How XML Schemas Simplify Dynamic Content Management

Sandeepan Banerjee, Director, Product Management Oracle Server Technologies.
(Dec 1999)

Introduction

Humans are adept at figuring out the meaning of a piece of content from a few hints gathered from its structure. Going through a pile of papers on my desk, I can, after a brief scan, determine which is a phone bill, which a bank statement, and what different interpretations I need to apply to the rows of each so as to understand its contents. If I am in doubt, the phone company or the bank often helpfully puts headings on columns to help me along -- 'Per Minute Rate', 'Sub-Total', 'Debits' and so on.

Computers, sadly, are in need of even more explicit hints than us in order to be able to decipher what things mean and how they are related. XML, insofar as it makes information self-describing in simple ways, is a major breakthrough in helping programs process content intelligently.

XML has been enthusiastically adopted by the Internet community. It promises to solve two important problems that all web users face: making web sites functionally richer and improving network performance. XML can improve the ability of web sites to make sense of interactions with users in a dynamic manner, by using the markup tags that define semantics of pages. HTML is useful for marking up headlines and fonts, but useless in terms of deciding which piece on a page is a Price. As a result, it is nearly impossible to use pricing information with your searches; your interaction with the web is limited because the language which brings content to you knows very little about the content. HTML is useful for painting documents onto your browser, but not so useful for taking orders, transmitting medical records or monitoring instruments; change the 'Quantity' field in your order, and in order to see the few digits in the 'Total' that change as a result, you ask a burdened server across a slow network to send you a brand-new, generated, graphics-rich page. If structural and semantic information could be added using XML, your browser (or cell-phone, or pocket planner) could do a great deal of processing on the spot. This means you would not have to hit that network or that web server as often. With structural and semantic information in Web pages, the Internet will be faster and friendlier

So far in the evolution of XML, a Document Type Definition (DTD) has been used to describe this structural and semantic information. However, it is increasingly clear that there are certain inherent limitations to DTDs, limitations which would make it hard for XML to assume the increasingly dynamic, data- and content-oriented role thrust upon it. The World Wide Web Consortium (W3C) has decided against pushing the current DTD standards anyfarther, and chartered a new working group to come up with a standard XML Schema that makes it easier for XML to address dynamic content

Shortcomings of DTDs

XML inherited the notion of DTDs from the Standard Generalized Markup Language (SGML). SGML dealt primarily with 'document-like' structures -- theses, books, advertisement-copy, and DTDs worked fine for representing these structures. XML, however, is rapidly expanding to address Business-to-Business and Business-to-Consumer eCommerce, as well as Web-content management. This brings into play dynamic interaction with databases, middleware, interprocess communication -- as well as involvement with new areas such as finance, bio-informatics, telecommunications.

A DTD-based XML document has two structures inside it -- the XML 'instance data' (i.e. the contents marked up with tags just like HTML), and the DTD that describes rules about how the instance data is set up.

For example, an XML instance fragment about a comment might look like:

```
<Comment>Some text to serve as comment </Comment>
```

The DTD for this piece of XML might be:

```
<!ELEMENT Comment (#PCDATA)* >
```

The DTDs have a syntax that is different from the XML instance. Consequently, parsers, tools and programs dealing with DTDs have to implement the specific rules that govern DTDs. Clearly, it would make sense to use the same syntax for DTDs. XML Schema allows models to be written in XML, which lets the same programs that read the data also read the definition of the data:

```
<type name ='Comment'> <ContentModel> <PCData/> </ContentModel>
</type>
```

So, programmers do not have to learn the abstruse #,* &amp;] conventions that the DTDs carried over from SGML. XML Schema can be parsed into a tree of nodes like any other XML document. The XML DOM can then be used to navigate the schema. (The definition property on the IXMLDOMNode interface returns thecorresponding element type or attribute type from the schema.) Consider the following document:

```
<xmlpaper> <speaker> Sandeepan Banerjee </speaker> <topic> How
XML Schemas Simplify Dynamic Content Management </topic> <pages>
7 </pages> </xmlpaper>
```

Suppose we DOM parse the above document, and the variable 'doc' holds a pointer to the parsed document. We can get the 'pages' node as the last child of doc, use the definition property on 'pages' to get the element type for pages, and then access any extended information present in the schema, such as the page limit for papers:

```
var pages = doc.documentElement.lastChild
var pagesType = pages.definition;
 var pageLimit = pagesET.childNodes(1).text;
```

