that is invalid according to a protective schema should be rejected.

But, for example, your house style may say that lists should have two or more items. If your schema enforces this rule the technical writer who is trying to develop several parallel chapters, most of which have several "features" in the initial list of features will be forced to commit tag abuse if one chapter is about something that really has only one feature. Similarly, it is reasonable to suggest that if the unit of currency of a price is Yen that the value should probably be greater than 1000. But if you enforce that you will have a problem if someone in your system is selling individual carpet tacks!

3. XML schema languages - 2005

I categorize the XML schema languages in use today into two types:

- Document modeling languages, and
- Business rules checkers.

Document modeling languages are for the creation of a complete representation of the document type. They allow one to specify, for one document type:

- all of the tags allowed (note: the schema specifies the "tag"s; elements are inferred from tagged objects, not specified in the schema)
- what relationships are allowed among the tagged elements (which are inside which, which are required, how many times the may occur, etc.)
- what attributes are allowed on which elements
- what content is allowed in the elements
- what content is allowed in the attributes

In other words, document modeling languages are for modeling a complete document.

Business rules checkers do not, generally, provide an overview of the document type. Instead, they specify particular rules that the documents must obey. These rules can be quite complex, and may apply to as much or as little of a document as needed. business rules checkers, unlike document models, can check interactions among multiple documents and can have access to things outside the XML document such as databases and authority lists. Examples of things that are often specified in business rules checkers are:

- The content of an element must appear in an external table, list, or file
- If the value of an attribute is one of a specified list, a specific element must appear
- the value of this attribute (or content of this element) must be greater than the value of that attribute (or content of that element)
- There may be no more that some number of instances of an element, in any context, throughout the document
- If this optional element is not present, that implied attribute must be present
- If one document contains something, another document must contain something parallel.

Of the XML schema languages available today, I categorize them as follows:

- Document Modeling Languages
  - XML 1.0 DTD (Document Type Definition)
  - XSD (XML Schema Definition Language, i.e. W3C XML Schema)
  - RELAX NG
- Business rules checkers
  - Schematron
  - special purpose languages such as BICS (Business Information Conformance Statement)

3.1. XML Document Type Definitions (DTDs)

DTDs were the first schema language for XML; they were defined in the XML specification. They are primarily about the element structure of the documents, and only incidentally about attributes. DTD validation is all or nothing; either the document is valid or it is not. Validation with a DTD changes the content of the information set. For example, default attribute values are provided. Referential integrity between elements within a document is enforceable through attributes (ID / IDREF). But that's about all you can specify, and check with a DTD.

3.1.1. Objections to DTDs

There are many objections to using DTDs as your XML schema language:

- The syntax is *not* XML document syntax (so you have to learn to read it)
  - it requires a single-purpose processor (XML parser)
- Restricted modeling functions
  - No context-dependent models
  - No AND functionality
- No element data typing (therefore no type validation)
- No inheritance (except by convention `xml:lang`)
- Documentation only as comments
- Lacks other things RDBMS schemas provide: referential integrity, co-occurrence constraints, classing and derivation

That Said, Many Organizations Use DTDs (2005), for good business reasons:

- Easy migration from earlier SGML
- Widely understood
- Easy to learn, read, write
- Tools are/were ubiquitous
- Breaks up the problem of validation
  - use DTDs for what they're good for

  - complete by supplementing with other methods

- Many of the advanced abilities (data typing and data types) are of limited use in processing narrative text

3.1.2. XSD (W3C XML Schema)

XSD seems to be the schema-language-de-jure. Partly, I think this is because it was defined by the W3C, and many people/organizations prefer to use W3C specifications than other specifications. (I think this preference for W3C specifications will fade as there are more and more of them and they are proven to be of uneven quality, and as ISO standardizes more and more XML-related specifications - some of which did not start in the W3C.

XSD provides nearly all of the functionality of DTDs, a great deal of additional functionality, especially datatyping, and uses XML document syntax for the document model. XSD operates over the XML document infoset, not over the XML document.

3.1.2.1. How XSD works

W3C XML Schema does not operate over XML documents (XML documents are strings of characters and entity references). It assumes document has been parsed *into a "tree"* with all entities resolved. This "infoset" is composed of structures (nodes) of named types (strings, numbers, booleans, what-have-you). Validating with a schema produces a *PSVI* ("post-schema-validation infoset"); which consists of trees and their "data content". It can be provided with labels (*type annotations*), defaults (element and attribute) can be added, and validity or invalidity outcomes noted in the PSVI.

3.1.3. RELAX NG

RELAX NG, pronounced as both “relax-N-G” and “relaxing”, was originally defined at OASIS and is now integrated into DSDL [ISO 19575: Document Schema Definition Language] (as Part 2: Grammar-based validation). In RELAX NG modeling is achieved with structural patterns with are not as complex as XSD modeling, go beyond XML DTDs, and can handle some constraints (especially useful in textual documents) that XSD cannot.