Another problem with DTDs is that they really do not provide support for 'typing' of data -- to a DTD, an XML structure is a string of characters. This worked in the document-centric world of SGML, but poses major problems when you try to model dynamic content that comes from strongly typed systems such as databases. Consider the following:

```
<Speed unit = 'mph'> 140 </Speed>
 <Speed unit = 'mph'> exceeds limit </Speed>
```

Clearly, the two cases would need different logic to handle them. In the absence of typing, an XML parser validating Speed would conclude both were correct, and the burden of adding program logic to deal with each case would fall on the developer. The XML Schema proposal provides for data-type integrity and the maintenance of bound conditions (with some minor scripting support.

DTDs are used today for expressing the relatively simple structures that are found in the document-centric world. There is no support in DTDs for complex structural schemas. DTDs are not well integrated with namespaces. Definition of incomplete constraints on the content of an element type in not possible. Due to a lack of typing, DTDs do not provide the integration of structural schemas with primitive data types. DTDs use content models to specify part-of relations, but they only specify kind-of relations implicitly or informally. XML Schema proposes to address these deficiencies.

Another advantage the XML Schema offers over DTDs is extensibility: XML Schema can be refined and successive schema authors can add their own elements and attributes. You can add additional constraints to the declaration of an element. This extensibility helps XML Schemas to create 'open' content models -- additional elements and/or attributes can be present within an element without one having to declare each and every element in the XML Schema. There are, of course, limitations on what you can do to a schema as part of this open content model. You cannot add/remove elements or attributes that will break the existing content model in some way. For example, if a type is defined as a sequence of elements, valid extensions must preserve that sequence before adding any 'open' content. Undeclared elements can be added only if they belong to a different namespace. There are, of course, cases where an open content model is not desired. You can always override the default and specify the content model as 'closed', in which case any additional elements or attributes will not validate:

```
<type name = 'signature' extendable = 'no'>
```

So we have seen some of the shortcomings of DTDs and how XML Schema proposes to alleviate them. Next, we take a broader look at XML Schema and dynamic content.

XML Schema & Dynamic Content

More and more content -- whether in the worlds of eCommerce, Web publishing and syndication, supervisory control, or application integration --is dynamic. Dynamic content -- especially that driven from databases involving strongly typed data that must be handled in specific ways -- creates many more problems and opportunities for XML than DTDs can handle.

In the Web publishing world, a single page that displays local weather, stock quotes, horoscopes, specific news channels based on user-preferences can involve dozens of queries made to underlying databases and application servers. These queries are made via SQL, the standard object-relational query language, or via some programmatic interface that ultimately calls SQL. Since both SQL and programming languages such as Java are strongly typed, they return information that possesses type, structure, constraints, relationships and so on. So the structural and data typing aspects of XML Schema can help exploit generation of viewable documents from databases.Let us look at some of these aspects.

Simple &Complex Types

XML Schema provides for type definitions that may be both 'simple' and 'complex'. Simple type definitions provide for atomic types, such as 'integer', that can be applied to character data in an instance document, whether it appears as an attribute value or the contents of an element. Complex types, whose expression in XML documents consists of elements with attributes and/or other elements, can also be defined.

Here is a specification for simple types conformant to a recent XML Schema draft:

```
<datatype basetype = NMTOKEN name = NMTOKEN schemaAbbrev =
NMTOKEN schemaName = CDATA> Content: (((minExclusive |
minInclusive) |(maxExclusive | maxInclusive) | (maxAbsoluteValue
, minAbsoluteValue) |encoding | enumeration | length | maxLength
| pattern | period | precision | scale)*) </datatype>
```

For example, you can define a simple datatype positiveInteger, and then define an attribute of this type:

```
<datatype name='positiveInteger' basetype='integer'/>
<minExclusive> 0 </minExclusive> </datatype>

<attribute name='foo'  type='positiveInteger'/>
```

A complex type allows far more. The specification for complex types might look like:

```
<type abstract = "yes" | "no" extendable = "yes" | "no" basetype
= NMTOKEN content = "textOnly" | "mixed" | "elemOnly" | "empty"
final = "yes" | "no" name = NMTOKEN schemaAbbrev = NMTOKEN
schemaName =CDATA> <-- Content: (restrictions , (any | element |
group)* , (attrGroup |attribute)*,anyAttribute)--></type>
```

Now, a complex type 'speed' may be defined as:

```
<type name = 'speed'>
     <element name = 'value'>
           <datatype basetype = 'number'>
                <minInclusive>0</minInclusive>
           </datatype>
      </element>
      <element name = 'unit' type = NMTOKEN'/>
       </element>
</type>

<element name='airspeed' type = 'speed'/>

<airspeed> <value> 123 </value> <unit> knots </unit>
 </airspeed>
```

Clearly, all this is much more sophisticated than DTDs. When dynamic data is generated from a database, it is typically expressed in terms of a database type system. The most popular type systems are SQL:92 and the recently adopted SQL:1999. SQL:92 provides for much richness in data types -- such as NULL-ness, variable precision (e.g. NUMBER(7,2)), check constraints and so on. SQL:1999 adds to the capabilities of database type systems by providing user-defined types, inheritance, references between types, collections of types and so on. XML Schema can capture a wide spectrum of schema constraints that go towards better matching generated documents to the underlying type-system of the data.

The applicability of the rich schema constraints provided by XML Schema is not limited to data-driven applications. There are more and more document-driven applications that exhibit dynamic behavior. A simple example might be a memo, which is routed differently based on markup tags. A more sophisticated example is a technical service manual for an intercontinental aircraft. Based on complex constraints provided by XML Schema, one can ensure that the author of such a manual always enters a valid part-number, and one might even ensure that part-number validity depends on dynamic considerations such as inventory levels, fluctuating demand and supply metrics, or changing regulatory mandates.

Type Hierarchies and Substitutability

In XML Schema, a type definition may identify another type as its supertype. By the provision of an implicit supertype for all types not explicitly identifying a supertype, we get a single type hierarchy. Further, substitutability goes hand-in-hand with inheritance -- any element that is schema-valid as per the type declared for its tag, must also be schema-valid per any of that type's supertypes. That is, if one type is constructed from another by adding new content structure (in the ways permitted by inheritance) then the new type is substitutable for the old.

It is possible to restrict some of the permissions or obligations inherited from a supertype. In such cases, since substitutability must bepreserved, only some kinds of restriction are permissible. If the supertype is a simple type, the restrictions must each narrow the corresponding facets of the inherited type, e.g. by reducing a range or removing members of an enumeration. If the supertype is a complex type, then attributes may be restricted by adding and/or fixing defaults, or by restricting the attribute's simple type. Restricting a content model is also possible.

Any schema implicitly defines an ur-type, which is the basetype for all types, simple or complex, which do not identify an explicit basetype. Here is an example of inheritance:

```
<type name = 'Item'> <element name =  'itemcode' type = 'string'>
</type>

<type name = 'TaxableItem'  basetype = 'Item'> <element
name='taxrate' type = 'real'/> </type>

<element name = 'invoice' <type> ... <any tag='item' type =
'Item'> </type> </element>

<invoice> ... <item> ...</item> </invoice>

<invoice> ... <item xsd:type='TaxableItem'> ... <taxrate> 0.0825
</taxrate> </item> </invoice>
```

A subtype of the Item type is defined, adding a taxrate element to its required content. Two schema-valid instances of an element are declared using a type wildcard with Item as base type; one using that type itself, and therefore not requiring disambiguation, and one using the 'xsd:type' attribute to indicate that it is using the TaxableItem type.

Why are inheritance and substitutability important for dynamic content? DTD mechanisms use content models to specify part-of relations, but they only specify kind-of (i.e. inheritance) relations implicitly or informally. Type-hierarchies with explicit kind-of relations make both the understanding and maintenance of types easier.

In most dynamic content, structures are more complex than those in static content. Experience with other interchange formats such as EDI has shown that even familiar notions like Purchase Order can become quite complex by the time they have been captured in a standardized form. Further, while the basic layout of a Purchase Order is easy to agree upon, most organizations find that their specific case always has some idiosyncratic attribute that requires special processing. Type hierarchies provide a solution is these cases; complexity can be mitigated by creating appropriate basetype abstractions with different levels of inheritance, adding layers of additional meaning. Second, a standard Schema can always to extended or restricted to suit a particular idiosyncracy.