RELAX NG oes not change the information set (there are no defaults or types added).It conceives of XML as text, also taking advantage of its tree structure. RELAX NG has two syntaxes (programmatically interchangeable); an XML one like XSD (for processing) a compact syntax (human readable).

3.2. Business Rules checkers

Business rules checkers generally don't model complete documents; they model "checkable" features of documents. Sort of like spot checking; they look for a specific set of conditions and report on their presence or absence. One of the things users tend to like best about business rules checkers is that the person setting up the rules usually specifies the messages that report on the presence or absence of the specified conditions, which means that the messages can be in the vocabulary of the users.

3.2.1. Schematron

Schematron does not define a document model, it defines rules based on path expressions. Originally a "meta-application" of XSLT, Schematron assertions are composed in XPath about what is to be expected or warned against. A generic stylesheet processes the assertions and returns a stylesheet which is run on the instance and generates a validation report. Schematron is currently being abstracted away from XSLT/XPath as part of DSDL (ISO/IEC 19757) (as *Part 3: Rule-based validation - Schematron*).

3.2.2. BCIS (Business Information Conformance Statement)

IBM's BICS (http://www-128.ibm.com/developerworks/xml/library/x-bics20/) an example of a special-purpose validation tool. IBM says: The *Business Information Conformance Statement* specification (sometimes referred to as “BICS”) provides an XML vocabulary framework for declaring a constraint processing model across abstract constraint mechanisms. BICS enables various schema, constraint templates, type systems, etc, to be defined as a concrete constraint mechanism. A BICS XML document instance then contains instances of concrete constraint mechanisms within a constraint processing model, resulting in a comprehensive statement of information constraints.

3.3. Selecting Appropriate Constraint/Validation Languages

There are a number of factors that should be considered in selecting a schema language:

- Nature of application
  - the great data versus narrative text divide
  - just validation or also authoring, query, etc.
- Suitability of particular modeling features (types, typing)
- Tradeoff between expressiveness and overhead / learnability
- Namespaces and inclusion of foreign XML vocabularies
- Ability to use XSLT to extract material from schemas
- Need particular feature in pipeline (PSVI, character entities, etc.)

Many people have argued (loudly) that schema languages should be selected for readability. First of all, I wonder why they think technology decisions should be made based on the convenience of the developers; it seems to me that there are many more important selection criteria. Also, oddly, most of the people arguing for readable schema formats are arguing for XSD - certainly the bulkiest of the XML schema formats and in my opinion the most difficult to read. Yes, tiny demonstrations of XSD are easy to read without learning anything, but when scaled up to a model big enough for real applications most readers would far rather learn a compact syntax than slog through a multi-thousand line XSD.

- RELAX NG has a short form (RELAX NG Compact)
- DTDs are math-like and concise (and I can teach anyone to read DTDs in a half-hour)
- Nobody cares if Schematron is readable (but it is)
- XSD can be viewed through XML document tools (hierarchy diagrams, etc.)

3.4. So, How's a User to Choose?

Don't! Do not choose one schema language. Especially early in a project, don't choose.

Selecting a schema language for an organization or a project early in the planning stages (when I tend to see such decisions) is sort of like deciding, early in the planning stages for a house that you will use nails for this building; no screws, no adhesives, just NAILS!

There is simply no need to do that; use the appropriate language for each function in your application. Don't allow the selection of a particular tool which may only work with one schema language dictate to all of your other tools what language to use.

Remember the many functions of schemas in XML? Why would you think that the same specification language would be the most appropriate way to protect your systems and to identify content that it unusual and may be in error? Or that the same language is appropriate to drive all of your tools, much less drive your tools and validate your content so that other users of the content will know what to expect?

Among the most important decisions in the architecture of an XML application is selection of schema languages. Use the most appropriate language for each function. You don't dance and climb mountains in the same shoes; you shouldn't drive an editor, identify odd content, and agree on data interchange structures with the same language.

Biography

B. Tommie **Usdin**
Mulberry Technologies, Inc. [http://www.mulberrytech.com]
Rockville
Maryland
United States of America

B. Tommie Usdin is President of Mulberry Technologies, Inc., a consultancy specializing in XML and SGML. Ms. Usdin has been working with SGML since 1985 and has been a supporter of XML since 1996. She chairs IDEAlliance's Extreme Markup Language conferences and was co-editor of "Markup Languages: Theory & Practice" published by the MIT Press. Ms. Usdin has developed DTDs, Schemas, and XML/SGML application frameworks for applications in government and industry. Projects include reference materials in medicine, science, engineering, and law; semiconductor documentation; historical and archival materials. Distribution formats have included print books, magazines, and journals, and both web- and media-based electronic publications.