Taken together, complex type structures and substitutability facilitate exchange between loosely coupled applications. Such exchange is currently hampered by the difficulty of fully describing the exchange data model in terms of DTDs; data model versioning issues further complicate interactions in Enterprise Application Integration (EAI) frameworks. When the data model is represented by the more expressive XML Schema definitions, the task of mapping the exchange data model to and from application internal data models is simplified.

Areas of Application

A number of areas in which XML Schema is especially suitable have been touched upon above. Let us recapitulate the major ones:

1. eCommerce: In both Business-to-Business and Business-to-Consumer eCommerce, data is largely dynamic. XML Schema maps robustly to server-driven data models such as SQL:1999, which provides user-defined types, inheritance, references between types, collections of types etc. Libraries of schemas define business transactions within markets and between parties. XML Schema validates a business document, and also provides access to its information set.
2. Web Publication and Syndication: The key to syndication on the web is in highly customizable distribution between publishing and syndication services. Collections of XML documents with complex relations (cross-references, kind-of) among them will be the norm. Protocols such as ICE, which are built on XML, will be able to take advantage of the complex structural aspects of XML Schema.
3. EAI: The essence of an EAI hub-and-spoke architecture is in dynamic data exchange between loosely coupled applications. DTDs cannot fully describe today's exchange data models. XML Schema is a big step forward -- its capability for metadata interchange is not only an EAI simplifier, but also an important optimization technology, as discussed below.

4. Process Control and Data Acquisition: In multi-vendor, distributed systems such as those in plant automation, security, devices, traffic routing etc., XML Schema can aid outgoing and incoming message validity. Controllers can determine which parts of messages they understand: when to ignore information and when to raise errors.

Optimization

Loose coupling between systems leads to all sorts of optimization issues. XML opens up the possibility of dynamic optimizations made on the basis of self-describing servers. A given database can emit a schema of itself to inform other systems what counts as legitimate and useful queries. Any query interface can inspect XML schemas to guide a user in the formulation of queries. There are a number of initiatives in the interchange of metadata (especially for database systems) and in the use of metadata registries to facilitate interoperability of database design, DBMS, query, user interface, data warehousing, and report generation tools. Examples include the ISO 11179 and ANSI X3.285 data registry standards, and OMG's proposed XMI standard. DTDs were inadequate for fully expressing database metadata; the new datatypes proposed by XML Schema, as well as the set of schema constraints the XML Schema will provide will enable dynamic description of database and run-time tuning and optimization of queries.

Another important kind of optimization will help cut down network traffic. Consider a case where someone is trying to book movie tickets using a cellphone. If you go to an on-line ticketing service and ask for all the popular movies showing in your area, you will likely get a long list that does not fit into the form factor of your cellphone display -- that is, you would get a long list that you would have to scroll up and down to inspect. To shorten this list, you need to fine-tune your showtime or your preferred movie title. In order to do this, you would have to send a request across the Web to your ticketing service and wait for a response. If, however, the list of movies in the area had been sent in XML, using XML Schema to express all the constraints about location, number of tickets available, earliest and latest showtimes, then the ticketing service you send a simple Java program along with the data that could refine and sort your choices in milliseconds, without incurring multiple network hits. Since the richness of XML Schema is comparable to programming languages, you can exploit additional processing of richly described structures at various clients. This helps dynamic content to be used with greater efficiency across scarce network resources.

Conclusion

The XML Schema proposal currently under deliberation by W3C adds significantly to the current DTD mechanism. XML Schemas can be used for constraining document structure (elements, attributes, namespaces) as well as content (datatypes, entities, notations); the datatypes themselves can either be primitive (such as bytes, dates, integers, sequences, intervals) or be user-defined (including ones that are derived from existing datatypes and which may constrain certain properties -- range, precision, length, mask -- of the basetype.) Application-specific constraints and descriptions are allowed. XML Schema provides inheritance for element, attribute, and datatype definitions. Mechanisms are provided for URI references to facilitate a standard, unambiguous semantic understanding of constructs. The schema language provides for embedded documentation or comments. The overall type system maps very well to databases and the object-relational SQL:1999 standard, promising significant improvements in dealing with dynamic data generated from database systems and loosely coupled EAI frameworks.